Retrieves the version of BASSenc that is loaded.

DWORD BASS_Encode_GetVersion();

Return value

The BASSenc version. For example, 0x02040103 (hex), would be version 2.4.1.3

BASS_CONFIG_ENCODE_CAST_TIMEOU' config option

The time to wait to send data to a cast server.

BASS_SetConfig(BASS_CONFIG_ENCODE_CAST_TIMEOUT, DWORD timeout

);

timeout The time to wait, in milliseconds.

Remarks

When an attempt to send data is timed-out, the data is discarded. <u>BASS_Encode_SetNotify</u> can be used to receive a notification of when this happens.

The default timeout is 5 seconds (5000 milliseconds). Changes take immediate effect.

See also

BASS_Encode_CastInit, BASS_Encode_SetNotify

BASS_GetConfig, BASS_SetConfig

BASS_CONFIG_ENCODE_PRIORITY config option

Priority of the encoder DSP.

```
BASS_SetConfig(
    BASS_CONFIG_ENCODE_PRIORITY,
    int priority
);
```

priority The priority.

Remarks

The priority determines where in the DSP chain the encoding is performed; all DSP with a higher priority will be present in the encoding. Changes only affect subsequent encodings, not those that have already been started. The default priority is -1000.

See also

BASS_Encode_Start

BASS_GetConfig, BASS_SetConfig

BASS_CONFIG_ENCODE_QUEUE config option

The maximum queue length.

BASS_SetConfig(
BASS_CONFIG	_ENCODE_QUEUE,
DWORD limit	
`	

);

limit The limit, in milliseconds... 0 = unlimited.

Remarks

When queued encoding is enabled, the queue's buffer will grow as needed to hold the queued data, up to a limit specified by this config option.

The default limit is 10 seconds (10000 milliseconds). Changes only apply to new encoders, not any already existing encoders.

See also

BASS_Encode_GetCount, BASS_Encode_SetNotify, BASS_Encode_Start

BASS_GetConfig, BASS_SetConfig

Sends a RIFF chunk to an encoder.

```
BOOL BASS_Encode_AddChunk(
    DWORD handle,
    char *id,
    void *buffer,
    DWORD length
);
```

handleThe encoderid The 4buffer The bufferlength Thehandle... acharactercontaining thenumber ofHENCODE.chunk id.chunk data.bytes in
the buffer.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

BASS_ERROR_NOTAVAIL No RIFF headers/chunks are being sent to the encoder (due to the BASS_ENCODE_NOHEAD flag being in effect), or sample data encoding has started.

BASS_ERROR_ENDED The encoder has died.

Remarks

BASSenc writes the minimum chunks required of a WAV file: "fmt" and "data", and "ds64" and "fact" when appropriate. This function can be used to add other chunks. For example, a BWF "bext" chunk or "INFO" tags.

Chunks can only be added prior to sample data being sent to the encoder. The BASS_ENCODE_PAUSE flag can be used when starting the encoder to ensure that no sample data is sent before additional chunks have been set.

See also BASS_Encode_Start, BASS_Encode_StartACMFile

Presents the user with a list of available ACM codec output formats to choose from (or suggests one).

```
DWORD BASS_Encode_GetACMFormat(
    DWORD handle,
    void *form,
    DWORD formlen,
    char *title,
    DWORD flags
);
```

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

form Format buffer.

formlen Size of the format buffer. If this is 0, then a suggested format buffer length is returned.

- title Window title for the selector... NULL = "Choose the output format".
- flags A combination of these flags.

BASS_ACM_DEFAULT Use the format buffer (*form*) contents as the default choice in the selector.
BASS_ACM_RATE Only include formats with the same sample rate as the source.
BASS_ACM_CHANS Only include formats with the same number of channels (mono/stereo) as the source.
BASS_ACM_SUGGEST Suggest a format without letting the user choose.
BASS_UNICODE *title* is Unicode (UTF-16).
The HIWORD - use MAKELONG(flags,format) - can be used to restrict the choice to a particular format tag (eg. WAVE FORMAT ADPCM). This is required with

BASS_ACM_SUGGEST, and is optional otherwise.

Return value

If successful, the user-selected (or suggested) format details are put in the *form* buffer and the length of the format details is returned, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code. If *formlen* is 0, then the suggested format buffer size is returned.

Error codes	
BASS_ERROR_HANDLE	<i>handle</i> is not valid.
	There are no codecs available that will accept the channel's format.
BASS_ERROR_ACM_CANCEL	The user pressed the "cancel" button.
BASS_ERROR_UNKNOWN	Some other mystery problem!

Remarks

Unless the BASS_ACM_SUGGEST flag is specified, the user is presented with a list of available ACM codecs to choose from, given the sample format of the channel. The details of the chosen codec's output are returned in the *form* buffer, which can then be used with <u>BASS_Encode_StartACM</u> or <u>BASS_Encode_StartACMFile</u> to begin encoding.

The *form* buffer contents are actually a WAVEFORMATEX structure, and if writing the encoder output to a WAVE file, would be the format chunk ("fmt ") of the file.

Platform-specific

This function is only available on Windows and Windows CE.

Example

Let the user choose a codec, and setup an encoder on a channel using it.

```
DWORD formlen=BASS_Encode_GetACMFormat(0, NULL, 0, NULL, 0); // get
void *form=malloc(formlen); // allocate the format buffer
if (BASS_Encode_GetACMFormat(channel, form, formlen, NULL, 0)) // le
BASS_Encode_StartACMFile(channel, form, 0, "acm.wav"); // begin
free(form); // free the format buffer
```

Without letting the user choose, setup an MP3 encoder on a channel.

```
DWORD formlen=BASS_Encode_GetACMFormat(0,NULL,0,NULL,0); // get sug
void *form=malloc(formlen); // allocate the format buffer
if (BASS_Encode_GetACMFormat(channel,form,formlen,NULL,
MAKELONG(BASS_ACM_SUGGEST|BASS_ACM_RATE|BASS_ACM_CHANS,WAVE_FORM
BASS_Encode_StartACMFile(channel,form,BASS_ENCODE_NOHEAD,"acm.ml
```

free(form); // free the format buffer

See also BASS_Encode_StartACM, BASS_Encode_StartACMFile Retrieves the channel that an encoder is set on.

DWORD BASS_Encode_GetChannel(
 HENCODE handle
);

handle The encoder.

Return value

If successful, the encoder's channel handle is returned, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

See also

BASS_Encode_SetChannel, BASS_Encode_Start

Retrieves the amount of data queued, sent to or received from an encoder, or sent to a cast server.

```
QWORD BASS_Encode_GetCount(
    HENCODE handle,
    DWORD count
);
```

handle The encoder.

The count to retrieve. One of the following. count BASS_ENCODE_COUNT_QUEUE Data currently in the queue, waiting to be sent to the encoder. BASS_ENCODE_COUNT_QUEUE_LIMIT The queue's size limit. BASS_ENCODE_COUNT_QUEUE_FAIL Data not queued due to the queue being full or out of memory. BASS_ENCODE_COUNT_IN Data sent to the encoder. BASS_ENCODE_COUNT_OUT Data received from the encoder. This only applies when the encoder outputs to STDOUT or it is an ACM encoder. BASS_ENCODE_COUNT_CAST Data sent to a cast server.

Return value

If successful, the requested count (in bytes) is returned, else -1 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Remarks

The queue counts are based on the channel's sample format (floating-point if the <u>BASS_CONFIG_FLOATDSP</u> option is enabled), while the BASS_ENCODE_COUNT_IN count is based on the sample format used by the encoder, which could be different if one of the BASS_ENCODE_FP flags is active or the encoder is using an ACM codec (which take 16-bit data).

When the encoder output is being sent to a cast server, the BASS_ENCODE_COUNT_CAST count will match the BASS_ENCODE_COUNT_OUT count, unless there have been problems (eg. network <u>timeout</u>) that have caused data to be dropped.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_NOTAVAILThe encoder does not have a queue.BASS_ERROR_ILLPARAM count is not valid.

See also

BASS_Encode_CastGetStats, BASS_Encode_IsActive

Checks if an encoder is running.

```
DWORD BASS_Encode_IsActive(
        DWORD handle
);
```

Parameters

handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.

Return value

The return value is one of the following. BASS_ACTIVE_STOPPED The encoder isn't running. BASS_ACTIVE_PLAYING The encoder is running. BASS_ACTIVE_PAUSED The encoder is paused.

Remarks

When checking if there's an encoder running on a channel, and there are multiple encoders on the channel, BASS_ACTIVE_PLAYING will be returned if any of them are active.

If an encoder stops running prematurely, <u>BASS_Encode_Stop</u> should still be called to release resources that were allocated for the encoding.

See also

BASS_Encode_GetCount, BASS_Encode_SetNotify, BASS_Encode_SetPaused, BASS_Encode_Stop Moves an encoder (or all encoders on a channel) to another channel.

```
BOOL BASS_Encode_SetChannel(
    DWORD handle,
    DWORD channel
);
```

Parameters

- handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.
- channel The channel to move the encoder(s) to... a HSTREAM, HMUSIC, or HRECORD.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* or *channel* is not valid.

BASS_ERROR_FORMAT The new channel's sample format is not the same as the old channel's.

Remarks

The new channel must have the same sample format (rate, channels, resolution) as the old channel, as that is what the encoder is expecting. A channel's sample format is available via <u>BASS_ChannelGetInfo</u>.

See also

BASS_Encode_GetChannel

Sets a callback function on an encoder (or all encoders on a channel) to receive notifications about its status.

```
BOOL BASS_Encode_SetNotify(
    DWORD handle,
    ENCODENOTIFYPROC *proc,
    void *user
);
```

Parameters

- handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.
- proc Callback function to receive the notifications... NULL = no callback.
- user User instance data to pass to the callback function.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

Remarks

When setting a notification callback on a channel, it only applies to the encoders that are currently set on the channel. Subsequent encoders will not automatically have the notification callback set on them, this function will have to be called again to set them up.

An encoder can only have one notification callback set. Subsequent calls of this function can be used to change the callback function, or disable notifications (proc = NULL).

The status of an encoder and its cast connection (if it has one) is checked when data is sent to the encoder or server, and by <u>BASS_Encode_IsActive</u>. That means an encoder's death will not be detected automatically, and so no notification given, while no data is being encoded.

If the encoder is already dead when setting up a notification callback, the callback will be triggered immediately.

See also

BASS_Encode_Start, ENCODENOTIFYPROC

Pauses or resumes an encoder, or all encoders on a channel.

```
BOOL BASS_Encode_SetPaused(
    DWORD handle,
    BOOL paused
);
```

Parameters

handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.

paused Paused?

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

Remarks

When an encoder is paused, no sample data will be sent to the encoder automatically via the DSP system. Data can still be sent to the encoder manually though, via the <u>BASS_Encode_Write</u> function.

See also

BASS_Encode_IsActive, BASS_Encode_Start, BASS_Encode_Stop

Sets up an encoder on a channel.

```
HENCODE BASS_Encode_Start(
    DWORD handle,
    char *cmdline,
    DWORD flags,
    <u>ENCODEPROC</u> *proc,
    void *user
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

cmdline The encoder command-line, including the executable filename and any options. Or the output filename if the BASS_ENCODE_PCM flag is used.

useu.	
A combination of these flags.	
BASS_ENCODE_PCM	Write plain PCM sample data to a file, without an encoder. The output filename is given in the <i>cmdline</i> parameter.
BASS_ENCODE_NOHEAD	 Don't send a WAVE header to the encoder. If this flag is used then the sample format must be passed to the encoder some other way, eg. via the command-line.
BASS_ENCODE_RF64	Send an RF64 header to the encoder instead of a standard RIFF header, allowing more than 4GB of sample data. This flag is ignored if the BASS_ENCODE_NOHEAD flag is used.
BASS_ENCODE_BIGEND	Send big-endian sample data to the encoder, else little-endian. This flag is ignored unless the BASS_ENCODE_NOHEAD flag is used, as WAV files are little-endian.
BASS_ENCODE_FP_8BIT, BASS_ENCODE_FP_16BIT, BASS_ENCODE_FP_24BIT, BASS_ENCODE_FP_32BIT	Convert floating-point sample data to 8/16/24/32 bit integer. If the encoder does not support 32- bit floating-point sample data, one of these flags can be used to have the sample data converted to integer before it is fed to the encoder. These flags are ignored
	A combination of these flags. BASS_ENCODE_PCM BASS_ENCODE_NOHEAD BASS_ENCODE_RF64 BASS_ENCODE_BIGEND BASS_ENCODE_BIGEND

	if the channel is not floating- point and the <u>BASS_CONFIG_FLOATDSP</u> option is not enabled.
BASS_ENCODE_QUEUE	Queue data to feed the encoder asynchronously. This prevents the data source (DSP system or <u>BASS_Encode_Write</u> call) getting blocked by the encoder, but if data is queud more quickly than the encoder can process it, that could result in lost data.
BASS_ENCODE_LIMIT	Limit the encoding rate to real- time speed, by introducing a delay when the rate is too high. With BASS 2.4.6 or above, this flag is ignored when the encoder is fed in a playback buffer update cycle (including <u>BASS_Update</u> and <u>BASS_ChannelUpdate</u> calls), to avoid possibly causing playback buffer underruns. Except for in those instances, this flag is applied automatically when the encoder is feeding a Shoutcast or Icecast server.
BASS_ENCODE_CAST_NOLIMIT	Don't limit the encoding rate to real-time speed when feeding a Shoutcast or Icecast server. This flag overrides the BASS_ENCODE_LIMIT flag.
BASS_ENCODE_PAUSE	Start the encoder in a paused state.
BASS_ENCODE_AUTOFREE	Automatically free the encoder when the source channel is freed. If queuing is enabled, any

		remaining queued data will be sent to the encoder before it is freed.
	BASS_UNICODE	<i>cmdline</i> is in UTF-16 form. Otherwise it is ANSI on Windows and UTF-8 on OSX.
roc	Optional callback function to receive the encoded data NULL = no	

- proc Optional callback function to receive the encoded data... NULL = no callback. To have the encoded data received by a callback function, the encoder needs to be told to output to STDOUT.
- user User instance data to pass to the callback function.

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE handle is not valid.
BASS_ERROR_FILEOPEN The encoder could not be started. Check that the executable exists.
BASS_ERROR_CREATE The PCM file could not be created.
BASS_ERROR_NOTAVAIL External encoders are not supported.
BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

The encoder must be told (via the command-line) to expect input from STDIN, rather than a file. The command-line should also tell the encoder what filename to write its output to, unless you are using a callback function, in which case it should be told to write its output to STDOUT.

No user interaction with the encoder is possible, so anything that would cause the encoder to require the user to press any keys should be avoided. For example, if the encoder asks whether to overwrite files, the encoder should be instructed to always overwrite (via the command-line), or the existing file should be deleted before starting the encoder.

Standard RIFF files are limited to a little over 4GB in size. When writing a WAV file, BASSenc will automatically stop at that point, so that the file is valid. That does not apply when sending data to an encoder though, as the encoder may (possibly via a command-line option) ignore the size restriction, but if it does not, it could mean that the encoder stops after a few hours (depending on the sample format). If longer encodings are needed, the BASS_ENCODE_NOHEAD flag can be used to omit the WAVE header, and the encoder informed of the sample format via the command-line instead. The 4GB size limit can also be overcome with the BASS_ENCODE_RF64 flag, but most

encoders are unlikely to support RF64.

When writing an RF64 WAV file, a standard RIFF header will still be written initially, which will only be replaced by an RF64 header at the end if the file size has exceeded the standard limit. When an encoder is used, it is not possible to go back and change the header at the end, so the RF64 header is sent at the beginning in that case.

Internally, the sending of sample data to the encoder is implemented via a DSP callback on the channel. That means when the channel is played (or <u>BASS_ChannelGetData</u> is called if it is a decoding channel), the sample data will be sent to the encoder at the same time. It also means that if the <u>BASS_CONFIG_FLOATDSP</u> option is enabled, the sample data will be 32-bit floating-point, and one of the BASS_ENCODE_FP flags will be required if the encoder does not support floating-point sample data. The <u>BASS_CONFIG_FLOATDSP</u> setting should not be changed while encoding is in progress.

By default, the encoder DSP has a priority setting of -1000, which determines where in the DSP chain the encoding is performed. That can be changed via the <u>BASS_CONFIG_ENCODE_PRIORITY</u> config option.

Besides the automatic DSP system, data can also be manually fed to the encoder via the <u>BASS_Encode_Write</u> function. Both methods can be used together, but in general, the "automatic" system ought to be paused when using the "manual" system, via the BASS_ENCODE_PAUSE flag or the <u>BASS_Encode_SetPaused</u> function. Data fed to the encoder manually does not go through the source channel's DSP chain, so any DSP/FX set on the channel will not be applied to the data.

When queued encoding is enabled via the BASS_ENCODE_QUEUE flag, the DSP system or <u>BASS_Encode_Write</u> call will just buffer the data, and the data will then be fed to the encoder by another thread. The buffer will grow as needed to hold the queued data, up to a limit specified by the <u>BASS_CONFIG_ENCODE_QUEUE</u> config option. If the limit is exceeded (or there is no free memory), data will be lost; <u>BASS_Encode_SetNotify</u> can be used to be notified of that occurrence. The amount of data that is currently queued, as well as the queue limit and how much data has been lost, is available from <u>BASS_Encode_GetCount</u>.

<u>BASS_Encode_IsActive</u> can be used to check that the encoder is still running. When done encoding, use <u>BASS_Encode_Stop</u> or <u>BASS_Encode_StopEx</u> to close the encoder.

The returned handle is the encoder's process handle, which can be used to do things like change the encoder's priority (SetPriorityClass) and get its exit code (GetExitCodeProcess).

Multiple encoders can be set on a channel. For convenience, most of the encoder functions will accept either an encoder handle or a channel handle. When a channel handle is used, the function is applied to all encoders that are set on that channel.

Platform-specific

External encoders are not supported on iOS or Windows CE, so only plain PCM file writing with the BASS_ENCODE_PCM flag is possible on those platforms.

Example

Start encoding a channel to an MP3 file (output.mp3) using LAME with the standard preset settings.

BASS_Encode_Start(channel, "lame --alt-preset standard - output.mp3 BASS_ChannelPlay(channel, 0); // start the channel playing & encodi

Start writing a channel to a WAV file (output.wav).

BASS_Encode_Start(channel, "output.wav", BASS_ENCODE_PCM, NULL, 0); BASS_ChannelPlay(channel, 0); // start the channel playing & encodi

See also

BASS_Encode_AddChunk, BASS_Encode_CastInit, BASS_Encode_IsActive, BASS_Encode_ServerInit, BASS_Encode_SetNotify, BASS_Encode_SetPaused, BASS_Encode_StartACM, BASS_Encode_StartCA,

BASS_Encode_StartLimit, BASS_Encode_Stop, BASS_Encode_Write,

ENCODEPROC callback, BASS_CONFIG_ENCODE_PRIORITY

Sets up an encoder on a channel, using an ACM codec and sending the output to a user defined function.

```
HENCODE BASS_Encode_StartACM(
    DWORD handle,
    void *form,
    DWORD flags,
    <u>ENCODEPROC</u> *proc,
    void *user
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

- ACM codec output format. form
- A combination of these flags. flags BASS_ENCODE_QUEUE

E

BASS	ENCODE	LIMIT

	data source (DSP system or
	BASS_Encode_Write call)
	getting blocked by the encoder,
	but if data is queud more quickly
	than the encoder can process it,
	that could result in lost data.
BASS_ENCODE_LIMIT	Limit the encoding rate to real-
	time speed, by introducing a
	delay when the rate is too high.
	With BASS 2.4.6 or above, this
	flag is ignored when the encoder
	is fed in a playback buffer update
	cycle (including <u>BASS_Update</u>
	and BASS_ChannelUpdate calls),
	to avoid possibly causing
	playback buffer underruns.
	Except for in those instances, this
	flag is applied automatically
	when the encoder is feeding a
	Shoutcast or Icecast server.
BASS_ENCODE_CAST_NOLIMIT	Don't limit the encoding rate to
	real-time speed when feeding a
	Shoutcast or Icecast server. This
	flag overrides the
	BASS_ENCODE_LIMIT flag.
BASS_ENCODE_PAUSE	Start the encoder paused.
BASS_ENCODE_AUTOFREE	Automatically free the encoder
	when the source channel is freed.
	If queuing is enabled, any

Queue data to feed the encoder asynchronously. This prevents the

remaining queued data will be sent to the encoder before it is freed.

- proc Callback function to receive the encoded data.
- user User instance data to pass to the callback function.

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_NOTAVAILThe codec specified in form couldn't be initialized.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function allows installed ACM (Audio Compression Manager) codecs to be used for encoding. The codec used is determined by the contents of the *form* parameter. The <u>BASS_Encode_GetACMFormat</u> function can be used to initialize that. ACM does not support floating-point data, so floating-point data will be converted to 16-bit before it is fed to the codec.

Internally, the sending of sample data to the encoder is implemented via a DSP callback on the channel. That means when you play the channel (or call <u>BASS_ChannelGetData</u> if it's a decoding channel), the sample data will be sent to the encoder at the same time. The encoding is performed in the DSP callback; there isn't a separate process doing the encoding, as when using an external encoder via <u>BASS_Encode_Start</u>.

By default, the encoder DSP has a priority setting of -1000, which determines where in the DSP chain the encoding is performed. That can be changed via the <u>BASS_CONFIG_ENCODE_PRIORITY</u> config option.

Besides the automatic DSP system, data can also be manually fed to the encoder via the <u>BASS_Encode_Write</u> function. Both methods can be used together, but in general, the "automatic" system ought to be paused when using the "manual" system, via the BASS_ENCODE_PAUSE flag or the <u>BASS_Encode_SetPaused</u> function. Data fed to the encoder manually does not go through the source channel's DSP chain, so any DSP/FX set on the channel will not be applied to the data.

When queued encoding is enabled via the BASS_ENCODE_QUEUE flag, the DSP system or <u>BASS_Encode_Write</u> call will just buffer the data, and the data will then be fed to the encoder by another thread. The buffer will grow as needed to hold the queued data, up to a limit specified by the <u>BASS_CONFIG_ENCODE_QUEUE</u> config option. If the limit is exceeded (or there is no free memory), data will be lost; <u>BASS_Encode_SetNotify</u> can be used to be notified of that occurrence. The amount of data that is currently queued, as well as the queue limit and how much data has been lost, is available from <u>BASS_Encode_GetCount</u>.

When done encoding, use <u>BASS_Encode_Stop</u> to close the encoder.

Multiple encoders can be set on a channel. For convenience, most of the encoder functions will accept either an encoder handle or a channel handle. When a channel handle is used, the function is applied to all encoders that are set on that channel.

<u>BASS_Encode_StartACMFile</u> can be used to have the encoder output sent to a file instead of a callback function.

Platform-specific

This function is only available on Windows and Windows CE.

See also

BASS_Encode_CastInit, BASS_Encode_GetACMFormat,

BASS_Encode_IsActive, BASS_Encode_ServerInit, BASS_Encode_SetPaused,

BASS_Encode_Start, BASS_Encode_StartACMFile, BASS_Encode_Stop,

BASS_Encode_Write, ENCODEPROC callback,

BASS_CONFIG_ENCODE_PRIORITY

Sets up an encoder on a channel, using an ACM codec and writing the output to a file.

```
HENCODE BASS_Encode_StartACMFile(
    DWORD handle,
    void *form,
    DWORD flags,
    char *file
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

form ACM codec output format.

flags	A combination of these flags.	
	BASS_ENCODE_NOHEAD	Don't write a WAVE header to the file.
	BASS_ENCODE_RF64	Write an RF64 header instead of a standard RIFF header, allowing the file to go beyond 4GB in size. This flag is ignored if the BASS_ENCODE_NOHEAD flag is used.
	BASS_ENCODE_QUEUE	Queue data to feed the encoder asynchronously. This prevents the data source (DSP system or <u>BASS Encode Write</u> call) getting blocked by the encoder, but if data is queud more quickly than the encoder can process it, that could result in lost data.
	BASS_ENCODE_PAUSE	Start the encoder paused.
	BASS_ENCODE_AUTOFREE	Automatically free the encoder when the source channel is freed. If queuing is enabled, any remaining queued data will be sent to the encoder before it is freed.
	BASS_UNICODE	<i>file</i> is a Unicode (UTF-16) filename.
file	The filename to write.	

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

BASS_ERROR_NOTAVAIL The codec specified in *form* couldn't be initialized.

BASS_ERROR_CREATE The file could not be created.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function is identical to <u>BASS_Encode_StartACM</u>, except that it writes the encoded data to a file instead of a callback function.

Unless the BASS_ENCODE_NOHEAD flag is specified, a WAVE header and the *form* contents will be written to the file. This is generally required for the file to be playable, but in some cases (eg. MP3) it's not. Standard RIFF WAV files are limited to a little over 4GB in size, so BASSenc will automatically stop encoding at that point. That size limit can be overcome with an RF64 file. When writing an RF64 WAV file, a standard RIFF header will still be written initially, which will only be replaced by an RF64 header at the end if the file size has exceeded the standard limit.

Platform-specific

This function is only available on Windows and Windows CE.

See also

BASS_Encode_AddChunk, BASS_Encode_GetACMFormat, BASS_Encode_IsActive, BASS_Encode_SetPaused, BASS_Encode_Start, BASS_Encode_StartACM, BASS_Encode_Stop, BASS_Encode_Write, ENCODEPROC callback, BASS_CONFIG_ENCODE_PRIORITY Sets up an encoder on a channel, using a CoreAudio codec and sending the output to a user defined function.

```
HENCODE BASS_Encode_StartCA(
    DWORD handle,
    DWORD ftype,
    DWORD atype,
    DWORD flags,
    DWORD bitrate,
    <u>ENCODEPROCEX</u> *proc,
    void *user
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

ftype File format identifier.

- atype Audio data format identifier.
- flags A combination of these flags. BASS_ENCODE_FP_8BIT, BASS_ENCODE_FP_16BIT, BASS_ENCODE_FP_24BIT, BASS_ENCODE_FP_32BIT

BASS_ENCODE_MONO

BASS_ENCODE_QUEUE

BASS_ENCODE_LIMIT

Convert floating-point sample data to 8/16/24/32 bit integer before encoding. These flags are ignored if the channel is not floating-point and the BASS CONFIG FLOATDSP option is not enabled. Convert to mono before encoding. Queue data to feed the encoder asynchronously. This prevents the data source (DSP system or BASS Encode Write call) getting blocked by the encoder, but if data is queud more quickly than the encoder can process it, that could result in lost data. Limit the encoding rate to realtime speed, by introducing a delay when the rate is too high. With BASS 2.4.6 or above, this flag is ignored when the encoder is fed in a playback buffer update cycle (including **BASS_Update** and **BASS** ChannelUpdate calls), to avoid possibly causing playback buffer underruns. Except for in those instances, this flag is applied automatically when the encoder is feeding a

	Shoutcast or Icecast server.
BASS_ENCODE_CAST_NOLIMIT	Don't limit the encoding rate to
	real-time speed when feeding a
	Shoutcast or Icecast server. This
	flag overrides the
	BASS_ENCODE_LIMIT flag.
BASS_ENCODE_PAUSE	Start the encoder in a paused
	state.
BASS_ENCODE_AUTOFREE	Automatically free the encoder when the source channel is freed.
	If queuing is enabled, any
	remaining queued data will be
	sent to the encoder before it is
	freed.
	

- bitrate The bitrate in bits per second... 0 = the codec's default bitrate.
- proc Callback function to receive the encoded data.
- user User instance data to pass to the callback function.

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_FILEFORMftype is not valid.BASS_ERROR_CODECatype is not valid or supported with ftype.BASS_ERROR_NOTAVAILbitrate is not supported by the codec.BASS_ERROR_FORMATThe channel's sample format is not supported by the codec.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function allows CoreAudio codecs to be used for encoding. A list of standard file and audio data format identifiers is available from Apple, <u>here</u>. The available file and audio data identifiers, as well as other information, can be retreived via the Audio File Services and Audio Format Services APIs, eg. the <u>kAudioFileGlobalInfo_WritableTypes</u> and <u>kAudioFormatProperty_EncodeFormatIDs</u> properties.

Internally, the sending of sample data to the encoder is implemented via a DSP callback on the channel. That means when you play the channel (or call <u>BASS_ChannelGetData</u> if it's a decoding channel), the sample data will be sent to the encoder at the same time. The encoding is performed in the DSP callback; there isn't a separate process doing the encoding, as when using an external encoder via <u>BASS_Encode_Start</u>.

By default, the encoder DSP has a priority setting of -1000, which determines where in the DSP chain the encoding is performed. That can be changed via the <u>BASS_CONFIG_ENCODE_PRIORITY</u> config option.

Besides the automatic DSP system, data can also be manually fed to the encoder via the <u>BASS_Encode_Write</u> function. Both methods can be used together, but in general, the "automatic" system ought to be paused when using the "manual" system, via the BASS_ENCODE_PAUSE flag or the <u>BASS_Encode_SetPaused</u> function. Data fed to the encoder manually does not go through the source channel's DSP chain, so any DSP/FX set on the channel will not be applied to the data.

When queued encoding is enabled via the BASS_ENCODE_QUEUE flag, the DSP system or <u>BASS_Encode_Write</u> call will just buffer the data, and the data will then be fed to the encoder by another thread. The buffer will grow as needed to hold the queued data, up to a limit specified by the <u>BASS_CONFIG_ENCODE_QUEUE</u> config option. If the limit is exceeded (or there is no free memory), data will be lost; <u>BASS_Encode_SetNotify</u> can be used to be notified of that occurrence. The amount of data that is currently queued, as well as the queue limit and how much data has been lost, is available from <u>BASS_Encode_GetCount</u>.

When done encoding, use <u>BASS_Encode_Stop</u> to close the encoder.

Multiple encoders can be set on a channel. For convenience, most of the encoder functions will accept either an encoder handle or a channel handle. When a channel handle is used, the function is applied to all encoders that are set on that channel.

<u>BASS_Encode_StartCAFile</u> can be used to have the encoder output sent to a file instead of a callback function.

Platform-specific

This function is only available on OSX and iOS.

See also

BASS_Encode_CastInit, BASS_Encode_GetACMFormat,

BASS_Encode_IsActive, BASS_Encode_ServerInit, BASS_Encode_SetPaused,

BASS_Encode_Start, BASS_Encode_StartCAFile, BASS_Encode_Stop,

BASS_Encode_Write, ENCODEPROCEX callback,

BASS_CONFIG_ENCODE_PRIORITY

Sets up an encoder on a channel, using a CoreAudio codec and sending the output to a file.

```
HENCODE BASS_Encode_StartCAFile(
    DWORD handle,
    DWORD ftype,
    DWORD atype,
    DWORD flags,
    DWORD bitrate,
    char *file
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

- ftype File format identifier.
- atype Audio data format identifier.
- flags A combination of these flags. BASS_ENCODE_FP_8BIT, Convert floating-point sample data to BASS_ENCODE_FP_16BIT, 8/16/24/32 bit integer before encoding. BASS ENCODE FP 24BIT, These flags are ignored if the channel BASS ENCODE FP 32BIT is not floating-point and the **BASS CONFIG FLOATDSP option** is not enabled. BASS_ENCODE_MONO Convert to mono before encoding. BASS ENCODE QUEUE Queue data to feed the encoder asynchronously. This prevents the data source (DSP system or **BASS** Encode Write call) getting blocked by the encoder, but if data is queud more quickly than the encoder can process it, that could result in lost data. BASS ENCODE PAUSE Start the encoder in a paused state. BASS_ENCODE_AUTOFREE Automatically free the encoder when the source channel is freed. If queuing is enabled, any remaining queued data will be sent to the encoder before it is freed. BASS_UNICODE *file* is a Unicode (UTF-16) filename. The bitrate in bits per second... 0 = the codec's default bitrate. bitrate file The filename to write.

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_FILEFORMftype is not valid.BASS_ERROR_CODECatype is not valid or supported with ftype.BASS_ERROR_NOTAVAILbitrate is not supported by the codec.BASS_ERROR_FORMATThe channel's sample format is not supported by the codec.

BASS_ERROR_CREATE The file could not be created. BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function is identical to <u>BASS_Encode_StartCA</u>, except that it writes the encoded data to a file instead of a callback function.

Platform-specific This function is only available on OSX and iOS.

Example

Start encoding a channel to an ALAC file (output.m4a).

BASS_Encode_StartCAFile(channel, 'm4af', 'alac', 0, 0, "output.m4a" BASS_ChannelPlay(channel, 0); // start the channel playing & encodi

Start encoding a channel to a 128 kb/s AAC MP4 file (output.mp4).

BASS_Encode_StartCAFile(channel, 'mp4f', 'aac ', 0, 128000, "output BASS_ChannelPlay(channel, 0); // start the channel playing & encodi See also

BASS_Encode_GetACMFormat, BASS_Encode_IsActive, BASS_Encode_SetPaused, BASS_Encode_Start, BASS_Encode_StartCA, BASS_Encode_Stop, BASS_Encode_Write, BASS_CONFIG_ENCODE_PRIORITY Sets up an encoder on a channel, and limits the amount of sample data that is fed to it.

```
HENCODE BASS_Encode_StartLimit(
    DWORD handle,
    char *cmdline,
    DWORD flags,
    <u>ENCODEPROC</u> *proc,
    void *user,
    DWORD limit
);
```

Parameters

handle The channel handle... a HSTREAM, HMUSIC, or HRECORD.

cmdline The encoder command-line, including the executable filename and any options. Or the output filename if the BASS_ENCODE_PCM flag is used.

useu.	
A combination of these flags.	
BASS_ENCODE_PCM	Write plain PCM sample data to a file, without an encoder. The output filename is given in the <i>cmdline</i> parameter.
BASS_ENCODE_NOHEAD	 Don't send a WAVE header to the encoder. If this flag is used then the sample format must be passed to the encoder some other way, eg. via the command-line.
BASS_ENCODE_RF64	Send an RF64 header to the encoder instead of a standard RIFF header, allowing more than 4GB of sample data. This flag is ignored if the BASS_ENCODE_NOHEAD flag is used.
BASS_ENCODE_BIGEND	Send big-endian sample data to the encoder, else little-endian. This flag is ignored unless the BASS_ENCODE_NOHEAD flag is used, as WAV files are little-endian.
BASS_ENCODE_FP_8BIT, BASS_ENCODE_FP_16BIT, BASS_ENCODE_FP_24BIT, BASS_ENCODE_FP_32BIT	Convert floating-point sample data to 8/16/24/32 bit integer. If the encoder does not support 32- bit floating-point sample data, one of these flags can be used to have the sample data converted to integer before it is fed to the encoder. These flags are ignored
	A combination of these flags. BASS_ENCODE_PCM BASS_ENCODE_NOHEAD BASS_ENCODE_RF64 BASS_ENCODE_BIGEND BASS_ENCODE_BIGEND

	if the channel is not floating- point and the <u>BASS_CONFIG_FLOATDSP</u> option is not enabled.
BASS_ENCODE_QUEUE	Queue data to feed the encoder asynchronously. This prevents the data source (DSP system or <u>BASS_Encode_Write</u> call) getting blocked by the encoder, but if data is queud more quickly than the encoder can process it, that could result in lost data.
BASS_ENCODE_LIMIT	Limit the data rate to real-time speed, by introducing a delay when the rate is too high. With BASS 2.4.6 or above, this flag is ignored when the encoder is fed in a playback buffer update cycle (including <u>BASS_Update</u> and <u>BASS_ChannelUpdate</u> calls), to avoid possibly causing playback buffer underruns. Except for in those instances, this flag is applied automatically when the encoder is feeding a Shoutcast or Icecast server.
BASS_ENCODE_CAST_NOLIMI	Γ Don't limit the data rate to real- time speed when feeding a Shoutcast or Icecast server. This flag overrides the BASS_ENCODE_LIMIT flag.
BASS_ENCODE_PAUSE	Start the encoder paused.
BASS_ENCODE_AUTOFREE	Automatically free the encoder when the source channel is freed. If queuing is enabled, any remaining queued data will be

sent to the encoder before it is freed.

BASS_UNICODE

cmdline is in UTF-16 form. Otherwise it is ANSI on Windows and UTF-8 on OSX.

- proc Optional callback function to receive the encoded data... NULL = no callback. To have the encoded data received by a callback function, the encoder needs to be told to output to STDOUT.
- user User instance data to pass to the callback function.
- limit The number of bytes of sample data to encode... 0 = unlimited. If one of the BASS_ENCODE_FP flags is used, the limit is applied after the effect of that.

Return value

The encoder handle is returned if the encoder is successfully started, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_FILEOPENThe encoder could not be started. Check that the
executable exists.BASS_ERROR_CREATEThe DCM (ill and black black

BASS_ERROR_CREATE The PCM file could not be created.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function is identical to <u>BASS_Encode_Start</u>, with the additional ability to limit the amount of sample data that is fed to the encoder. This can be useful in situations where the encoder needs to know in advance how much data it will be receiving. For example, when using a callback function with a file format that stores the length in the header, as the header cannot then be updated at the end of encoding. The length is communicated to the encoder via the WAVE header, so it requires that the BASS_ENCODE_NOHEAD flag is not used.

Once the limit is hit, the encoder will "die". <u>BASS_Encode_SetNotify</u> can be used to be notified of that occurrence.

Example

Start encoding a channel to an MP3 file (output.mp3) using LAME with the standard preset settings, limiting it to 1000000 bytes of sample data.

BASS_Encode_StartLimit(channel, "lame --alt-preset standard - output BASS_ChannelPlay(channel, 0); // start the channel playing & encodi See also

BASS_Encode_AddChunk, BASS_Encode_CastInit, BASS_Encode_IsActive, BASS_Encode_SetNotify, BASS_Encode_SetPaused, BASS_Encode_Start, BASS_Encode_Stop, BASS_Encode_Write, ENCODEPROC callback, BASS_CONFIG_ENCODE_PRIORITY Frees an encoder or all encoders on a channel.

```
BOOL BASS_Encode_Stop(
    DWORD handle
);
```

Parameters

handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Remarks

This function will free an encoder immediately, without waiting for any data that may be remaining in the queue. <u>BASS_Encode_StopEx</u> can be used to have an encoder process the queue before it is freed.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

See also

BASS_Encode_Start, BASS_Encode_StopEx

Sends sample data to an encoder or all encoders on a channel.

```
BOOL BASS_Encode_Write(
    DWORD handle,
    void *buffer,
    DWORD length
);
```

Parameters

handle The encoder or channel handle a	buffer The buffer	length The
HENCODE, HSTREAM,	containing	number
HMUSIC, or HRECORD.	the sample	of bytes
	data.	in the
		buffer.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

BASS_ERROR_ENDED No more data may be fed to the encoder. The encoder has died, or the encoder's queue is being processed before it's closed.

Remarks

There is usually no need to use this function, as the channel's sample data will automatically be fed to the encoder as it is decoded/played. But in some situations, it could be useful to be able to manually feed the encoder instead. The sample data is expected to be the same format as the channel's, or floating-point if the <u>BASS_CONFIG_FLOATDSP</u> option is enabled.

When queued encoding is enabled, this function will return successfully even if the queue did not have space for all of the provided data.

<u>BASS_Encode_GetCount</u> can be used to check that there is sufficient space prior to calling this function.

See also

BASS_Encode_SetPaused, BASS_Encode_Start

ENCODENOTIFYPROC callback

User defined callback function to receive notifications on an encoder's status.

```
void CALLBACK EncodeNotifyProc(
    HENCODE handle,
    DWORD status,
    void *user
);
```

Parameters

handle	The encoder that the notification is from.			
status	The encoder's status, one of the following.			
	BASS_ENCODE_NOTIFY_ENCODER	The encoder died.		
	BASS_ENCODE_NOTIFY_CAST	Cast server connection died.		
	BASS_ENCODE_NOTIFY_CAST_TIMEOUT	Cast data sending timeou The connection is not dead at this point; it may just be a temporary problem.		
	BASS_ENCODE_NOTIFY_QUEUE_FULL	The queue length has reached its limit (or out (memory) and data has been dropped. The total amount of dropped data available from BASS_Encode_GetCour		
licor	The user instance data given when BASS Encode SetNotify was called			

user The user instance data given when <u>BASS_Encode_SetNotify</u> was called.

Remarks

It is safe to call <u>BASS_Encode_Stop</u> to free an encoder from within a notification callback.

See also

BASS_Encode_SetNotify

ENCODEPROC callback

User defined callback function to process encoded sample data.

```
void CALLBACK EncodeProc(
    HENCODE handle,
    DWORD channel,
    void *buffer,
    DWORD length,
    void *user
);
```

Parameters

handle The encoder that the data is from.

channel The channel that the encoder is set on.

buffer Buffer containing the encoded data.

length The number of bytes in the buffer.

user The user instance data given when <u>BASS_Encode_Start</u> was called.

See also BASS_Encode_Start

ENCODEPROCEX callback

User defined callback function to process encoded sample data.

```
void CALLBACK EncodeProcEx(
    HENCODE handle,
    DWORD channel,
    void *buffer,
    DWORD length,
    DWORD offset,
    void *user
);
```

Parameters

handle The encoder that the data is from.

channel The channel that the encoder is set on.

buffer Buffer containing the encoded data.

length The number of bytes in the buffer.

offset File offset of the data.

user The user instance data given when <u>BASS_Encode_StartCA</u> was called.

Example

A callback function to write the encoded data to to a file.

```
void CALLBACK MyFileWriter(HENCODE handle, DWORD channel, void *buf
{
    FILE *file=(FILE*)user;
    fseek(file, offset, SEEK_SET); // seek to file offset
    fwrite(buffer, 1, length, file); // write the data
}
```

NOTE: This is just an example. It is simpler to use <u>BASS_Encode_StartCAFile</u> to encode to a file.

See also BASS_Encode_StartCA

Retrieves stats from the Shoutcast or Icecast server.

```
char *BASS_Encode_CastGetStats(
    HENCODE handle
    DWORD type,
    char *pass
);
```

Parameters

handle The encoder handle.

type	The type of stats to retrieve. One of the following.		
	BASS_ENCODE_STATS_SHOUT	Shoutcast stats, including listener information and additional server information.	
	BASS_ENCODE_STATS_ICE	Icecast mount-point listener information.	
	BASS_ENCODE_STATS_ICESERV	V Icecast server stats, including information on all mount points on the server.	
nass	Password when retrieving Icecast server stats $NULL = use the$		

pass Password when retrieving Icecast server stats... NULL = use the password provided in the <u>BASS_Encode_CastInit</u> call.

Return value

If successful, the stats are returned, else NULL is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

BASS_ERROR_ILLTYPE *type* is invalid.

BASS_ERROR_NOTAVAIL There isn't a cast of the requested type set on the encoder.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

The stats are returned in XML format.

Each encoder has a single stats buffer, which is reused by each call of this function for the encoder. So if the data needs to be retained across multiple calls, it should be copied to another buffer.

Example

Display the number of listeners.

See also BASS_Encode_CastInit, BASS_Encode_GetCount

Initializes sending an encoder's output to a Shoutcast or Icecast server.

```
BOOL BASS_Encode_CastInit(
    HENCODE handle,
    char *server,
    char *pass,
    char *content,
    char *content,
    char *name,
    char *url,
    char *genre,
    char *desc,
    char *headers,
    DWORD bitrate,
    BOOL pub
);
```

Parameters

handle The encoder handle.

- server The server to send to, in the form of "address:port" (Shoutcast) or "address:port/mount" (Icecast).
- pass The server password.
- content The MIME type of the encoder output. This can be one of the following.

BASS_ENCODE_TYPE_MP3 MP3.

BASS_ENCODE_TYPE_OGG OGG.

BASS_ENCODE_TYPE_AAC AAC.

- name The stream name... NULL = no name.
- url The URL, for example, of the radio station's webpage... NULL = no URL.
- genre The genre... NULL = no genre.
- desc Description... NULL = no description. This applies to Icecast only.
- headers Other headers to send to the server... NULL = none. Each header should end with a carriage return and line feed ("\r\n").
- bitrate The bitrate (in kbps) of the encoder output... 0 = undefined bitrate. In cases where the bitrate is a "quality" (rather than CBR) setting, the *headers* parameter can be used to communicate that instead, eg. "ice-bitrate: Quality 0\r\n".
- pub Public? If TRUE, the stream is added to the public directory of streams, at <u>shoutcast.com</u> or <u>dir.xiph.org</u> (or as defined in the server config).

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_ALREADYThere is already a cast set on the encoder.BASS_ERROR_ILLPARAMserver doesn't include a port number.BASS_ERROR_FILEOPENCouldn't connect to the server.BASS_ERROR_CAST_DENIEDpass is not valid.BASS_ERROR_UNKNOWNSome other mystery problem!

Remarks

This function sets up a Shoutcast/Icecast source client, sending the encoder's output to a server, which listeners can then connect to and receive the data from. The Shoutcast and Icecast server software is available from www.shoutcast.com/download/serve.phtml and <a href="https://www.shoutcast.com/download/serve.phtml"

An encoder needs to be started, but with no data yet sent to it, before using this function to setup the sending of the encoder's output to a Shoutcast or Icecast server. If <u>BASS_Encode_Start</u> is used, the encoder should be setup to write its output to STDOUT. Due to the length restrictions of WAVE headers/files, the encoder should also be started with the BASS_ENCODE_NOHEAD flag, and the sample format details sent via the command-line.

Unless the BASS_ENCODE_CAST_NOLIMIT flag is set on the encoder, BASSenc automatically limits the rate that data is processed to real-time speed to avoid overflowing the server's buffer, which means that it is safe to simply try to process data as quickly as possible, eg. when the source is a decoding channel. Encoders set on recording channels are automatically exempt from the rate limiting, as they are inherently real-time. With BASS 2.4.6 or above, also exempt are encoders that are fed in a playback buffer update cycle (including <u>BASS_Update</u> and <u>BASS_ChannelUpdate</u> calls), eg. when the source is a playing channel; that is to avoid delaying the update thread, which could result in playback buffer underruns.

<u>BASS_Encode_ServerInit</u> can be used to setup a server that listeners can connect to directly, without a Shoutcast/Icecast server intermediary.

Platform-specific

This function is not available on Windows CE.

Example

Start encoding a stereo 44100hz channel to 128kb/s MP3, and send the output to a Shoutcast server.

```
HENCODE encoder=BASS_Encode_Start(channel, "lame -r -s 44100 -b 128
BASS_Encode_CastInit(encoder, "server.com:8000", "password", BASS_EI
    "genre", NULL, NULL, 128, TRUE); // start the cast
```

See also

BASS_Encode_CastGetStats, BASS_Encode_CastSetTitle,

BASS_Encode_ServerInit, BASS_Encode_SetNotify, BASS_Encode_Start,

BASS_Encode_StartACM, BASS_Encode_StartCA,

BASS_CONFIG_ENCODE_CAST_TIMEOUT

Sets the title of a cast stream.

```
BOOL BASS_Encode_CastSetTitle(
    HENCODE handle,
    char *title,
    char *url
);
```

Parameters

handle The encoder handle.

- title The title... NULL = no title.
- url URL to go with the title... NULL = no URL. This applies to Shoutcast only.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLEhandle is not valid.BASS_ERROR_NOTAVAILThere isn't a cast set on the encoder.BASS_ERROR_UNKNOWNSome other mystery problem!

See also BASS_Encode_CastInit

Initializes a server to send an encoder's output to connecting clients.

```
DWORD BASS_Encode_ServerInit(
    HENCODE handle,
    char *port,
    DWORD buffer,
    DWORD burst,
    DWORD flags,
    ENCODECLIENTPROC *proc,
    void user
);
```

Parameters

handle The encoder handle.

- port The IP address and port number to accept client connections on... "xxx.xxx.xxx:port", NULL = an available port on all local addresses. The IP address should be local and the port number should be lower than 65536. If the address is "0.0.0.0" or omitted, then the server will accept connections on all local addresses. If the port is "0" or omitted, then an available port will be assigned.
- buffer The server's buffer length in bytes.
- burst The amount of buffered data to send to new clients. This will be capped at the size of the buffer.
- flags A combination of these flags. BASS_ENCODE_SERVER_NOHTTP Do not read or send HTTP headers.
- proc Callback function to receive notification of clients connecting and disconnecting... NULL = no callback.
- user User instance data to pass to the callback function.

Return value

If successful, the new server's port number is returned, else 0 is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

BASS_ERROR_ALREADY There is already a server set on the encoder.

BASS_ERROR_ILLPARAM *port* is not valid.

BASS_ERROR_BUSY The port is in use.

BASS_ERROR_MEM There is insufficient memory.

BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function allows remote (or local) clients to receive the encoder's output by setting up a TCP server for them to connect to, using <u>BASS_StreamCreateURL</u> for example. Connections can be refused by the <u>ENCODECLIENTPROC</u> callback function, and already connected clients can be kicked with the <u>BASS_Encode_ServerKick</u> function.

The server buffers the data that it receives from the encoder, and the data is then sent from the buffer to the connected clients. The buffer should be at least big enough to account for the time that it takes for the clients to receive the data. If a client falls too far behind (beyond the buffer length), it will miss some data. When a client connects, buffered data can be "burst" to the client, allowing it to prebuffer and begin playback more quickly.

An encoder needs to be started, but with no data yet sent to it, before using this function to setup the server. If <u>BASS_Encode_Start</u> is used, the encoder should be setup to write its output to STDOUT. Due to the length restrictions of WAVE headers/files, the encoder should also be started with the BASS_ENCODE_NOHEAD flag, and the sample format details sent via the command-line.

Platform-specific

This function is not available on Windows CE.

Example

Start encoding a stereo 44100hz channel to 128kb/s MP3, and start a server on port 8000 with a fully burstable 4 second (64KB) buffer.

HENCODE encoder=BASS_Encode_Start(channel, "lame -r -s 44100 -b 128 BASS_Encode_ServerInit(encoder, "8000", 64000, 64000, 0, NULL, NULL

Start encoding a stereo 44100hz channel to 160kb/s OGG, and start a server on any available port on the loopback address (127.0.0.1) with a fully burstable 2 second (40KB) buffer.

HENCODE encoder=BASS_Encode_Start(channel, "oggenc -r -R 44100 -M 1 DWORD port=BASS_Encode_ServerInit(encoder, "127.0.0.1", 40000, 4000

See also

BASS_Encode_CastInit, BASS_Encode_ServerKick, BASS_Encode_SetNotify, BASS_Encode_Start, BASS_Encode_StartACM, BASS_Encode_StartCA, BASS_CONFIG_ENCODE_CAST_TIMEOUT Kicks clients from a server.

```
DWORD BASS_Encode_ServerInit(
    HENCODE handle,
    char *client
);
```

Parameters

handle The encoder handle.

client The client(s) to kick... "" (empty string) = all clients. Unless a port number is included, this string is compared with the start of the connected clients' IP address.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not valid. BASS_ERROR_NOTAVAIL No matching clients were found.

Remarks

The clients may not be kicked immediately, but shortly after the call. If the server has been setup with an <u>ENCODECLIENTPROC</u> callback function, that will receive notification of the disconnections.

Example

Kick a client connected from port 1234 at 1.2.3.4.

BASS_Encode_ServerKick(encoder, "1.2.3.4:1234");

Kick all clients connected from 1.2.3.4.

BASS_Encode_ServerKick(encoder, "1.2.3.4:");

See also

BASS_Encode_ServerInit, ENCODECLIENTPROC

ENCODECLIENTPROC callback

User defined callback function to receive notification of client connections and disconnections, and optionally refuse connections.

```
BOOL CALLBACK EncodeClientProc(
    HENCODE handle,
    BOOL connect,
    char *client,
    char *headers,
    void *user
);
```

Parameters

- handle The encoder/server that the client is connecting to or disconnecting from.
- connect The client is connecting? TRUE = connecting, FALSE = disconnecting.
- client The client's IP address and port number... "xxx.xxx.xxx.xxx:port".
- headers The request headers... NULL = the client is disconnecting or HTTP headers have been disabled via the BASS_ENCODE_SERVER_NOHTTP flag. The headers are in the same form as would be given by <u>BASS_ChannelGetTags</u>, which is a series of null-terminated strings, the final string ending with a double null. The request headers can optionally be replaced with response headers to send back to the client, each ending with a carriage return and line feed ("\r\n"). The response headers should not exceed 1KB in length.
- user The user instance data given when <u>BASS_Encode_ServerInit</u> was called.

Return value

If the client is connecting, FALSE means the connection is denied, otherwise it is accepted. The return value is ignored if the client is disconnecting.

Remarks

This function can be used to keep track of how many clients are connected, and who is connected. The request headers can be used to authenticate clients, and response headers can be used to pass information back to the clients. By default, connecting clients will be sent an "HTTP/1.0 200 OK" status line if accepted, and an "HTTP/1.0 403 Forbidden" status line if denied. That can be overridden in the first response header.

Disconnection notifications will be received for clients that have disconnected themselves or that have been kicked by <u>BASS_Encode_ServerKick</u>, but there will no notification of any clients that are disconnected by the encoder being freed.

Each server has its own thread that handles new connections and sends data to its clients. The notification callbacks also come from that thread, so the callback function should avoid introducing long delays as that could result in clients missing some data and delay other clients connecting.

Example

A callback function that only allows connections from the 196.168/16 network, and only 5 clients.

```
int listeners=0; // client count
BOOL CALLBACK EncodeClientProc(HENCODE handle, BOOL connect, char *{
    if (connect) {
        if (listeners==5) { // hit client limit
            strcpy(headers, "HTTP/1.0 403 Server Full\r\n"); // set
            return FALSE; // refuse the connection
        }
        if (strncmp(client, "192.168.", 8)) // not on the 196.168/16 r
            return FALSE; // refuse the connection
        listeners++; // increment the client count
    } else
        listeners--; // decrement the client count
    return TRUE;
}
```

A callback function that only allows connections with a particular "User-Agent" request header.

```
BOOL CALLBACK EncodeClientProc(HENCODE handle, BOOL connect, char *
{
    if (connect) {
        char *p=headers;
        while (*p) {
            if (!strncimp(p, "User-Agent:", 11)) { // found the User if (strcmp(p+12, "Special Agent")) // not the wanter return FALSE; // refuse the connection
            break;
        }
        p+=strlen(p)+1; // go to next header
        }
    }
    return TRUE;
}
```

See also BASS_Encode_ServerInit, BASS_Encode_ServerKick

Frees an encoder or all encoders on a channel, optionally delaying it until the queue has been processed.

```
BOOL BASS_Encode_StopEx(
    DWORD handle,
    BOOL queue
);
```

Parameters

- handle The encoder or channel handle... a HENCODE, HSTREAM, HMUSIC, or HRECORD.
- queue Wait for the queue? If so, the encoder will not be freed until any data remaining in the queue has been processed, and it will not accept any new data in the meantime.

Return value

If successful, TRUE is returned, else FALSE is returned. Use <u>BASS_ErrorGetCode</u> to get the error code.

Remarks

When an encoder has been told to wait for its queue to be processed, <u>BASS_Encode_Stop</u> (or this function with *queue = FALSE*) can be used to cancel that and free the encoder immediately.

Error codes

BASS_ERROR_HANDLE *handle* is not valid.

See also

BASS_Encode_Start, BASS_Encode_Stop