

BASS_Mixer_GetVersion

Retrieves the version of BASSmix that is loaded.

```
DWORD BASS_Mixer_GetVersion();
```

Return value

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

BASS_CONFIG_MIXER_BUFFER config option

The source channel buffer length.

```
BASS_SetConfig(  
    BASS_CONFIG_MIXER_BUFFER,  
    DWORD length  
);
```

Parameters

length The buffer length... 1 to 5 is a multiplier of the [BASS_CONFIG_BUFFER](#) setting (at the time of the mixer's creation), otherwise it is an absolute length in milliseconds.

Remarks

When a source channel has buffering enabled, the mixer will buffer the decoded data so that it is available to the [BASS_Mixer_ChannelGetData](#) and [BASS_Mixer_ChannelGetLevel](#) and [BASS_Mixer_ChannelGetLevelEx](#) functions. The source channel buffer length can be set as a multiple of the [BASS_CONFIG_BUFFER](#) setting (at the time of the mixer's creation) or as an absolute length. If it is set lower than the [BASS_CONFIG_BUFFER](#) setting and the mixer is not a decoding channel, then it will be automatically raised to match that.

Larger buffers obviously require more memory, so this should not be set higher than necessary. If a source is played at its default rate, then the buffer only needs to be as big as the mixer's playback buffer, but if it is played at a faster rate, then the buffer needs to be bigger for it to contain the data that is currently being heard from the mixer. For example, playing a channel at 2x its normal speed would require its buffer to be 2x the normal size.

The default setting is 2, for 2x the [BASS_CONFIG_BUFFER](#) setting. Changes only affect subsequently set up channel buffers. An existing channel can have its buffer reinitialized by removing and then resetting the `BASS_MIXER_BUFFER` flag via [BASS_Mixer_ChannelFlags](#).

See also

[BASS_Mixer_ChannelFlags](#), [BASS_Mixer_ChannelGetData](#),
[BASS_Mixer_ChannelGetLevel](#), [BASS_Mixer_ChannelGetLevelEx](#),
[BASS_Mixer_StreamAddChannel](#)

[BASS_GetConfig](#), [BASS_SetConfig](#)

BASS_CONFIG_MIXER_POSEX config option

How far back to keep record of source positions to make available for [BASS_Mixer_ChannelGetPositionEx](#).

```
BASS_SetConfig(  
    BASS_CONFIG_MIXER_POSEX,  
    DWORD length  
);
```

Parameters

length The length of time to back, in milliseconds.

Remarks

If a mixer is not a decoding channel (not using the `BASS_STREAM_DECODE` flag), this config setting will just be a minimum and the mixer will always have a position record at least equal to its playback buffer length, as determined by the [BASS_CONFIG_BUFFER](#) config option.

The default setting is 2000ms. Changes only affect newly created mixers, not any that already exist.

See also

[BASS_Mixer_ChannelGetPositionEx](#), [BASS_Mixer_StreamCreate](#)

BASS_CONFIG_SPLIT_BUFFER config option

The splitter buffer length.

```
BASS_SetConfig(  
    BASS_CONFIG_SPLIT_BUFFER,  
    DWORD length  
);
```

Parameters

length The buffer length in milliseconds.

Remarks

When a source has its first splitter stream created, a buffer is allocated to hold its sample data, which all of its subsequently created splitter streams will share. This config option determines how big that buffer is. The default is 2000ms.

The buffer will always be kept as empty as possible, so its size does not necessarily affect latency; it just determines how far splitter streams can drift apart before there are buffer overflow issues for those left behind.

Changes do not affect buffers that have already been allocated; any sources that have already had splitter streams created will continue to use their existing buffers.

See also

[BASS_Split_StreamCreate](#)

BASS_Mixer_StreamCreate

Creates a mixer stream.

```
HSTREAM BASS_Mixer_StreamCreate(  
    DWORD freq,  
    DWORD chans,  
    DWORD flags  
);
```

Parameters

freq	The sample rate of the mixer output.
chans	The number of channels... 1 = mono, 2 = stereo, 4 = quadrasonic, 6 = 5.1, 8 = 7.1.
flags	Any combination of these flags.
BASS_SAMPLE_8BITS	Produce 8-bit output. If neither this or the BASS_SAMPLE_FLOAT flags are specified, then the stream is 16-bit.
BASS_SAMPLE_FLOAT	Produce 32-bit floating-point output. WDM drivers or the BASS_STREAM_DECODE flag are required to use this flag in Windows. See Floating-point channels for more info.
BASS_SAMPLE_SOFTWARE	Force the stream to not use hardware mixing. Note this only applies to playback of the mixer's output; the mixing of the source channels is always performed by BASSmix.
BASS_SAMPLE_3D	Use 3D functionality. This requires that the BASS_DEVICE_3D flag was specified when calling BASS_Init , and the stream must be mono (<i>chans=1</i>). The SPEAKER flags can not be used together with this flag.
BASS_SAMPLE_FX	Enable the old implementation of DirectX 8 effects. See the DX8 effect implementations section for details. Use BASS_ChannelSetFX to add effects to the stream.
BASS_STREAM_AUTOFREE	Automatically free the stream when playback ends.
BASS_STREAM_DECODE	Mix the sample data, without playing

it. Use [BASS_ChannelGetData](#) to retrieve the mixed sample data. The `BASS_SAMPLE_3D`, `BASS_STREAM_AUTOFREE` and `SPEAKER` flags can not be used together with this flag. The `BASS_SAMPLE_SOFTWARE`, `BASS_SAMPLE_FX` and `BASS_MIXER_RESUME` flags are also ignored.

`BASS_MIXER_END`

End the stream when there are no active (including stalled) source channels, else it is never-ending.

`BASS_MIXER_NONSTOP`

Do not stop producing output when there are no active source channels, else it will be stalled until there are active sources.

`BASS_MIXER_POSEX`

Keep a record of the source positions, making it possible to account for output latency when retrieving a source position. How far back the position record goes is determined by the [BASS_CONFIG_MIXER_POSEX](#) config option. If this flag is not used and neither is the `BASS_STREAM_DECODE` flag, then the mixer will automatically have a position record of equal length to the [BASS_CONFIG_BUFFER](#) setting.

`BASS_MIXER_RESUME`

When stalled, resume the mixer immediately upon a source being added or unpaused, else it will be resumed at the next update cycle.

BASS_SPEAKER_xxx

Speaker assignment flags. These flags have no effect when the stream is more than stereo.

Return value

If successful, the new stream's handle is returned, else 0 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_INIT	BASS_Init has not been successfully called.
BASS_ERROR_NOTAVAIL	Only decoding channels (BASS_STREAM_DECODE) are allowed when using the "no sound" device. The BASS_STREAM_AUTOFREE flag is also unavailable to decoding channels.
BASS_ERROR_FORMAT	The sample format is not supported by the device/drivers. If the stream is more than stereo or the BASS_SAMPLE_FLOAT flag is used, it could be that they are not supported.
BASS_ERROR_SPEAKER	The specified SPEAKER flags are invalid. The device/drivers do not support them, they are attempting to assign a stereo stream to a mono speaker or 3D functionality is enabled.
BASS_ERROR_MEM	There is insufficient memory.
BASS_ERROR_NO3D	Could not initialize 3D support.
BASS_ERROR_UNKNOWN	Some other mystery problem!

Remarks

Source channels are "plugged" into a mixer using the [BASS_Mixer_StreamAddChannel](#) or [BASS_Mixer_StreamAddChannelEx](#) functions, and "unplugged" using the [BASS_Mixer_ChannelRemove](#) function. Sources can be added and removed at any time, so a mixer does not have a predetermined length and [BASS_ChannelGetLength](#) is not applicable. Likewise, seeking is not possible, except to position 0, as described below.

If the mixer output is being played (it is not a decoding channel), then there will be some delay in the effect of adding/removing source channels or changing their attributes being heard. This latency can be reduced by making use of the [BASS_CONFIG_BUFFER](#) and [BASS_CONFIG_UPDATEPERIOD](#) config options. The playback buffer can be flushed by calling [BASS_ChannelPlay](#) (*restart = TRUE*) or [BASS_ChannelSetPosition](#) (*pos = 0*). That can also be done to restart a mixer that has ended.

Unless the `BASS_MIXER_END` flag is specified, a mixer stream will never end. When there are no sources (or the sources have ended/stalled), it will produce no output until there is an active source. That is unless the `BASS_MIXER_NONSTOP` flag is used, in which case it will produce silent output while there are no active sources. The `BASS_MIXER_END` and `BASS_MIXER_NONSTOP` flags can be toggled at any time, using [BASS_ChannelFlags](#).

Besides mixing channels, a mixer stream can be used for sample rate conversion. In that case the *freq* parameter would be set to the new sample rate, and the source channel's attributes would be left at their defaults. A mixer stream can also be used to downmix, upmix and generally rearrange channels, using the [matrix mixing](#) features.

Platform-specific

Away from Windows, all mixing is done in software (by BASS), so the BASS_SAMPLE_SOFTWARE flag is unnecessary. The BASS_SAMPLE_FX flag is also ignored.

See also

[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#)

[BASS_ChannelPlay](#), [BASS_StreamFree](#)

BASS_Mixer_StreamAddChannel

Plugs a channel into a mixer.

```
BOOL BASS_Mixer_StreamAddChannel(  
    HSTREAM handle,  
    DWORD channel,  
    DWORD flags  
);
```


Parameters

handle	The mixer handle.
channel	The handle of the channel to plug into the mixer... a HMUSIC, HSTREAM or HRECORD.
flags	Any combination of these flags. BASS_MIXER_MATRIX Creates a channel matrix , allowing the source's channels to be sent to any of the mixer output channels, at any levels. The matrix can be retrieved and modified via the BASS_Mixer_ChannelGetMatrix and BASS_Mixer_ChannelSetMatrix functions. The matrix will initially contain a one-to-one mapping, eg. left out = left in, right out = right in, etc.
BASS_MIXER_DOWNMIX	If the source has more channels than the mixer output (and that is stereo or mono), then matrix mixing is enabled and initialized with the appropriate downmixing matrix. Note the source data is assumed to follow the standard channel ordering, as described in the STREAMPROC documentation.
BASS_MIXER_BUFFER	Buffer the sample data, for use by BASS_Mixer_ChannelGetData and BASS_Mixer_ChannelGetLevel and BASS_Mixer_ChannelGetLevelEx . This increases memory requirements, so should not be enabled needlessly. The size of the buffer can be controlled via the BASS_CONFIG_MIXER_BUFFER

BASS_MIXER_LIMIT	<p>config option.</p> <p>Limit the mixer processing to the amount of data available from this source, while the source is active (not ended). If the source stalls, then the mixer will too, rather than continuing to mix other sources, as it would normally do. This flag can only be applied to one source per mixer, so it will automatically be removed from any other source of the same mixer.</p>
BASS_MIXER_NORAMPIN	<p>Do not ramp-in the start, including after seeking</p> <p>(BASS_Mixer_ChannelSetPosition)</p> <p>This is useful for gap-less playback, where a source channel is intended to seamlessly follow another. This does not affect volume and pan changes, which are always ramped.</p>
BASS_MIXER_PAUSE	<p>Pause processing of the source. Use BASS_Mixer_ChannelFlags to resume processing.</p>
BASS_STREAM_AUTOFREE	<p>Automatically free the source channel when it ends. This allows you to add a channel to a mixer and forget about it, as it will automatically be freed when it has reached the end, or when the source is removed from the mixer or when the mixer is freed.</p>
BASS_SPEAKER_xxx	<p>Speaker assignment flags. If matrix mixing is enabled then the matrix will be initialized to place the source on the requested speaker(s), with downmixing also applied if the</p>

BASS_MIXER_DOWNMIX flag is specified. The [BASS_Init](#) BASS_DEVICE_NOSPEAKER flag has effect here.

Return value

If successful, then TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	At least one of <i>handle</i> and <i>channel</i> is not valid.
BASS_ERROR_DECODE	<i>channel</i> is not a decoding channel.
BASS_ERROR_ALREADY	<i>channel</i> is already plugged into a mixer. It must be unplugged first.
BASS_ERROR_SPEAKER	The mixer does not support the requested speaker(s), or you are attempting to assign a stereo stream to a mono speaker.

Remarks

Internally, a mixer will use the [BASS_ChannelGetData](#) function to get data from its source channels. That means that the source channels must be decoding channels (not using a [RECORDPROC](#) in the case of a recording channel). Plugging a channel into more than one mixer at a time is not possible because the mixers would be taking data away from each other. An advantage of this is that there is no need for a mixer's handle to be provided with the channel functions. It is actually possible to plug a channel into multiple mixers via the use of [splitter streams](#).

Channels are "unplugged" using the [BASS_Mixer_ChannelRemove](#) function. Channels are also automatically unplugged when they are freed.

When mixing a channel, the mixer makes use of the channel's freq/volume/pan attributes, as set with [BASS_ChannelSetAttribute](#). The [BASS_CONFIG_CURVE_VOL](#) and [BASS_CONFIG_CURVE_PAN](#) config option settings are also used.

If a multi-channel stream has more channels than the mixer output, the extra channels will be discarded. For example, if a 5.1 stream is plugged into a stereo mixer, only the front-left/right channels will be retained. That is unless matrix mixing is used.

The mixer processing is performed in floating-point, so it makes sense (for both quality and efficiency reasons) for the source channels to be floating-point too, though they do not have to be. It is also more efficient if the source channels have the same sample rate as the mixer output because no sample rate conversion is required then. When sample rate conversion is required, windowed sinc interpolation is used and the source's [BASS_ATTRIB_SRC](#) attribute determines how many points/samples are used in that, as follows: 0 (or below) = 4 points, 1 = 8 points, 2 = 16 points, 3 = 32 points, 4 = 64 points, 5 = 128 points, 6 (or above) = 256 points. 8 points are used if the [BASS_ATTRIB_SRC](#) attribute is unavailable (old BASS version). A higher number of points results in better sound quality (less aliasing and smaller transition band in the low-pass filter), but also higher CPU usage.

Platform-specific

The sample rate conversion processing is limited to 128 points on iOS and Android. The mixer processing is performed in fixed-point rather than floating-point on some platforms/architectures, as indicated by the [BASS_CONFIG_FLOAT](#) value.

See also

[BASS Mixer ChannelFlags](#), [BASS Mixer ChannelGetLevel](#),
[BASS Mixer ChannelGetMixer](#), [BASS Mixer ChannelGetPosition](#),
[BASS Mixer ChannelRemove](#), [BASS Mixer ChannelSetMatrix](#),
[BASS Mixer ChannelSetPosition](#), [BASS Mixer StreamAddChannelEx](#),
[BASS Mixer StreamCreate](#)

BASS_Mixer_StreamAddChannelEx

Plugs a channel into a mixer, optionally delaying the start and limiting the length.

```
BOOL BASS_Mixer_StreamAddChannelEx(  
    HSTREAM handle,  
    DWORD channel,  
    DWORD flags,  
    QWORD start,  
    QWORD length  
);
```

Parameters

handle	The mixer handle.
channel	The handle of the channel to plug into the mixer... a HMUSIC, HSTREAM or HRECORD.
flags	Any combination of these flags. BASS_MIXER_MATRIX Creates a channel matrix , allowing the source's channels to be sent to any of the mixer output channels, at any levels. The matrix can be retrieved and modified via the BASS_Mixer_ChannelGetMatrix and BASS_Mixer_ChannelSetMatrix functions. The matrix will initially contain a one-to-one mapping, eg. left out = left in, right out = right in, etc...
BASS_MIXER_DOWNMIX	If the source has more channels than the mixer output (and the mixer is stereo or mono), then a channel matrix is created, initialized with the appropriate downmixing matrix. Note the source data is assumed to follow the standard channel ordering, as described in the STREAMPROC documentation.
BASS_MIXER_BUFFER	Buffer the sample data, for use by BASS_Mixer_ChannelGetData and BASS_Mixer_ChannelGetLevel and BASS_Mixer_ChannelGetLevelEx . This increases memory requirements, so should not be enabled needlessly. The size of the buffer can be controlled via the

	<u>BASS_CONFIG_MIXER_BUFFER</u> config option.
BASS_MIXER_LIMIT	Limit the mixer processing to the amount of data available from this source, while the source is active (not ended). If the source stalls, then the mixer will too, rather than continuing to mix other sources, as it would normally do. This flag can only be applied to one source per mixer, so it will automatically be removed from any other source of the same mixer.
BASS_MIXER_NORAMPIN	Do not ramp-in the start, including after seeking (<u>BASS_Mixer_ChannelSetPosition</u>) This is useful for gap-less playback, where a source channel is intended to seamlessly follow another. This does not affect volume and pan changes, which are always ramped.
BASS_MIXER_PAUSE	Pause processing of the source. Use <u>BASS_Mixer_ChannelFlags</u> to resume processing.
BASS_STREAM_AUTOFREE	Automatically free the source channel when it ends. This allows you to add a channel to a mixer and forget about it, as it will automatically be freed when it has reached the end, or when the source is removed from the mixer or when the mixer is freed.
BASS_SPEAKER_XXX	<u>Speaker assignment flags</u> . Ignored when using the BASS_MIXER_MATRIX or BASS_MIXER_DOWNMIX flag.

The [BASS_Init](#)
BASS_DEVICE_NOSPEAKER
flag has effect here.

start Delay (in bytes) before the channel is mixed in.
length The maximum amount of data (in bytes) to mix... 0 = no limit. Once
 this end point is reached, the channel will be removed from the mixer.

Return value

If successful, then TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	At least one of <i>handle</i> and <i>channel</i> is not valid.
BASS_ERROR_DECODE	<i>channel</i> is not a decoding channel.
BASS_ERROR_ALREADY	<i>channel</i> is already plugged into a mixer. It must be unplugged first.
BASS_ERROR_SPEAKER	The mixer does not support the requested speaker(s), or you are attempting to assign a stereo stream to a mono speaker.

Remarks

This function is identical to [BASS_Mixer_StreamAddChannel](#), but with the additional ability to specify a delay and duration for the channel.

The *start* and *length* parameters relate to the mixer output. So when calculating these values, use the mixer stream's sample format rather than the source channel's. The *start* parameter is automatically rounded-down to the nearest sample boundary, while the *length* parameter is rounded-up to the nearest sample boundary.

Example

Add a channel to a mixer, delaying the start by 1 second and limiting the duration to 2 seconds.

```
QWORD start=BASS_ChannelSeconds2Bytes(mixer, 1); // delay
QWORD length=BASS_ChannelSeconds2Bytes(mixer, 2); // duration
BASS_Mixer_StreamAddChannelEx(mixer, channel, 0, start, length); //
```


See also

[BASS_Mixer_ChannelFlags](#), [BASS_Mixer_ChannelGetLevel](#),
[BASS_Mixer_ChannelGetMixer](#), [BASS_Mixer_ChannelGetPosition](#),
[BASS_Mixer_ChannelRemove](#), [BASS_Mixer_ChannelSetMatrix](#),
[BASS_Mixer_ChannelSetPosition](#), [BASS_Mixer_StreamAddChannel](#),
[BASS_Mixer_StreamCreate](#)

BASS_Mixer_StreamGetChannels

Retrieves a mixer's source channels.

```
DWORD BASS_Mixer_StreamGetChannels(  
    HSTREAM handle,  
    DWORD *channels,  
    DWORD count  
);
```

Parameters

- handle The mixer handle.
- channels An array to receive the mixer's source channel handles.
- count The maximum number of channels to receive in the *channels* array...
0 = get the number of source channels without getting the handles.

Return value

If successful, the number of source channels placed in the *channels* array is returned, or the total number of source channels if *count* = 0, else -1 is returned.

Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not a valid mixer handle.

Remarks

To determine whether a particular channel is plugged in a mixer, it is simpler to use [BASS_Mixer_ChannelGetMixer](#) instead of this function.

Example

Remove all source channels from a mixer.

```
DWORD *channels;
DWORD a, count;
count=BASS_Mixer_StreamGetChannels(mixer, NULL, 0); // get the number of channels
channels=(DWORD*)malloc(count*sizeof(DWORD)); // allocate channels array
BASS_Mixer_StreamGetChannels(mixer, channels, count); // get the channel IDs
for (a=0; a<count; a++) // go through them all and...
    BASS_Mixer_ChannelRemove(channels[a]); // remove from the mixer
free(channels); // free the channels array
```

See also

[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#),
[BASS_Mixer_ChannelGetMixer](#)

BASS_ATTRIB_MIXER_LATENCY attribute

Custom output latency.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIXER_LATENCY,  
    float latency  
);
```

Parameters

handle The mixer stream handle.

latency The latency in seconds.

Remarks

When a mixer is played by BASS, the [BASS_Mixer_ChannelGetData](#), [BASS_Mixer_ChannelGetLevel](#), [BASS_Mixer_ChannelGetLevelEx](#), and [BASS_Mixer_ChannelGetPosition](#) functions will get the output latency and account for that so that they reflect what is currently being heard, but that cannot be done when a different output system is used, eg. ASIO or WASAPI. In that case, this attribute can be used to tell the mixer what the output latency is, so that those functions can still account for it. The mixer needs to have the `BASS_STREAM_DECODE` and `BASS_MIXER_POSEX` flags set to use this attribute.

The default setting is 0 (no accounting for latency). Changes take immediate effect.

See also

[BASS_Mixer_ChannelGetData](#), [BASS_Mixer_ChannelGetLevel](#),
[BASS_Mixer_ChannelGetLevelEx](#), [BASS_Mixer_ChannelGetPosition](#)

[BASS_ChannelGetAttribute](#), [BASS_ChannelSetAttribute](#)

BASS_Mixer_ChannelFlags

Modifies and retrieves a channel's mixer flags.

```
DWORD BASS_Mixer_ChannelFlags(  
    DWORD handle,  
    DWORD flags,  
    DWORD mask  
);
```

Parameters

handle	The channel handle.
flags	A combination of these flags. BASS_MIXER_BUFFER Buffer the sample data, for use by BASS_Mixer_ChannelGetData and BASS_Mixer_ChannelGetLevel and BASS_Mixer_ChannelGetLevelEx . BASS_MIXER_LIMIT Limit the mixer processing to the amount of data available from this source. This flag can only be applied to one source per mixer, so it will automatically be removed from any other source of the same mixer. BASS_MIXER_NORAMPIN Do not ramp-in the start, including after seeking (BASS_Mixer_ChannelSetPosition). BASS_MIXER_PAUSE Pause processing of the source. BASS_STREAM_AUTOFREE Automatically free the source channel when it ends. BASS_SPEAKER_xxx Speaker assignment flags . If matrix mixing is enabled, then the matrix will be modified to place the source on the requested speaker(s).
mask	The flags (as above) to modify. Flags that are not included in this are left as they are, so it can be set to 0 in order to just retrieve the current flags. To modify the speaker flags, any of the BASS_SPEAKER_xxx flags can be used in the mask (no need to include all of them).

Return value

If successful, the channel's updated flags are returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.

BASS_ERROR_SPEAKER The mixer does not support the requested speaker(s), or the channel has matrix mixing enabled.

Remarks

This function only deals with the channel's mixer related flags. The channel's standard flags, for example looping (`BASS_SAMPLE_LOOP`), are unaffected; use [BASS_ChannelFlags](#) to modify them.

Example

Disable ramping-in of a channel.

```
BASS_Mixer_ChannelFlags(channel, BASS_MIXER_NORAMPIN, BASS_MIXER_NOI
```

Enable ramping-in of a channel.

```
BASS_Mixer_ChannelFlags(channel, 0, BASS_MIXER_NORAMPIN); // remove
```

See also

[BASS_Mixer_StreamAddChannel](#)

BASS_Mixer_ChannelGetData

Retrieves the immediate sample data (or an FFT representation of it) of a mixer source channel.

```
DWORD BASS_Mixer_ChannelGetData(  
    DWORD handle,  
    void *buffer,  
    DWORD length  
);
```

Parameters

handle The channel handle.

buffer Pointer to a buffer to receive the data.

length Number of bytes wanted, and/or the [BASS_ChannelGetData](#) flags.

Return value

If an error occurs, -1 is returned, use [BASS_ErrorGetCode](#) to get the error code. When requesting FFT data, the number of bytes read from the channel (to perform the FFT) is returned. When requesting sample data, the number of bytes written to *buffer* will be returned (not necessarily the same as the number of bytes read when using the BASS_DATA_FLOAT flag). When using the BASS_DATA_AVAILABLE flag, the number of bytes in the channel's buffer is returned.

Error codes

BASS_ERROR_HANDLE *handle* is not plugged into a mixer.
BASS_ERROR_NOTAVAIL The channel does not have buffering
 (BASS_MIXER_BUFFER) enabled.

Remarks

This function is like the standard [BASS_ChannelGetData](#), but it gets the data from the channel's buffer instead of decoding it from the channel, which means that the mixer does not miss out on any data. In order to do this, the source channel must have buffering enabled, via the `BASS_MIXER_BUFFER` flag.

If the mixer is being played by BASS, the returned data will be in sync with what is currently being heard from the mixer. If another output system is being used, the [BASS_ATTRIB_MIXER_LATENCY](#) option can be used to tell the mixer what the latency is so that it can be taken account of, otherwise the channel's most recent data will be returned. The [BASS_CONFIG_MIXER_BUFFER](#) config option determines how far back data will be available from, so it should be set high enough to cover any latency.

See also

[BASS_Mixer_ChannelGetLevel](#), [BASS_ATTRIB_MIXER_LATENCY](#),
[BASS_CONFIG_MIXER_BUFFER](#)

[BASS_ChannelGetData](#)

BASS_Mixer_ChannelGetEnvelopePos

Retrieves the current position and value of an envelope on a channel.

```
QWORD BASS_Mixer_ChannelGetEnvelopePos(  
    DWORD handle,  
    DWORD type,  
    float *value  
);
```

Parameters

- handle The channel handle.
- type The envelope to get the position/value of. One of the following.
- BASS_MIXER_ENV_FREQ Sample rate.
 - BASS_MIXER_ENV_VOL Volume.
 - BASS_MIXER_ENV_PAN Panning/balance.
- value Pointer to a variable to receive the envelope value at the current position... NULL = do not retrieve it.

Return value

If successful, the current position of the envelope is returned, else -1 is returned.

Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	The channel is not plugged into a mixer.
BASS_ERROR_ILLTYP	<i>type</i> is not valid.
BASS_ERROR_NOTAVAIL	There is no envelope of the requested type on the channel.

Remarks

During playback, the envelope's current position is not necessarily what is currently being heard, due to buffering.

See also

[BASS_Mixer_ChannelSetEnvelope](#), [BASS_Mixer_ChannelSetEnvelopePos](#)

BASS_Mixer_ChannelGetLevel

Retrieves the level (peak amplitude) of a mixer source channel.

```
DWORD BASS_Mixer_ChannelGetLevel(  
    DWORD handle  
);
```


Parameters

handle The channel handle.

Return value

If an error occurs, -1 is returned, use [BASS_ErrorGetCode](#) to get the error code. If successful, the level of the left channel is returned in the low word (low 16-bits), and the level of the right channel is returned in the high word (high 16-bits). If the channel is mono, then the low word is duplicated in the high word. The level ranges linearly from 0 (silent) to 32768 (max). 0 will be returned when a channel is stalled.

Error codes

BASS_ERROR_HANDLE	<i>handle</i> is not plugged into a mixer.
BASS_ERROR_NOTAVAIL	The channel does not have buffering (BASS_MIXER_BUFFER) enabled.
BASS_ERROR_NOPLAY	The mixer is not playing.

Remarks

This function is like the standard [BASS_ChannelGetLevel](#), but it gets the level from the channel's buffer instead of decoding data from the channel, which means that the mixer does not miss out on any data. In order to do this, the source channel must have buffering enabled via the `BASS_MIXER_BUFFER` flag.

This function measures the level of the channel's sample data, not its level in the mixer output. It includes the effect of any DSP/FX set on the channel, but not the effect of the channel's [BASS_ATTRIB_VOL](#) or [BASS_ATTRIB_PAN](#) attributes or [matrix mixing](#) or any envelope set via [BASS_Mixer_ChannelSetEnvelope](#).

If the mixer is being played by BASS, the returned level will be in sync with what is currently being heard from the mixer. If another output system is being used, the [BASS_ATTRIB_MIXER_LATENCY](#) option can be used to tell the mixer what the latency is so that it can be taken account of, otherwise the channel's most recent data will be used to get the level. The [BASS_CONFIG_MIXER_BUFFER](#) config option determines how far back the level will be available from, so it should be set high enough to cover any latency.

More flexible level retrieval is available with [BASS_Mixer_ChannelGetLevelEx](#).

See also

[BASS_Mixer_ChannelGetData](#), [BASS_Mixer_ChannelGetLevelEx](#),
[BASS_ATTRIB_MIXER_LATENCY](#), [BASS_CONFIG_MIXER_BUFFER](#)

[BASS_ChannelGetLevel](#)

BASS_Mixer_ChannelGetLevelEx

Retrieves the level of a mixer source channel.

```
DWORD BASS_Mixer_ChannelGetLevelEx(  
    DWORD handle,  
    float *levels,  
    float length,  
    DWORD flags  
);
```

Parameters

- levels An array to receive the levels.
- length The amount of data to inspect to calculate the level, in seconds. The maximum is 1 second. Less data than requested may be used if the full amount is not available, eg. if the source's buffer (determined by the [BASS_CONFIG_MIXER_BUFFER](#) config option) is shorter.
- flags A combination of these flags.
- BASS_LEVEL_MONO Get a mono level. If neither this or the BASS_LEVEL_STEREO flag is used, then a separate level is retrieved for each channel; the number of channels is available from [BASS_WASAPI_GetInfo](#).
- BASS_LEVEL_STEREO Get a stereo level. The left level will be from the even channels, and the right level will be from the odd channels. If there are an odd number of channels then the left and right levels will both include all channels.
- BASS_LEVEL_RMS Get the RMS level. Otherwise the peak level.

Return value

If an error occurs, -1 is returned, use [BASS_ErrorGetCode](#) to get the error code. If successful, the level of the left channel is returned in the low word (low 16-bits), and the level of the right channel is returned in the high word (high 16-bits). If the channel is mono, then the low word is duplicated in the high word. The level ranges linearly from 0 (silent) to 32768 (max). 0 will be returned when a channel is stalled.

Error codes

BASS_ERROR_HANDLE	<i>handle</i> is not plugged into a mixer.
BASS_ERROR_NOTAVAIL	The channel does not have buffering (BASS_MIXER_BUFFER) enabled.
BASS_ERROR_NOPLAY	The mixer is not playing.

Remarks

This function is like the standard [BASS_ChannelGetLevelEx](#), but it gets the level from the channel's buffer instead of decoding data from the channel, which means that the mixer does not miss out on any data. In order to do this, the source channel must have buffering enabled via the `BASS_MIXER_BUFFER` flag.

This function measures the level of the channel's sample data, not its level in the mixer output. It includes the effect of any DSP/FX set on the channel, but not the effect of the channel's [BASS_ATTRIB_VOL](#) or [BASS_ATTRIB_PAN](#) attributes or [matrix mixing](#) or any envelope set via [BASS_Mixer_ChannelSetEnvelope](#).

If the mixer is being played by BASS, the returned level will be in sync with what is currently being heard from the mixer. If another output system is being used, the [BASS_ATTRIB_MIXER_LATENCY](#) option can be used to tell the mixer what the latency is so that it can be taken account of, otherwise the channel's most recent data will be used to get the level. The [BASS_CONFIG_MIXER_BUFFER](#) config option determines how far back the level will be available from, so it should be set high enough to cover any latency.

See also

[BASS_Mixer_ChannelGetData](#), [BASS_Mixer_ChannelGetLevel](#),
[BASS_ATTRIB_MIXER_LATENCY](#), [BASS_CONFIG_MIXER_BUFFER](#)

[BASS_ChannelGetLevelEx](#)

BASS_Mixer_ChannelGetMatrix

Retrieves a channel's mixing matrix, if it has one.

```
BOOL BASS_Mixer_ChannelGetMatrix(  
    DWORD handle,  
    void *matrix  
);
```

Parameters

- handle The channel handle.
- matrix Location to write the matrix.

Return value

If successful, a TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.
BASS_ERROR_NOTAVAIL The channel is not using matrix mixing.

Example

Get the matrix of a stereo channel plugged into a quad mixer.

```
float matrix[4][2]; // 4x2 array to receive the matrix  
BASS_Mixer_ChannelGetMatrix(handle, matrix); // get the matrix
```


See also

[BASS Mixer ChannelSetMatrix](#), [BASS Mixer StreamAddChannel](#),
[BASS Mixer StreamAddChannelEx](#)

BASS_Mixer_ChannelGetMixer

Retrieves the mixer that a channel is plugged into.

```
HSTREAM BASS_Mixer_ChannelGetMixer(  
    DWORD handle  
);
```

Parameters

handle The channel handle.

Return value

If successful, the mixer stream's handle is returned, else 0 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.

See also

[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#),
[BASS_Mixer_StreamGetChannels](#)

BASS_Mixer_ChannelGetPosition

Retrieves the playback position of a mixer source channel.

```
QWORD BASS_Mixer_ChannelGetPosition(  
    DWORD handle,  
    DWORD mode  
);
```

Parameters

handle The channel handle.

mode How to retrieve the position. One of the following.

BASS_POS_BYTE Get the position in bytes.

BASS_POS_MUSIC_ORDER Get the position in orders and rows...
LOWORD = order, HIWORD = row *
scaler

(BASS_ATTRIB_MUSIC_PSCALER
(HMUSIC only)

other modes may be supported by add-ons, see the documentation.

Return value

If successful, then the channel's position is returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not plugged into a mixer.
BASS_ERROR_NOTAVAIL The requested position is not available.
BASS_ERROR_UNKNOWN Some other mystery problem!

Remarks

This function is like the standard [BASS_ChannelGetPosition](#), but it compensates for the mixer's playback buffering to return the position that is currently being heard. If the mixer is not being played by BASS, it is possible to account for any other output system latency with the [BASS_ATTRIB_MIXER_LATENCY](#) option or the [BASS_Mixer_ChannelGetPositionEx](#) function.

See also

[BASS_Mixer_ChannelGetPositionEx](#), [BASS_Mixer_ChannelSetPosition](#),
[BASS_Mixer_ChannelSetSync](#), [BASS_ATTRIB_MIXER_LATENCY](#)

[BASS_ChannelGetPosition](#)

BASS_Mixer_ChannelGetPositionEx

Retrieves the playback position of a mixer source channel, optionally accounting for some latency.

```
QWORD BASS_Mixer_ChannelGetPositionEx(  
    DWORD handle,  
    DWORD mode,  
    DWORD delay  
);
```

Parameters

- handle The channel handle.
- mode How to retrieve the position. One of the following.
- BASS_POS_BYTE Get the position in bytes.
 - BASS_POS_MUSIC_ORDER Get the position in orders and rows...
LOWORD = order, HIWORD = row *
scaler
([BASS_ATTRIB_MUSIC_PSCALER](#)
(HMUSIC only)
- other modes may be supported by add-ons, see the documentation.*
- delay How far back (in bytes) in the mixer output to get the source channel's position from.

Return value

If successful, then the channel's position is returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	<i>handle</i> is not plugged into a mixer.
BASS_ERROR_NOTAVAIL	The requested position <i>mode</i> is not available, or <i>delay</i> goes beyond where the mixer has record of the source channel's position.
BASS_ERROR_UNKNOWN	Some other mystery problem!

Remarks

[BASS Mixer ChannelGetPosition](#) compensates for the mixer's playback buffering to give the position that is currently being heard, but if the mixer is feeding some other output system, it will not know how to compensate for that. This function fills that gap by allowing the latency to be specified in the call. This functionality requires the mixer to keep a record of its sources' position going back some time, and that is enabled via the `BASS_MIXER_POSEX` flag when a mixer is created, with the [BASS_CONFIG_MIXER_POSEX](#) config option determining how far back the position record goes. If the mixer is not a decoding channel (not using the `BASS_STREAM_DECODE` flag), then it will automatically have a position record at least equal to its playback buffer length.

See also

[BASS_Mixer_ChannelGetPosition](#), [BASS_CONFIG_MIXER_POSEX](#)

[BASS_ChannelGetPosition](#)

BASS_Mixer_ChannelRemove

Unplugs a channel from a mixer.

```
BOOL BASS_Mixer_ChannelRemove(  
    DWORD handle  
);
```

Parameters

handle The handle of the channel to unplug.

Return value

If successful, then TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.

See also

[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_ChannelGetMixer](#)

BASS_Mixer_ChannelRemoveSync

Removes a synchronizer from a mixer source channel.

```
BOOL BASS_Mixer_ChannelRemoveSync(  
    DWORD handle,  
    HSYNC sync  
);
```


Parameters

handle The channel handle.

sync Handle of the synchronizer to remove.

Return value

If successful, TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE At least one of *handle* and *sync* is not valid.

Remarks

This function can only remove syncs that were set via [BASS_Mixer_ChannelSetSync](#), not those that were set via [BASS_ChannelSetSync](#).

See also

[BASS_Mixer_ChannelSetSync](#)

[BASS_ChannelRemoveSync](#)

BASS_Mixer_ChannelSetEnvelope

Sets an envelope to modify the sample rate, volume or pan of a channel over a period of time.

```
BOOL BASS_Mixer_ChannelSetEnvelope(  
    DWORD handle,  
    DWORD type,  
    BASS\_MIXER\_NODE *nodes,  
    DWORD count  
);
```

Parameters

handle	The channel handle.
type	The attribute to modify with the envelope. One of the following. BASS_MIXER_ENV_FREQ Sample rate. BASS_MIXER_ENV_VOL Volume. BASS_MIXER_ENV_PAN Panning/balance. BASS_MIXER_ENV_LOOP Loop the envelope. This is a flag and can be used in combination with any of the above.
nodes	The array of envelope nodes, which should have sequential positions.
count	The number of elements in the <i>nodes</i> array... 0 = no envelope.

Return value

If successful, TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.

BASS_ERROR_ILLTYP *e* is not valid.

Remarks

Envelopes are applied on top of the channel's attributes, as set via [BASS_ChannelSetAttribute](#). In the case of `BASS_MIXER_ENV_FREQ` and `BASS_MIXER_ENV_VOL`, the final sample rate and volume is a product of the channel attribute and the envelope. While in the `BASS_MIXER_ENV_PAN` case, the final panning is a sum of the channel attribute and envelope.

[BASS_Mixer_ChannelSetEnvelopePos](#) and [BASS_Mixer_ChannelGetEnvelopePos](#) can be used to set and get the current envelope position. A `BASS_SYNC_MIXER_ENVELOPE` sync can be set via [BASS_Mixer_ChannelSetSync](#) to be informed of when an envelope ends. This function can be called again from such a sync, in order to set a new envelope to follow on from the old one.

Any previous envelope of the same type is replaced by the new envelope. A copy is made of the *nodes* array, so it does not need to persist beyond this function call.

Example

Set an envelope to bounce the pan position between left and right every 4 seconds.

```
BASS_MIXER_NODE nodes[4];
nodes[0].pos=0;
nodes[0].val=0; // start at centre
nodes[1].pos=BASS_ChannelSeconds2Bytes(mixer, 1);
nodes[1].val=-1; // full left after 1 second
nodes[2].pos=BASS_ChannelSeconds2Bytes(mixer, 3);
nodes[2].val=1; // full right after 3 seconds
nodes[3].pos=BASS_ChannelSeconds2Bytes(mixer, 4);
nodes[3].val=0; // back at centre after 4 seconds
BASS_Mixer_ChannelSetEnvelope(channel, BASS_MIXER_ENV_PAN|BASS_MIXE
```

See also

[BASS_Mixer_ChannelGetEnvelopePos](#), [BASS_Mixer_ChannelSetEnvelopePos](#),
[BASS_MIXER_NODE](#) structure

BASS_Mixer_ChannelSetEnvelopePos

Sets the current position of an envelope on a channel.

```
BOOL BASS_Mixer_ChannelSetEnvelopePos(  
    DWORD handle,  
    DWORD type,  
    QWORD pos  
);
```

Parameters

- handle The channel handle.
- type The envelope to set the position of. One of the following.
BASS_MIXER_ENV_FREQ Sample rate.
BASS_MIXER_ENV_VOL Volume.
BASS_MIXER_ENV_PAN Panning/balance.
- pos The new envelope position, in bytes. If this is beyond the end of the envelope it will be capped or looped, depending on whether the envelope has looping enabled.

Return value

If successful, the current position of the envelope is returned, else -1 is returned.

Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	The channel is not plugged into a mixer.
BASS_ERROR_ILLTYP	<i>type</i> is not valid.
BASS_ERROR_NOTAVAIL	There is no envelope of the requested type on the channel.

Remarks

During playback, the effect of changes are not heard instantaneously, due to buffering. To reduce the delay, use the [BASS_CONFIG_BUFFER config option](#) config option to reduce the buffer length.

See also

[BASS_Mixer_ChannelGetEnvelopePos](#), [BASS_Mixer_ChannelSetEnvelope](#)

BASS_Mixer_ChannelSetMatrix

Sets a channel's mixing matrix.

```
BOOL BASS_Mixer_ChannelSetMatrix(  
    DWORD handle,  
    void *matrix  
);
```

Parameters

handle The channel handle.
matrix Pointer to the matrix.

Return value

If successful, a TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.
BASS_ERROR_NOTAVAIL The channel is not using matrix mixing.

Remarks

See the [matrix mixing](#) documentation for examples.

See also

[BASS_Mixer_ChannelGetMatrix](#), [BASS_Mixer_ChannelSetMatrixEx](#),
[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#)

BASS_Mixer_ChannelSetMatrixEx

Sets a channel's mixing matrix, transitioning from the current matrix.

```
BOOL BASS_Mixer_ChannelSetMatrixEx(  
    DWORD handle,  
    void *matrix,  
    float time  
);
```

Parameters

- handle The channel handle.
- matrix Pointer to the matrix.
- time The time to take (in seconds) to transition from the current matrix to the specified matrix.

Return value

If successful, a TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE The channel is not plugged into a mixer.
BASS_ERROR_NOTAVAIL The channel is not using matrix mixing.

Remarks

The function is identical to [BASS_Mixer_ChannelSetMatrix](#) but with the option of transitioning over time to the specified matrix. If this function or [BASS_Mixer_ChannelSetMatrix](#) is called while a previous matrix transition is still in progress, then that transition will be stopped. If [BASS_Mixer_ChannelGetMatrix](#) is called mid-transition, it will give the mid-transition matrix values.

See also

[BASS_Mixer_ChannelGetMatrix](#), [BASS_Mixer_ChannelSetMatrix](#),
[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#)

BASS_Mixer_ChannelSetPosition

Sets the playback position of a mixer source channel.

```
BOOL BASS_Mixer_ChannelSetPosition(  
    DWORD handle,  
    QWORD pos,  
    DWORD mode  
);
```

Parameters

handle	The channel handle.
pos	The position, in units determined by the <i>mode</i> .
mode	How to set the position. One of the following, with optional flags.
BASS_POS_BYTE	The position is in bytes, which will be rounded down to the nearest sample boundary.
BASS_POS_MUSIC_ORDER	The position is in orders and rows... use MAKELONG(<i>order,row</i>). (HMUSIC only)
BASS_POS_DECODETO	Flag: Decode/render up to the position rather than seeking to it. This is useful for streams that are unseekable or that have inexact seeking, but it is generally slower than normal seeking and the requested position cannot be behind the current decoding position. This flag can only be used with the BASS_POS_BYTE mode.
BASS_MUSIC_POSRESET	Flag: Stop all notes. This flag is applied automatically if it has been set on the channel, eg. via BASS_ChannelFlags . (HMUSIC)
BASS_MUSIC_POSRESETEX	Flag: Stop all notes and reset bpm/etc. This flag is applied automatically if it has been set on the channel, eg. via BASS_ChannelFlags . (HMUSIC)
BASS_MIXER_NORAMPIN	Flag: Do not ramp-in the start after seeking. This flag is applied automatically if it has been set on the channel, eg. via BASS_Mixer_ChannelFlags .
BASS_POS_MIXER_RESET	Flag: Flush the mixer's playback

buffer, so that the new position is heard immediately in the mixer output. This generally should not be used when the mixer is playing multiple sources, as it will cause a skip in the sound of the other sources. This flag has no effect if the mixer has the `BASS_STREAM_DECODE` flag set, as the mixer does not have a playback buffer then.

other modes & flags may be supported by add-ons, see the documentation.

Return value

If successful, then TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	<i>handle</i> is not plugged into a mixer.
BASS_ERROR_NOTFILE	The stream is not a file stream.
BASS_ERROR_POSITION	The requested position is invalid, eg. it is beyond the end or the download has not yet reached it.
BASS_ERROR_NOTAVAIL	The requested <i>mode</i> is not available. Invalid flags are ignored and do not result in this error.
BASS_ERROR_UNKNOWN	Some other mystery problem!

Remarks

This function works exactly like the standard [BASS_ChannelSetPosition](#), except that it also resets things for the channel in the mixer, as well as supporting the BASS_MIXER_NORAMPIN and BASS_POS_MIXER_RESET flags.

For custom looping purposes (eg. in a mixtime [SYNCPROC](#)), the standard [BASS_ChannelSetPosition](#) function should be used instead of this.

See also

[BASS_Mixer_ChannelGetPosition](#) [BASS_ChannelSetPosition](#)

BASS_Mixer_ChannelSetSync

Sets up a synchronizer on a mixer source channel.

```
HSYNC BASS_Mixer_ChannelSetSync(  
    DWORD handle,  
    DWORD type,  
    QWORD param,  
    SYNCPROC *proc,  
    void *user  
);
```

Parameters

- handle** The channel handle.
- type** The type of sync. This can be one of the standard sync types, as available via [BASS_ChannelSetSync](#), or one of the mixer specific sync types listed below.
- param** The sync parameter.
- proc** The callback function.
- user** User instance data to pass to the callback function.

Sync types, with *param* and [SYNCPROC](#) *data* definitions.

- BASS_SYNC_MIXER_ENVELOPE** Sync when an envelope ends. This is not triggered by looping envelopes.
param : envelope type to sync on, 0 = all types. *data* : envelope type.
- BASS_SYNC_MIXER_ENVELOPE_NODE** Sync when an envelope reaches a new node.
param : envelope type to sync on, 0 = all types. *data* : LOWORD = envelope type, HIWORD = node number.
- BASS_SYNC_STALL** Sync when mixing of the channel is stalled/resumed. This is like the standard **BASS_SYNC_STALL** sync, except it can be either mixtime or not.
param : not used. *data* : 0 = stalled, 1 = resumed.

Return value

If successful, then the new synchronizer's handle is returned, else 0 is returned.

Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE	The channel is not plugged into a mixer.
BASS_ERROR_ILLTTYPE	An illegal <i>type</i> was specified.
BASS_ERROR_ILLPARAM	An illegal <i>param</i> was specified.

Remarks

When used on a decoding channel (eg. a mixer source channel), syncs set with [BASS_ChannelSetSync](#) are automatically "mixtime", which means that they will be triggered as soon as the sync event is encountered. But if the mixer output is being played, then there is a playback buffer involved, which will delay the hearing of the sync event. This function compensates for that, delaying the triggering of the sync until the event is actually heard.

Sync types that would automatically be mixtime when using [BASS_ChannelSetSync](#) are not so when using this function. The `BASS_SYNC_MIXTIME` flag should be specified in those cases, or [BASS_ChannelSetSync](#) used instead.

If the mixer itself is a decoding channel, or the `BASS_SYNC_MIXTIME` flag is used, then there is effectively no real difference between this function and [BASS_ChannelSetSync](#), except for the mixer specific sync types listed above.

When a source is removed from a mixer, any syncs that have been set on it via this function are automatically removed. If the channel is subsequently plugged back into a mixer, the previous syncs will not still be set on it. Syncs set via [BASS_ChannelSetSync](#) are unaffected.

See also

[BASS_Mixer_ChannelRemoveSync](#)

[BASS_ChannelSetSync](#), [SYNCPROC callback](#)

BASS_MIXER_NODE

Used with [BASS_Mixer_ChannelSetEnvelope](#) to set an envelope on a channel.

```
typedef struct {  
    QWORD pos;  
    float val;  
} BASS_MIXER_NODE;
```

Members

- pos The position of the node in bytes. This is based on the mixer's sample format, not the source channel's format.
- val The envelope value at the position.

See also

[BASS Mixer ChannelSetEnvelope](#)

BASS_Split_StreamCreate

Creates a splitter stream.

```
HSTREAM BASS_Split_StreamCreate(  
    DWORD channel,  
    DWORD flags,  
    int *chanmap  
);
```

Parameters

channel	The handle of the channel to split... a HMUSIC, HSTREAM or HRECORD.
flags	Any combination of these flags.
BASS_SAMPLE_SOFTWARE	Force the stream to not use hardware mixing.
BASS_SAMPLE_3D	Use 3D functionality. This requires that the BASS_DEVICE_3D flag was specified when calling BASS_Init , and the stream must be mono. The SPEAKER flags can not be used together with this flag.
BASS_SAMPLE_FX	Enable the old implementation of DirectX 8 effects. See the DX8 effect implementations section for details. Use BASS_ChannelSetFX to add effects to the stream.
BASS_STREAM_AUTOFREE	Automatically free the stream when playback ends.
BASS_STREAM_DECODE	Render the sample data, without playing it. Use BASS_ChannelGetData to retrieve the sample data. The BASS_SAMPLE_3D, BASS_STREAM_AUTOFREE and SPEAKER flags can not be used together with this flag. The BASS_SAMPLE_SOFTWARE and BASS_SAMPLE_FX flags are also ignored.
BASS_SPLIT_POS	The splitter's length and position is based on the splitter's (rather

	than the source's) channel count.
BASS_SPLIT_SLAVE	Only get data from the splitter buffer, not directly from the source.
BASS_SPEAKER_XXX	<u>Speaker assignment flags</u> . These flags have no effect when the stream is more than stereo.
chanmap	Channel mapping... pointer to an array of channel indexes (0=first, -1=end of array), NULL = a 1:1 mapping of the source.

Return value

If successful, the new stream's handle is returned, else 0 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_INIT	BASS_Init has not been successfully called.
BASS_ERROR_HANDLE	<i>channel</i> is not valid.
BASS_ERROR_DECODE	<i>channel</i> is not a decoding channel.
BASS_ERROR_ILLPARAM	<i>chanmap</i> contains an invalid channel index.
BASS_ERROR_NOTAVAIL	Only decoding channels (BASS_STREAM_DECODE) are allowed when using the "no sound" device. The BASS_STREAM_AUTOFREE flag is also unavailable to decoding channels.
BASS_ERROR_FORMAT	The sample format is not supported by the device/drivers. If the stream is more than stereo or the BASS_SAMPLE_FLOAT flag is used, it could be that they are not supported.
BASS_ERROR_SPEAKER	The specified SPEAKER flags are invalid. The device/drivers do not support them, they are attempting to assign a stereo stream to a mono speaker or 3D functionality is enabled.
BASS_ERROR_MEM	There is insufficient memory.
BASS_ERROR_NO3D	Could not initialize 3D support.
BASS_ERROR_UNKNOWN	Some other mystery problem!

Remarks

A "splitter" basically does the opposite of a mixer: it splits a single source into multiple streams rather than mixing multiple sources into a single stream. Like mixer sources, splitter sources must be decoding channels.

The splitter stream will have the same sample rate and resolution as its source, but it can have a different number of channels, as dictated by the *chanmap* parameter. Even when the number of channels is different (and so the amount of data produced is different), [BASS_ChannelGetLength](#) will give the source length, and [BASS_ChannelGetPosition](#) will give the source position that is currently being output by the splitter stream, unless the BASS_SPLIT_POS flag is used. The BASS_SPLIT_POS flag can be toggled at any time via [BASS_ChannelFlags](#).

All splitter streams with the same source share a buffer to access its sample data. The length of the buffer is determined by the [BASS_CONFIG_SPLIT_BUFFER](#) config option; the splitter streams should not be allowed to drift apart beyond that, otherwise those left behind will suffer buffer overflows. Data will usually be requested from the source only when it is needed, but it can also be gotten ahead of time asynchronously instead via the [BASS_ATTRIB_SPLIT_ASYNCBUFFER](#) attribute, so that it is ready for the splitter(s) to access immediately when needed.

If the BASS_SPLIT_SLAVE flag is used, the splitter stream will only receive data from the buffer and will not request more data from the source, so it can only receive data that has already been received by another splitter stream with the same source. The BASS_SPLIT_SLAVE flag can be toggled at any time via [BASS_ChannelFlags](#).

When [BASS_ChannelSetPosition](#) is used on a splitter stream, its source will be set to the requested position and the splitter stream's buffer state will be reset so that it immediately receives data from the new position. The position change will affect all of the source's splitter streams, but the others will not have their buffer state reset; they will continue to receive any buffered data before reaching the data from the new position. [BASS_Split_StreamReset](#) can be used to reset the buffer state; that can also be used to reset a splitter stream that has ended so that it can be played again.

When a source is freed, all of its splitter streams are automatically freed.

Platform-specific

Away from Windows, all mixing is done in software (by BASS), so the BASS_SAMPLE_SOFTWARE flag is unnecessary. The BASS_SAMPLE_FX flag is also ignored.

Example

Create a splitter stream from a stereo source with the channels reversed.

```
int chanmap[]={1, 0, -1}; // channel mapping: left = source right, right = source left  
split=BASS_Split_StreamCreate(source, 0, chanmap); // create the splitter stream
```

See also

[BASS_Split_StreamGetSource](#), [BASS_Split_StreamReset](#),
[BASS_Split_StreamResetEx](#), [BASS_ATTRIB_SPLIT_ASYNCBUFFER](#),
[BASS_CONFIG_SPLIT_BUFFER](#)

[BASS_ChannelPlay](#), [BASS_StreamFree](#)

BASS_Split_StreamGetAvailable

Retrieves the amount of buffered data available to a splitter stream, or the amount of data in a splitter source buffer.

```
DWORD BASS_Split_StreamGetAvailable(  
    DWORD handle  
);
```

Parameters

handle The splitter or source handle.

Return value

If successful, then the amount of buffered data (in bytes) is returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is neither a splitter stream or source.

Remarks

When used on a splitter source, this function reports how much data is in the buffer that is shared by all of its splitter streams. When used on a splitter stream, this function reports how much data is ahead of it in the buffer, before it will receive any new data from the source. A splitter stream can be repositioned within the buffer via the [BASS_Split_StreamResetEx](#) function.

The amount of data that can be buffered is limited by the buffer size, which is determined by the [BASS_CONFIG_SPLIT_BUFFER](#) config option.

The returned buffered byte count is always based on the source's sample format, even with splitter streams that were created with a different channel count.

See also

[BASS_Split_StreamResetEx](#), [BASS_CONFIG_SPLIT_BUFFER](#)

BASS_Split_StreamGetSource

Retrieves the source of a splitter stream.

```
DWORD BASS_Split_StreamGetSource(  
    HSTREAM handle  
);
```

Parameters

handle The splitter stream handle.

Return value

If successful, the source channel's handle is returned, else 0 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not a splitter stream.

See also

[BASS_Split_StreamCreate](#), [BASS_Split_StreamGetSplits](#)

BASS_Split_StreamGetSplits

Retrieves the splitter streams of a channel.

```
DWORD BASS_Split_StreamGetSplits(  
    DWORD handle,  
    HSTREAM *splits,  
    DWORD count  
);
```

Parameters

- handle The channel handle... a HMUSIC, HSTREAM or HRECORD.
- splits An array to receive the splitter stream handles.
- count The maximum number of handles to receive in the *splits* array... 0 = get the number of splitters that the channel has without getting the handles.

Return value

If successful, the number of splitter streams placed in the *splits* array is returned, or the total number of splitter streams if *count* = 0, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* has never had any splitter streams.

See also

[BASS_Split_StreamGetSource](#)

BASS_Split_StreamReset

Resets a splitter stream or all splitter streams of a source.

```
BOOL BASS_Split_StreamReset(  
    DWORD handle  
);
```

Parameters

handle The splitter or source handle.

Return value

If successful, TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is neither a splitter stream or source.

Remarks

This function resets the splitter stream's buffer state, so that the next sample data that it receives will be from the source's current position. If the stream has ended, that is reset too, so that it can be played again. Unless called from within a mixtime sync callback, the stream's output buffer (if it has one) is also flushed.

See also

[BASS_Split_StreamCreate](#), [BASS_Split_StreamResetEx](#)

BASS_Split_StreamResetEx

Resets a splitter stream and sets its position in the source buffer.

```
BOOL BASS_Split_StreamResetEx(  
    DWORD handle,  
    DWORD offset  
);
```

Parameters

- handle The splitter handle.
- offset How far back (in bytes) to position the splitter in the source buffer. This is based on the source's sample format, which may have a different channel count to the splitter.

Return value

If successful, TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.

Error codes

BASS_ERROR_HANDLE *handle* is not a splitter stream.

Remarks

This function is the same as [BASS_Split_StreamReset](#) except that it also provides the ability to position the splitter stream within the buffer that is shared by all of the splitter streams of the same source. A splitter stream's buffer position determines what data it will next receive. For example, if its position is half a second back, it will receive half a second of buffered data before receiving new data from the source. Calling this function with *offset* = 0 will result in the next data that the splitter stream receives being new data from the source, and is identical to using [BASS_Split_StreamReset](#).

offset is automatically limited to the amount of data that the source buffer contains, which is in turn limited to the buffer size, determined by the [BASS_CONFIG_SPLIT_BUFFER](#) config option. The amount of source data buffered, as well as a splitter stream's position within it, is available from [BASS_Split_StreamGetAvailable](#).

Example

Create a new splitter stream that will first play 0.5s of already buffered data (if available).

```
split=BASS_Split_StreamCreate(source, 0, NULL); // create a splitter stream
BASS_Split_StreamResetEx(split, BASS_ChannelSeconds2Bytes(source, 0.5)); // reset to 0.5s
BASS_ChannelPlay(split, FALSE); // start playing
```

See also

[BASS_Split_StreamGetAvailable](#), [BASS_Split_StreamReset](#),
[BASS_CONFIG_SPLIT_BUFFER](#)

BASS_ATTRIB_SPLIT_ASYNCBUFFER attribute

Amount of data to asynchronously buffer from a splitter's source.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_SPLIT_ASYNCBUFFER,  
    float buffer  
);
```

Parameters

handle The splitter stream handle.

buffer The amount to buffer, in seconds... 0 = disable asynchronous buffering. The asynchronous buffering will be limited to the splitter's buffer length, as determined by [BASS_CONFIG_SPLIT_BUFFER](#).

Remarks

A splitter stream will usually get data from its source only when it is needed. This attribute allows the data to be gotten ahead of time asynchronously instead, so that it is ready for the splitter to access immediately when needed. This is not really useful with normal BASS playback (which is already buffered) but it can be used to implement buffering in other cases, eg. for mixer sources. The setting applies to all splitter streams that have the same source

When there are multiple splitters with the same source, the asynchronous buffering is based on the most advanced of them, which means that the asynchronous buffer length should be under the splitter buffer length ([BASS_CONFIG_SPLIT_BUFFER](#)) to allow the splitter positions to get apart from each other without the buffer overflowing for any of them. That margin should be at least equal to the maximum amount that you expect the splitter positions to get apart at any time.

By default, the asynchronous buffering will try to fill any space in the buffer in one data request of the source. It can be broken down into smaller amounts via the [BASS_ATTRIB_SPLIT_ASYNCPERIOD](#) attribute.

If a splitter stream needs more data than has been buffered then it will revert to synchronously getting data from the source for the remainder, unless it has the `BASS_SPLIT_SLAVE` flag set.

The amount of data that a splitter has buffered can be retrieved from [BASS_Split_StreamGetAvailable](#).

The default setting is 0 (no asynchronous buffering). Changes take immediate effect.

See also

[BASS_Split_StreamCreate](#), [BASS_Split_StreamGetAvailable](#),
[BASS_ATTRIB_SPLIT_ASYNCPERIOD](#)

[BASS_ChannelGetAttribute](#), [BASS_ChannelSetAttribute](#)

BASS_ATTRIB_SPLIT_ASYNCPERIOD attribute

Maximum amount of data to asynchronously buffer at a time from a splitter's source.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_SPLIT_ASYNCPERIOD,  
    float period  
);
```

Parameters

handle The splitter stream handle.

period The maximum amount to data to asynchronously buffer at a time from the source, in seconds... 0 = as much as possible.

Remarks

When asynchronous buffering is enabled via the [BASS_ATTRIB_SPLIT_ASYNCBUFFER](#) attribute, this attribute limits how much data is requested from the source at a time. When there is more space available in the buffer, the request will be repeated until the space is filled.

The setting applies to all splitter streams that have the same source. The default setting is 0 (as much as possible). Changes take immediate effect.

See also

[BASS_Split_StreamCreate](#), [BASS_Split_StreamGetAvailable](#),
[BASS_ATTRIB_SPLIT_ASYNCBUFFER](#)

[BASS_ChannelGetAttribute](#), [BASS_ChannelSetAttribute](#)

Matrix mixing

Normally when mixing channels, the source channels are sent to the output in the same order; the left input is sent to the left output, and so on. Sometimes something a bit more complex than that is required. For example, if the source has more channels than the output, you may want to "downmix" the source so that all channels are present in the output. Equally, if the source has fewer channels than the output, you may want to "upmix" it so that all output channels have sound. Or you may just want to rearrange the channels. Matrix mixing allows all of these.

A matrix mixer is created on a per-source basis (you can mix'n'match normal and matrix mixing), by using the `BASS_MIXER_MATRIX` and/or `BASS_MIXER_DOWNMIX` flag when calling [BASS_Mixer_StreamAddChannel](#) or [BASS_Mixer_StreamAddChannelEx](#). The matrix itself is a 2-dimensional array of floating-point mixing levels, with the source channels on one axis, and the output channels on the other. Some simple examples are shown below.

When using matrix mixing, the source channel's volume attribute still has effect, but the pan attribute does not. Whenever necessary, panning changes can be achieved by modifying the matrix.

Example

In = stereo, Out = stereo.

```
float matrix[2][2]={
    {1, 0}, // left out = left in
    {0, 1} // right out = right in
};
BASS_Mixer_ChannelSetMatrix(handle, matrix); // apply the matrix
```

In = stereo, Out = swapped stereo.

```
float matrix[2][2]={
    {0, 1}, // left out = right in
    {1, 0} // right out = left in
};
BASS_Mixer_ChannelSetMatrix(handle, matrix); // apply the matrix
```

In = stereo, Out = mono.

```
float matrix[1][2]={
    {0.5, 0.5} // mono out = half left + right in
};
BASS_Mixer_ChannelSetMatrix(handle, matrix); // apply the matrix
```

In = stereo, Out = quadraphonic (4 channels).

```
float matrix[4][2]={
    {1, 0}, // left front out = left in
    {0, 1}, // right front out = right in
    {1, 0}, // left rear out = left in
    {0, 1} // right rear out = right in
};
BASS_Mixer_ChannelSetMatrix(handle, matrix); // apply the matrix
```

See also

[BASS_Mixer_ChannelGetMatrix](#), [BASS_Mixer_ChannelSetMatrix](#),
[BASS_Mixer_StreamAddChannel](#), [BASS_Mixer_StreamAddChannelEx](#)