As with every release, all functions/etc affected by a change are listed in the "History" section of the BASS.TXT file. To make upgrading existing code simple, some brief guidelines are also listed below.

BASS_GetDeviceDescription, BASS_RecordGetDeviceDescription
These functions have been superseded by the new BASS_GetDeviceInfo and BASS_RecordGetDeviceInfo functions, respectively.

BASS_INFO, BASS_RECORDINFO
The "driver" member has been moved to the new BASS_DEVICEINFO structure.

BASS_Init, BASS_RecordInit
The default device (-1) is no longer hardcoded to the first device on Windows (already the case on OSX).

BASS_ChannelGetAttributes, BASS_ChannelSetAttributes, BASS_ChannelSlideAttributes
These functions have been replaced by BASS_ChannelGetAttribute, BASS_ChannelSetAttribute and BASS_ChannelSlideAttribute, respectively.

BASS_ChannelIsSliding
An "attrib" parameter has been added to check if a specific attribute is sliding.

BASS_MusicGetAttribute, BASS_MusicSetAttribute
These functions have been incorporated into BASS_ChannelGetAttribute and BASS_ChannelSetAttribute. The channel and instrument volume settings are also now floating-point, with a range of 0 to 1. The attribute names have also changed slightly: the "MUSIC" and "ATTRIB" parts are swapped around.

BASS_ChannelGetLength, BASS_ChannelGetPosition, BASS_ChannelSetPosition
A "mode" parameter has been added to these 3 functions. Set that to BASS_POS_BYTE (or 0) to maintain the previous functionality.
BASS_MusicGetOrders, BASS_MusicGetOrderPosition, MAKEMUSICPOS
These functions (and macro) have been incorporated into
BASS_ChannelGetLength, BASS_ChannelGetPosition and
BASS_ChannelSetPosition, via the BASS_POS_MUSIC_ORDER "mode".

BASS_MusicLoad
The "offset" parameter is now 64-bit (QWORD).

BASS_SampleLoad
The "offset" parameter is now 64-bit (QWORD).

BASS_SampleCreate
This function now returns a sample handle. The sample's data can be set using
the new BASS_SampleSetData function.

BASS_SampleCreateDone
This function is no longer required following the change to
BASS_SampleCreate.

BASS_StreamCreateFile
The "offset" and "length" parameters are now 64-bit (QWORD).

BASS_StreamGetPosition
The return value is now 64-bit (QWORD).

BASS_GetVolume, BASS_SetVolume
The volume is now a floating-point value, with a range of 0 to 1.

BASS_CONFIG_MAXVOL
This config option has been removed, as it is no longer needed following the
advent of floating-point volume settings.

BASS_CONFIG_GVOL_MUSIC, BASS_CONFIG_GVOL_SAMPLE,
BASS_CONFIG_GVOL_STREAM
The global volume settings now have a range of 0 to 10000.

BASS_RecordGetInput, BASS_RecordSetInput
The volume has been separated from the flags and made a floating-point value,
with a range of 0 to 1.
BASS_Update
A "length" parameter has been added.

BASS_ChannelPreBuf
This function has been superseded by BASS_ChannelUpdate. No parameter changes are required.

BASS_ChannelSetFlags
This function has been superseded by BASS_ChannelFlags. For the same functionality, set the "mask" parameter to -1.

BASS_ChannelSetSync
BASS_SYNC_META sync callbacks no longer receive the metadata in the "data" parameter. It can be retrieved via BASS_ChannelGetTags (BASS_TAG_META) instead. The BASS_SYNC_META sync is also no longer triggered by a new logical bitstream in a chained OGG stream; a dedicated BASS_SYNC_OGG_CHANGE sync has been added for that purpose. The BASS_SYNC_MESSAGE flag has been removed, and can be reproduced by using the PostMessage Win32 API function in a SYNCPROC.

BASS_ChannelBytes2Seconds
The return value is now 64-bit floating-point (double).

BASS_ChannelSeconds2Bytes
The "pos" parameter is now 64-bit floating-point (double).

BASS_SetConfig
This function now returns a boolean value, instead of the config's new setting. If needed, BASS_GetConfig can be used to get that.

BASS_CONFIG_NET_AGENT, BASS_CONFIG_NET_PROXY
Config options that deal with pointers, like these 2, are now handled by the new BASS_SetConfigPtr and BASS_GetConfigPtr functions.

BASS_CHANNELINFO
"sample" and "filename" members have been added.

DSPPROC, DOWNLOADPROC, RECORDPROC, STREAMPROC, SYNCPROC
All callback function "user" parameters are now pointers.
Upgrading from 2.2

In case you're skipping a version, here are the previous upgrade guidelines.

**BASS_GetVersion**
The format of the return value has been changed, to include the minor revision.

**BASS_StreamGetTags**
This function is replaced by **BASS_ChannelGetTags**. No parameter changes are required.

**BASS_MusicGetName**
This function is replaced by **BASS_ChannelGetTags** and the **BASS_TAG_MUSIC_NAME** tag type.

**BASS_PluginLoad**
A "flags" parameter has been added.

**BASS_CONFIG_NET_NOPROXY**
This config option has been replaced by the more flexible **BASS_CONFIG_NET_PROXY** config option. The **BASS_CONFIG_NET_NOPROXY** behaviour can be reproduced by setting the new option to NULL.

**BASS_CTYPE_STREAM_WAV**
This "ctype" is now just a flag to indicate any type of WAVE file, and is no longer a channel type in itself. The LOWORD used with this flag indicates the specific codec (PCM, ADPCM, etc).

**BASS_CHANNELINFO, BASS_INFO, BASS_RECORDINFO, BASS_SAMPLE**
These four structures have new "plugin", "freq", "freq" and "mingap" members, respectively.
Upgrading from 2.1

In case you're skipping another version, here are the previous upgrade guidelines.

**BASS_StreamGetLength, BASS_MusicGetLength**
These functions have been merged into `BASS_ChannelGetLength`, which gives the byte length of a channel. To get the number of orders in a MOD music, `BASS_MusicGetOrders` has been added. Also note that requesting the length when streaming in blocks will now result in a `BASS_ERROR_NOTAVAIL` error, instead of just 0.

**BASS_ChannelGetPosition**
This function now returns the position in bytes with MOD musics, as it does with other channels. `BASS_MusicGetOrderPosition` has been added to get the order/row position.

**BASS_ChannelSetPosition**
This function now sets the position in bytes with MOD musics by default. A `MAKEMUSICPOS` macro has been added for setting the position in orders/rows.

**BASS_ChannelSetSync**
The `BASS_SYNC_POS` sync parameter is now in bytes for MOD musics too. Use the `BASS_SYNC_MUSICPOS` sync for order/row positions.

**BASS_ChannelPreBuf**
A "length" parameter has been added, to specify how much data to pre-buffer.

**BASS_SampleCreate**
The number of channels is now specified in a separate parameter, rather than via the use of the `BASS_SAMPLE_MONO` flag. The sample length is also now specified in bytes rather than samples.

**BASS_StreamCreateFile**
WAV files are no longer automatically converted to the device initialization resolution (as specified in the `BASS_Init` call), eg. 8-bit WAV files will produce
8-bit streams.

**BASS_StreamCreateURL**
Shoutcast metadata is now requested automatically, so the BASS_STREAM_META flag has been removed.

**BASS_INFO structure, BASS_RECORDINFO structure**
The "size" member has been removed, so there's now no need to set that before calling **BASS_GetInfo** and **BASS_RecordGetInfo**.

**BASS_SAMPLE structure**
A "chans" member has been added.

**STREAMFILEPROC callback**
The BASS_FILE_QUERY action has been removed. The BASS_FILE_LEN action may be used more than once with unbuffered streams, not just at the start.
Upgrading from 2.0

In case you're skipping another version, here are the previous upgrade guidelines.

BASS_StreamPlay
This function is replaced by BASS_ChannelPlay. The "flags" parameter is not carried forward to the new function, instead the flags can be changed at any time using BASS_ChannelSetFlags.

BASS_MusicPlay, BASS_MusicPlayEx
These functions are also replaced by BASS_ChannelPlay. The BASS_MusicPlayEx "pos" and "flags" parameters are not carried forward to the new function, instead they can be implemented via the BASS_ChannelSetPosition and BASS_ChannelSetFlags functions. The "reset" parameter is also not carried forward, the BASS_MUSIC_POSRESETEX flag replaces that.

BASS_SamplePlay, BASS_SamplePlayEx, BASS_SamplePlay3D, BASS_SamplePlay3DEx
These functions are replaced by a combination of BASS_SampleGetChannel and BASS_ChannelPlay. The BASS_ChannelSetAttributes and BASS_ChannelSet3DAttributes functions can be used to replace the functionality of the "Ex" versions.

BASS_ChannelResume
Another function replaced by BASS_ChannelPlay.

These functions are all replaced by BASS_MusicSetAttribute and BASS_MusicGetAttribute.

BASS_StreamPreBuf, BASS_MusicPreBuf
These functions are replaced by BASS_ChannelPreBuf.
**BASS_RecordStart**
The number of channels (mono/stereo) is now specified in a separate parameter, rather than via the use of the BASS_SAMPLE_MONO flag.

**BASS_ChannelGetLevel**
The level reading is now more precise, having a range of 0 to 32768, instead of the old 0 to 128.

**BASS_ChannelSetFX**
There is a new "priority" parameter.

**BASS_CONFIG_FLOATDSP**
Note that this config option can now affect FX too.
BASS_GetConfig

Retrieves the value of a config option.

```c
DWORD BASS_GetConfig(
    DWORD option
);
```
### Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BASS_CONFIG_3DALGORITHM</code></td>
<td>The 3D algorithm for software mixed 3D channels.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_AIRPLAY</code></td>
<td>Enabled Airplay receivers.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_ASYNCFILE_BUFFER</code></td>
<td>Asynchronous file reading buffer length.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_BUFFER</code></td>
<td>Playback buffer length.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_CURVE_PAN</code></td>
<td>Panning translation curve.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_CURVE_VOL</code></td>
<td>Volume translation curve.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_DEV_BUFFER</code></td>
<td>Output device buffer length.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_DEV_DEFAULT</code></td>
<td>Include a &quot;Default&quot; entry in the output device list?</td>
</tr>
<tr>
<td><code>BASS_CONFIG_DEV_NONSTOP</code></td>
<td>Do not stop the output device when nothing is playing?</td>
</tr>
<tr>
<td><code>BASS_CONFIG_FLOATDSP</code></td>
<td>Pass 32-bit floating-point sample data to all DSP functions?</td>
</tr>
<tr>
<td><code>BASS_CONFIG_GVOL_MUSIC</code></td>
<td>Global MOD music volume.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_GVOL_SAMPLE</code></td>
<td>Global sample volume.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_GVOL_STREAM</code></td>
<td>Global stream volume.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_MUSIC_VIRTUAL</code></td>
<td>IT virtual channels.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_NET_BUFFER</code></td>
<td>Internet download buffer length.</td>
</tr>
<tr>
<td><code>BASS_CONFIG_NET_PASSIVE</code></td>
<td>Use passive mode in FTP connections?</td>
</tr>
<tr>
<td><code>BASS_CONFIG_NET_PLAYLIST</code></td>
<td>Process URLs in</td>
</tr>
</tbody>
</table>
PLAYLISTS?

BASS_CONFIG_NET_PREBUF
Amount to pre-buffer when opening internet streams.

BASS_CONFIG_NET_READTIMEOUT
Time to wait for a server to deliver more data.

BASS_CONFIG_NET_TIMEOUT
Time to wait for a server to respond to a connection request.

BASS_CONFIG_OGG_PRESCAN
Pre-scan chained OGG files?

BASS_CONFIG_PAUSE_NOPLAY
Prevent channels being played when the output is paused?

BASS_CONFIG_REC_BUFFER
Recording buffer length.

BASS_CONFIG_SRC
Default sample rate conversion quality.

BASS_CONFIG_SRC_SAMPLE
Default sample rate conversion quality for samples.

BASS_CONFIG_UNICODE
Unicode device information?

BASS_CONFIG_UPDATEPERIOD
Update period of playback buffers.

BASS_CONFIG_UPDATETHREADS
Number of update threads.

BASS_CONFIG_VERIFY
File format verification length.

BASS_CONFIG_VERIFY_NET
File format verification length for internet streams.

BASS_CONFIG_VISTA_SPEAKERS
Enable speaker assignment with panning/balance control on Windows Vista and playlists?
BASS_CONFIG_VISTA_TRUEPOS

Enable true play position mode on Windows Vista and newer?

other config options may be supported by add-ons, see the documentation.
Return value
If successful, the value of the requested config option is returned, else -1 is returned. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_ILLPARAM  *option* is invalid.
See also
BASS_GetConfigPtr, BASS_SetConfig
BASS_GetConfigPtr

Retrieves the value of a pointer config option.

```c
void *BASS_GetConfigPtr(
    DWORD option
);
```
Parameters

option  The option to set the value of... one of the following.

BASS_CONFIG_NET_AGENT  "User-Agent" header.
BASS_CONFIG_NET_PROXY  Proxy server settings.

*other config options may be supported by add-ons, see the documentation.*
Return value
If successful, the value of the requested config option is returned, else NULL is returned. NULL may also be a valid setting with some config options, in which case the error code should be used to confirm whether it's an error. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_ILLPARAM  option is invalid.
See also
BASS_GetConfig, BASS_SetConfigPtr
Sets the value of a config option.

```c
BOOL BASS_SetConfig(
    DWORD option,
    DWORD value
);
```
Parameters

option  The option to set the value of... one of the following.

BASS_CONFIG_3DALGORITHM  The 3D algorithm for software mixed 3D channels.
BASS_CONFIG_AIRPLAY  Enabled Airplay receivers.
BASS_CONFIG_ASYNCFILE_BUFFER  Asynchronous file reading buffer length.
BASS_CONFIG_BUFFER  Playback buffer length.
BASS_CONFIG_CURVE_PAN  Panning translation curve.
BASS_CONFIG_CURVE_VOL  Volume translation curve.
BASS_CONFIG_DEV_BUFFER  Output device buffer length.
BASS_CONFIG_DEV_DEFAULT  Include a "Default" entry in the output device list?
BASS_CONFIG_DEV_NONSTOP  Do not stop the output device when nothing is playing?
BASS_CONFIG_FLOATDSP  Pass 32-bit floating-point sample data to all DSP functions?
BASS_CONFIG_GVOL_MUSIC  Global MOD music volume.
BASS_CONFIG_GVOL_SAMPLE  Global sample volume.
BASS_CONFIG_GVOL_STREAM  Global stream volume.
BASS_CONFIG_MF_VIDEO  Play the audio from videos using Media Foundation?
BASS_CONFIG_MUSIC_VIRTUAL  IT virtual channels.
BASS_CONFIG_NET_BUFFER  Internet download buffer length.
**BASS_CONFIG_NET_PASSIVE**
- Use passive mode in FTP connections?

**BASS_CONFIG_NET_PLAYLIST**
- Process URLs in playlists?

**BASS_CONFIG_NET_PREBUF**
- Amount to pre-buffer when opening internet streams.

**BASS_CONFIG_NET_READTIMEOUT**
- Time to wait for a server to deliver more data.

**BASS_CONFIG_NET_TIMEOUT**
- Time to wait for a server to respond to a connection request.

**BASS_CONFIG_OGG_PRESCAN**
- Pre-scan chained OGG files?

**BASS_CONFIG_PAUSE_NOPLAY**
- Prevent channels being played when the output is paused?

**BASS_CONFIG_REC_BUFFER**
- Recording buffer length.

**BASS_CONFIG_SRC**
- Default sample rate conversion quality.

**BASS_CONFIG_SRC_SAMPLE**
- Default sample rate conversion quality for samples.

**BASS_CONFIG_UNICODE**
- Unicode device information?

**BASS_CONFIG_UPDATEPERIOD**
- Update period of playback buffers.

**BASS_CONFIG_UPDATETHREADS**
- Number of update threads.

**BASS_CONFIG_VERIFY**
- File format verification length.

**BASS_CONFIG_VERIFY_NET**
- File format verification length for internet streams.
<table>
<thead>
<tr>
<th><strong>BASS_CONFIG_VISTA_SPEAKERS</strong></th>
<th>Enable speaker assignment with panning/balance control on Windows Vista and newer?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BASS_CONFIG_VISTA_TRUEPOS</strong></td>
<td>Enable true play position mode on Windows Vista and newer?</td>
</tr>
</tbody>
</table>

*other config options may be supported by add-ons, see the documentation.*

**value** The new option setting. See the option's documentation for details on the possible values.
Return value
If successful, TRUE is returned, else FALSE is returned. Use
BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_ILLPARAM  option is invalid.
Remarks
Some config options have a restricted range of values, so the config's actual value may not be the same as requested if it was out of range. `BASS_GetConfig` can be used to confirm what the value is.

Config options can be used at any time and are independent of initialization, i.e. `BASS_Init` does not need to have been called beforehand.

Where a config option is shown to have a "BOOL" value, 0 (zero) is taken to be "FALSE" and anything else is taken to be "TRUE".
See also
BASS_GetConfig, BASS_SetConfigPtr
Sets the value of a pointer config option.

```c
BOOL BASS_SetConfigPtr(
    DWORD option,
    void *value
);
```
Parameters

option  The option to set the value of... one of the following.
   BASS_CONFIG_NET_AGENT  "User-Agent" header.
   BASS_CONFIG_NET_PROXY  Proxy server settings.
other config options may be supported by add-ons, see the documentation.

value  The new option setting. See the option's documentation for details on the possible values.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use
[BASS_ErrorGetCode](#) to get the error code.
Error codes
BASS_ERROR_I LLPARAM  option is invalid.
Remarks
Config options can be used at any time and are independent of initialization, ie. BASS_Init does not need to have been called beforehand.
See also
BASS_GetConfigPtr, BASS_SetConfig
The 3D algorithm for software mixed 3D channels.

BASS_SetConfig(
    BASS_CONFIG_3DALGORITHM,
    DWORD algo
);

BASS_CONFIG_3DALGORITHM config option
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo</td>
<td>One of these algorithms.</td>
</tr>
<tr>
<td><strong>BASS_3DALG_DEFAULT</strong></td>
<td>The default algorithm. If the user has selected a surround sound speaker configuration (eg. 4 or 5.1) in the control panel, the sound is panned among the available directional speakers. Otherwise it equates to BASS_3DALG_OFF.</td>
</tr>
<tr>
<td><strong>BASS_3DALG_OFF</strong></td>
<td>Uses normal left and right panning. The vertical axis is ignored except for scaling of volume due to distance. Doppler shift and volume scaling are still applied, but the 3D filtering is not performed. This is the most CPU efficient algorithm, but provides no virtual 3D audio effect. Head Related Transfer Function processing will not be done. Since only normal stereo panning is used, a channel using this algorithm may be accelerated by a 2D hardware voice if no free 3D hardware voices are available.</td>
</tr>
<tr>
<td><strong>BASS_3DALG_FULL</strong></td>
<td>This algorithm gives the highest quality 3D audio effect, but uses more CPU. This algorithm requires WDM drivers, if it's not available then BASS_3DALG_OFF will automatically be used instead.</td>
</tr>
<tr>
<td><strong>BASS_3DALG_LIGHT</strong></td>
<td>This algorithm gives a good 3D audio effect, and uses less CPU than the FULL algorithm. This algorithm also requires WDM drivers, if it's not available then BASS_3DALG_OFF will automatically be used instead.</td>
</tr>
</tbody>
</table>
**Remarks**
These algorithms only affect 3D channels that are being mixed in software. [BASS_ChannelGetInfo](#) can be used to check whether a channel is being software mixed.

Changing the algorithm only affects subsequently created or loaded samples, musics, or streams; it does not affect any that already exist.
**Platform-specific**
On Windows, DirectX 7 or above is required for this option to have effect. On other platforms, only the BASS_3DALG_DEFAULT and BASS_3DALG_OFF options are available.
See also
BASS_GetConfig, BASS_SampleCreate, BASS_SampleLoad, BASS_SetConfig
BASS_CONFIG_AIRPLAY config option

Enabled Airplay receivers.

BASS_SetConfig(
    BASS_CONFIG_AIRPLAY,
    DWORD receivers
);
**Parameters**

**receivers**  
Enabled receivers... the 1st bit is the 1st receiver, the 2nd bit is the 2nd receiver, etc. If a bit is set, then the corresponding receiver is enabled.
Remarks
This config option determines which Airplay receivers will receive the sound when the Airplay output device is used. The receiver configuration is a global setting, so changes will also affect any other software that uses the Airplay device.

BASS_GetDeviceInfo can be used to enumerate the available Airplay receivers. Unlike the output device list, where entries are never removed (the BASS_DEVICE_ENABLED flag is just unset), entries may be removed from the Airplay receiver list as it only contains receivers that are currently available. That means you should not depend on the bit indexes remaining constant.
**Platform-specific**
This config option is only available on OSX.
**Example**
Enable the 1st and 2nd Airplay receivers.

```c
BASS_SetConfig(BASS_CONFIG_AIRPLAY, 3);
```
See also
BASS_GetConfig, BASS_GetDeviceInfo, BASS_SetConfig
BASS_CONFIG_ASYNCFILE_BUFFER
config option

The buffer length for asynchronous file reading.

BASS_SetConfig(
    BASS_CONFIG_ASYNCFILE_BUFFER,
    DWORD length
);

**Parameters**

length  The buffer length in bytes. This will be rounded up to the nearest 4096 byte (4KB) boundary.
Remarks
This determines the amount of file data that can be read ahead of time with asynchronous file reading. The default setting is 65536 bytes (64KB). Changes only affect streams that are created afterwards, not any that already exist. So it is possible to have streams with differing buffer lengths by using this config option before creating each of them.

When asynchronous file reading is enabled, the buffer level is available from BASS_StreamGetFilePosition.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFile, BASS_StreamGetFilePosition
**BASS_CONFIG_BUFFER config option**

The playback buffer length for HSTREAM and HMUSIC channels.

```c
BASS_SetConfig(  
    BASS_CONFIG_BUFFER,  
    DWORD length  
);
```
**Parameters**

*length*  The buffer length in milliseconds. The minimum length is 1ms above the update period (*BASS_CONFIG_UPDATEPERIOD*), the maximum is 5000 milliseconds. If the length specified is outside this range, it is automatically capped.
Remarks
The default buffer length is 500 milliseconds. Increasing the length, decreases the chance of the sound possibly breaking-up on slower computers, but also increases the latency for DSP/FX.

Small buffer lengths are only required if the sound is going to be changing in real-time, for example, in a soft-synth. If you need to use a small buffer, then the minbuf member of BASS_INFO should be used to get the recommended minimum buffer length supported by the device and its drivers. Even at this default length, it's still possible that the sound could break up on some systems, it's also possible that smaller buffers may be fine. So when using small buffers, you should have an option in your software for the user to finetune the length used, for optimal performance.

Using this config option only affects the HMUSIC/HSTREAM channels that are created afterwards, not any that have already been created. So you can have channels with differing buffer lengths by using this config option each time before creating them.

If automatic updating is disabled, make sure you call BASS_Update frequently enough to keep the buffers updated.
**Example**

Use the recommended minimum buffer length, added to the update period.

```c
DWORD len=BASS_GetConfig(BASS_CONFIG_UPDATEPERIOD); // get update period
BASS_INFO info;
BASS_GetInfo(&info); // retrieve device info
len+=info.minbuf; // add the 'minbuf'
BASS_SetConfig(BASS_CONFIG_BUFFER, len); // set the buffer length
```
See also
BASS_GetConfig, BASS_GetInfo, BASS_SetConfig,
BASS_ATTRIB_NOBUFFER, BASS_CONFIG_UPDATEPERIOD
The translation curve of volume values.

```c
BASS_SetConfig(  
    BASS_CONFIG_CURVE_VOL,  
    BOOL logvol  
);
```
Parameters
logvol  Volume curve... FALSE = linear, TRUE = logarithmic.
Remarks
When using the linear curve, the volume range is from 0% (silent) to 100% (full). When using the logarithmic curve, the volume range is from -100 dB (effectively silent) to 0 dB (full). For example, a volume level of 0.5 is 50% linear or -50 dB logarithmic.

The linear curve is used by default.
See also
BASS_GetConfig, BASS_SetConfig, BASS_SetVolume, BASS_ATTRIB_VOL, BASS_CONFIG_CURVE_PAN
BASS_CONFIG_CURVE_PAN config option

The translation curve of panning values.

```c
BASS_SetConfig(
    BASS_CONFIG_CURVE_PAN,
    BOOL logpan
);
```
Parameters
logpan  Panning curve... FALSE = linear, TRUE = logarithmic.
Remarks
The panning curve affects panning in exactly the same way as the volume curve (BASS_CONFIG_CURVE_VOL) affects the volume.

The linear curve is used by default.
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_PAN,
BASS_CONFIG_CURVE_VOL
**BASS_CONFIG_DEV_BUFFER config option**

The output device buffer length.

```c
BASS_SetConfig(
    BASS_CONFIG_DEV_BUFFER,
    DWORD length
);
```
**Parameters**

length  The buffer length in milliseconds.
Remarks
The device buffer is where the final mix of all playing channels is placed, ready for the device to play. Its length affects the latency of things like starting and stopping playback of a channel, so you will probably want to avoid setting it unnecessarily high, but setting it too short could result in breaks in the output.

When using a large device buffer, the BASS_ATTRIB_NOBUFFER attribute could be used to skip the channel buffering stage, to avoid further increasing latency for real-time generated sound and/or DSP/FX changes.

Changes to this config setting only affect subsequently initialized devices, not any that are already initialized.
**Platform-specific**
This config option is available on Linux, Android, and Windows CE. The device's buffer is determined automatically on other platforms.

On Linux, BASS will attempt to set the device buffer-feeding thread to real-time priority (as on other platforms) to reduce the chances of it getting starved of CPU, but if that is not possible (e.g., the user account lacks permission) then it may be necessary to increase the buffer length to avoid breaks in the output when the CPU is busy. The driver may also choose to use a different buffer length if it decides that the specified length is too short or long. The buffer length actually being used can be obtained with `BASS_INFO`, like this: `latency + minbuf / 2`. 
See also
BASS_GetConfig, BASS_GetInfo, BASS_Init, BASS_SetConfig,
BASS_ATTRIB_NOBUFFER
BASS_CONFIG_DEV_DEFAULT config option

Include a "Default" entry in the output device list?

BASS_SetConfig(
    BASS_CONFIG_DEV_DEFAULT,
    BOOL default
);
Parameters

default  If TRUE, a "Default" device will be included in the device list.
Remarks
BASS does not usually include a "Default" entry in its device list, as that would ultimately map to one of the other devices and be a duplicate entry. When the default device is requested in a BASS_Init call (with device = -1), BASS will check the default device at that time, and initialize it. But Windows 7 has the ability to automatically switch the default output to the new default device whenever it changes, and in order for that to happen, the default device (rather than a specific device) needs to be used. That is where this option comes in.

When enabled, the "Default" device will also become the default device to BASS_Init (with device = -1). When the "Default" device is used, the BASS_SetVolume and BASS_GetVolume functions work a bit differently to usual; they deal with the "session" volume, which only affects the current process's output on the device, rather than the device's volume.

This option can only be set before BASS_GetDeviceInfo or BASS_Init has been called.
**Platform-specific**
This config option is only available on Windows. It is available on all Windows versions (not including CE), but only Windows 7 has the default output switching feature.
See also
BASS_GetConfig, BASS_SetConfig, BASS_Init
BASS_CONFIG_FLOATDSP config option

Pass 32-bit floating-point sample data to all DSP functions?

```c
BASS_SetConfig(
    BASS_CONFIG_FLOATDSP,
    BOOL floatdsp
);
```
**Parameters**

floatdsp  If TRUE, 32-bit floating-point sample data is passed to all **DSPPROC** callback functions.
Remarks
Normally DSP functions receive sample data in whatever format the channel is using, ie. it can be 8, 16 or 32-bit. But using this config option, BASS will convert 8/16-bit sample data to 32-bit floating-point before passing it to DSP functions, and then convert it back after all the DSP functions are done. As well as simplifying the DSP code (no need for 8/16-bit processing), this also means that there is no degradation of quality as sample data passes through a chain of DSP.

This config option also applies to effects set via BASS_ChannelSetFX, except for DX8 effects when using the "With FX flag" DX8 effect implementation.

Changing the setting while there are DSP or FX set could cause problems, so should be avoided.
Platform-specific
On Android and Windows CE, 8.24 fixed-point is used instead of floating-point.
Floating-point DX8 effect processing requires DirectX 9 (or above) on Windows.
See also
BASS_GetConfig, BASS_SetConfig, DSPPROC callback
BASS_CONFIG_GVOL_MUSIC config option

The global MOD music volume level.

BASS_SetConfig(  
    BASS_CONFIG_GVOL_MUSIC,  
    DWORD volume  
);
**Parameters**

volume MOD music global volume level... 0 (silent) to 10000 (full).
Remarks
This config option allows you to have control over the volume levels of all the MOD musics, which is useful for setup options, eg. separate music and fx volume controls.

A channel's final volume = channel volume x global volume / 10000. For example, if a stream's volume is 0.5 and the global stream volume is 8000, then effectively the stream's volume level is 0.4 (0.5 x 8000 / 10000 = 0.4).
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_VOL,
BASS_CONFIG_GVOL_SAMPLE, BASS_CONFIG_GVOL_STREAM
BASS_CONFIG_GVOL_SAMPLE config option

The global sample volume level.

BASS_SetConfig(  
    BASS_CONFIG_GVOL_SAMPLE,  
    DWORD volume  
);
**Parameters**

volume  Sample global volume level... 0 (silent) to 10000 (full).
Remarks
This config option allows you to have control over the volume levels of all the samples, which is useful for setup options, eg. separate music and fx volume controls.

A channel's final volume = \( \text{channel volume} \times \text{global volume} / 10000 \). For example, if a stream's volume is 0.5 and the global stream volume is 8000, then effectively the stream's volume level is 0.4 (0.5 \times 8000 / 10000 = 0.4).
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_VOL,
BASS_CONFIG_GVOL_MUSIC, BASS_CONFIG_GVOL_STREAM
BASS_CONFIG_GVOL_STREAM config option

The global stream volume level.

```c
BASS_SetConfig(
    BASS_CONFIG_GVOL_STREAM,
    DWORD volume
);
```
Parameters

volume  Stream global volume level... 0 (silent) to 10000 (full).
Remarks
This config option allows you to have control over the volume levels of all the streams, which is useful for setup options, eg. separate music and fx volume controls.

A channel's final volume = \textit{channel volume} \times \textit{global volume} / 10000. For example, if a stream's volume is 0.5 and the global stream volume is 8000, then effectively the stream's volume level is 0.4 (0.5 \times 8000 / 10000 = 0.4).
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_VOL,
BASS_CONFIG_GVOL_MUSIC, BASS_CONFIG_GVOL_SAMPLE
BASS_CONFIG_MF_VIDEO config option

Play the audio from video files using Media Foundation?

```c
BASS_SetConfig(
    BASS_CONFIG_MF_VIDEO,
    BOOL video
);
```
**Parameters**

video  Accept video files?
Remarks
This option is enabled by default.
**Platform-specific**
This config option is only available on Windows, and only has effect on Windows Vista and newer.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFile,
BASS_StreamCreateFileUser, BASS_StreamCreateURL
BASS_CONFIG_MUSIC_VIRTUAL config option

The maximum number of virtual channels to use in the rendering of IT files.

```c
BASS_SetConfig(
    BASS_CONFIG_MUSIC_VIRTUAL,
    DWORD chans
);
```
**Parameters**

chans  The number of virtual channels... 1 (min) to 512 (max). If the value specified is outside this range, it is automatically capped.
Remarks
This setting only affects IT files, as the other MOD music formats do not have virtual channels. The default setting is 64. Changes only apply to subsequently loaded files, not any that are already loaded.
See also
BASS_GetConfig, BASS_MusicLoad, BASS_SetConfig,
BASS_ATTRIB_MUSIC_ACTIVE
The "User-Agent" request header sent to servers.

```c
BASS_SetConfigPtr(
    BASS_CONFIG_NET_AGENT,
    char *agent
);
```
Parameters
agent  The "User-Agent" header.
**Remarks**
BASS does not make a copy of the config string, so it must reside in the heap (not the stack), eg. a global variable. This also means that the agent setting can subsequently be changed at that location without having to call this function again.

Changes take effect from the next internet stream creation call.
Platform-specific
On Windows CE, the string is in UTF-16 form ("WCHAR" rather than "char").
See also
BASS_GetConfigPtr, BASS_SetConfigPtr, BASS_StreamCreateURL
BASS_CONFIG_NET_BUFFER config option

The internet download buffer length.

BASS_SetConfig(
    BASS_CONFIG_NET_BUFFER,
    DWORD length
);
Parameters

length    The buffer length in milliseconds.
Remarks
Increasing the buffer length decreases the chance of the stream stalling, but also increases the time taken to create the stream as more data has to be pre-buffered (adjustable via the BASS_CONFIG_NET_PREBUF config option). Aside from the pre-buffering, this setting has no effect on streams without either the BASS_STREAM_BLOCK or BASS_STREAM_RESTRATE flags.

When streaming in blocks, this option determines the download buffer length. The effective buffer length can actually be a bit more than that specified, including data that has been read from the buffer by the decoder but not yet decoded.

This config option also determines the buffering used by "buffered" user file streams created with BASS_StreamCreateFileUser.

The default buffer length is 5 seconds (5000 milliseconds). The net buffer length should be larger than the length of the playback buffer (BASS_CONFIG_BUFFER), otherwise the stream is likely to stall soon after starting playback.

Using this config option only affects streams created afterwards, not any that have already been created.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFileUser,
BASS_StreamCreateURL, BASS_CONFIG_BUFFER,
BASS_CONFIG_NET_PREBUF, BASS_CONFIG_NET_TIMEOUT
BASS_CONFIG_NET_PASSIVE config option

Use passive mode in FTP connections?

BASS_SetConfig(
    BASS_CONFIG_NET_PASSIVE,
    BOOL passive
);
**Parameters**

passive  If TRUE, passive mode is used, otherwise normal/active mode is used.
Remarks
Changes take effect from the next internet stream creation call. By default, passive mode is enabled.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateURL
Process URLs in PLS and M3U playlists?

BASS_SetConfig(
    BASS_CONFIG_NET_PLAYLIST,
    DWORD netlists
);
**Parameters**

**netlists**  When to process URLs in PLS and M3U playlists... 0 = never, 1 = in `BASS_StreamCreateURL` only, 2 = in `BASS_StreamCreateFile` and `BASS_StreamCreateFileUser` too.
Remarks
When enabled, BASS will process PLS and M3U playlists, trying each URL until it finds one that it can play. BASS_ChannelGetInfo can be used to find out the URL that was successfully opened.

Nested playlists are supported, that is a playlist can contain the URL of another playlist.

By default, playlist processing is disabled.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateURL
BASS_CONFIG_NET_PREBUF config option

Amount to pre-buffer when opening internet streams.

BASS_SetConfig(
    BASS_CONFIG_NET_PREBUF,
    DWORD prebuf
);
**Parameters**

prebuf  Amount (percentage) to pre-buffer.
Remarks
This setting determines what percentage of the buffer length (BASS_CONFIG_NET_BUFFER) should be filled by BASS_StreamCreateURL. The default is 75%. Setting this lower (eg. 0) is useful if you want to display a "buffering progress" (using BASS_StreamGetPosition) when opening internet streams, but note that this setting is just a minimum; BASS will always pre-download a certain amount to verify the stream.

As well as internet streams, this config setting also applies to "buffered" user file streams created with BASS_StreamCreateFileUser.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateURL,
BASS_CONFIG_NET_BUFFER
BASS_CONFIG_NET_PROXY config option

Proxy server settings.

BASS_SetConfigPtr(
    BASS_CONFIG_NET_PROXY,
    char *proxy
);

Parameters

proxy  The proxy server settings, in the form of "user:pass@server:port"... NULL = don't use a proxy. "" (empty string) = use the OS's default proxy settings. If only the "user:pass@" part is specified, then those authorization credentials are used with the default proxy server. If only the "server:port" part is specified, then that proxy server is used without any authorization credentials.
Remarks
BASS does not make a copy of the config string, so it must reside in the heap (not the stack), eg. a global variable. This also means that the proxy settings can subsequently be changed at that location without having to call this function again.

Changes take effect from the next internet stream creation call.
**Platform-specific**
On Windows CE, the string is in UTF-16 form ("WCHAR" rather than "char").
See also
BASS_GetConfigPtr, BASS_SetConfigPtr, BASS_StreamCreateURL
BASS_CONFIG_NET_READTIMEOUT
config option

The time to wait for a server to deliver more data for an internet stream.

BASS_SetConfig(
    BASS_CONFIG_NET_READTIMEOUT,
    DWORD timeout
);

Parameters

timeout  The time to wait, in milliseconds... 0 = no timeout.
**Remarks**
When the timeout is hit, the connection with the server will be closed.

The default setting is 0, no timeout.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateURL, BASS_CONFIG_NET_TIMEOUT
**BASS_CONFIG_NET_TIMEOUT** config option

The time to wait for a server to respond to a connection request.

```c
BASS_SetConfig(
    BASS_CONFIG_NET_TIMEOUT,
    DWORD timeout
);
```
Parameters

timeout  The time to wait, in milliseconds.
Remarks
The default timeout is 5 seconds (5000 milliseconds).
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateURL,
BASS_CONFIG_NET_BUFFER, BASS_CONFIG_NET_READTIMEOUT
Pre-scan chained OGG files?

```c
BASS_SetConfig(
    BASS_CONFIG_OGG_PRESCAN,
    BOOL prescan
);
```
Parameters

prescan  If TRUE, chained OGG files are pre-scanned.
 Remarks
This option is enabled by default, and is equivalent to including the BASS_STREAM_PRESCAN flag in a BASS_StreamCreateFile call when opening an OGG file. It can be disabled if seeking and an accurate length reading are not required from chained OGG files, for faster stream creation.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFile
BASS_CONFIG_PAUSE_NOPLAY config option

Prevent channels being played while the output is paused?

```c
BASS_SetConfig(    BASS_CONFIG_PAUSE_NOPLAY,
    BOOL noplay );
```
**Parameters**

**noplay**  If TRUE, channels cannot be played while the output is paused.
Remarks
When the output is paused using `BASS_Pause`, and this config option is enabled, channels cannot be played until the output is resumed using `BASS_Start`. Any attempts to play a channel will result in a `BASS_ERROR_START` error.

By default, this config option is enabled.
See also
BASS_GetConfig, BASS_SetConfig, BASS_Pause
The buffer length for recording channels.

BASS_SetConfig(
    BASS_CONFIG_REC_BUFFER,
    DWORD length
);
**Parameters**

length  The buffer length in milliseconds... 1000 (min) - 5000 (max). If the length specified is outside this range, it is automatically capped.
Remarks
Unlike a playback buffer, where the aim is to keep the buffer full, a recording buffer is kept as empty as possible and so this setting has no effect on latency. The default recording buffer length is 2000 milliseconds. Unless processing of the recorded data could cause significant delays, or you want to use a large recording period with BASS_RecordStart, there should be no need to increase this.

Using this config option only affects the recording channels that are created afterwards, not any that have already been created. So it is possible to have channels with differing buffer lengths by using this config option each time before creating them.
See also
BASS_GetConfig, BASS_RecordStart, BASS_SetConfig
The default sample rate conversion quality.

```c
BASS_SetConfig(
    BASS_CONFIG_SRC,
    DWORD quality
);
```
**Parameters**

quality  The sample rate conversion quality... 0 = linear interpolation, 1 = 8 point sinc interpolation, 2 = 16 point sinc interpolation, 3 = 32 point sinc interpolation. Other values are also accepted.
Remarks
This config option determines what sample rate conversion quality new channels will initially have, except for sample channels (HCHANNEL), which use the BASS_CONFIG_SRC_SAMPLE setting. A channel's sample rate conversion quality can subsequently be changed via the BASS_ATTRIB_SRC attribute.

The default setting is 1 (8 point sinc interpolation).
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_SRC, BASS_CONFIG_SRC_SAMPLE
BASS_CONFIG_SRC_SAMPLE config option

The default sample rate conversion quality for samples.

BASS_SetConfig(
    BASS_CONFIG_SRC_SAMPLE,
    DWORD quality
);
### Parameters

**quality**  The sample rate conversion quality... 0 = linear interpolation, 1 = 8 point sinc interpolation, 2 = 16 point sinc interpolation, 3 = 32 point sinc interpolation. Other values are also accepted.
Remarks
This config option determines what sample rate conversion quality a new sample channel will initially have, following a BASS_SampleGetChannel call. The channel's sample rate conversion quality can subsequently be changed via the BASS_ATTRIB_SRC attribute.

The default setting is 0 (linear interpolation).
Platform-specific
This option is not available on Windows.
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_SRC, BASS_CONFIG_SRC
BASS_CONFIG_UNICODE config option

Use the Unicode character set in device information?

BASS_SetConfig(
    BASS_CONFIG_UNICODE,
    BOOL unicode
);
**Parameters**

unicode  If TRUE, device information will be in UTF-8 form. Otherwise it will be ANSI.
Remarks
This config option determines what character set is used in the `BASS_DEVICEINFO` structure and by the `BASS_RecordGetInputName` function. The default setting is ANSI, and it can only be changed before `BASS_GetDeviceInfo` or `BASS_Init` or `BASS_RecordGetDeviceInfo` or `BASS_RecordInit` has been called.
**Platform-specific**
This config option is only available on Windows.
See also
BASS_GetConfig, BASS_SetConfig, BASS_DEVICEINFO structure
The update period of HSTREAM and HMUSIC channel playback buffers.

BASS_SetConfig(
    BASS_CONFIG_UPDATEPERIOD,
    DWORD period
);
**Parameters**

**period**  The update period in milliseconds... 0 = disable automatic updating. The minimum period is 5ms, the maximum is 100ms. If the period specified is outside this range, it is automatically capped.
Remarks
The update period is the amount of time between updates of the playback buffers of HSTREAM/HMUSIC channels. Shorter update periods allow smaller buffers to be set with the BASS_CONFIG_BUFFER config option, but as the rate of updates increases, so the overhead of setting up the updates becomes a greater part of the CPU usage. The update period only affects HSTREAM and HMUSIC channels; it does not affect samples. Nor does it have any effect on decoding channels, as they are not played.

BASS creates one or more threads (determined by BASS_CONFIG_UPDATETHREADS) specifically to perform the updating, except when automatic updating is disabled (period = 0), in which case BASS_Update or BASS_ChannelUpdate should be used instead. This allows BASS's CPU usage to be synchronized with your software's. For example, in a game loop you could call BASS_Update once per frame, to keep all the processing in sync so that the frame rate is as smooth as possible.

The update period can be altered at any time, including during playback. The default period is 100ms.
See also
BASS_ChannelUpdate, BASS_GetConfig, BASS_SetConfig, BASS_Update,
BASS_CONFIG_BUFFER, BASS_CONFIG_UPDATETHREADS,
BASS_ATTRIB_NOBUFFER
**BASS_CONFIG_UPDATETHREADS config option**

The number of threads to use for updating playback buffers.

```c
BASS_SetConfig(
    BASS_CONFIG_UPDATETHREADS,
    DWORD threads
);
```
Parameters

threads  The number of threads to use... 0 = disable automatic updating.
Remarks

The number of update threads determines how many HSTREAM/HMUSIC channel playback buffers can be updated in parallel; each thread can process one channel at a time. The default is to use a single thread, but additional threads can be used to take advantage of multiple CPU cores. There is generally nothing much to be gained by creating more threads than there are CPU cores, but one benefit of using multiple threads even with a single CPU core is that a slowly updating channel need not delay the updating of other channels.

When automatic updating is disabled (threads = 0), BASS_Update or BASS_ChannelUpdate should be used instead.

The number of update threads can be changed at any time, including during playback.
Platform-specific
The number of update threads is limited to 1 on the Windows CE platform.
See also
BASS_ChannelUpdate, BASS_GetConfig, BASS_SetConfig, BASS_Update,
BASS_CONFIG_BUFFER, BASS_CONFIG_UPDATEPERIOD
BASS_CONFIG_VERIFY config option

The amount of data to check in order to verify/detect the file format.

```c
BASS_SetConfig(
    BASS_CONFIG_VERIFY,
    DWORD length
);
```
Parameters

length  The amount of data to check, in bytes... 1000 (min) to 1000000 (max).
        If the value specified is outside this range, it is automatically capped.
Remarks
Of the file formats supported as standard, this setting only affects the detection of MP3/MP2/MP1 formats, but it may also be used by add-ons (see the documentation). The verification length excludes any tags that may be found at the start of the file. The default length is 16000 bytes.

For internet (and "buffered" user file) streams, the BASS_CONFIG_VERIFY_NET setting determines how much data is checked.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFile,
BASS_StreamCreateFileUser, BASS_CONFIG_VERIFY_NET
BASS_CONFIG_VERIFY_NET config option

The amount of data to check in order to verify/detect the file format of internet streams.

BASS_SetConfig(
    BASS_CONFIG_VERIFY_NET,
    DWORD length
);
Parameters
length The amount of data to check, in bytes... 1000 (min) to 1000000 (max), or 0 = 25% of the BASS_CONFIG_VERIFY setting (with a minimum of 1000 bytes). If the value specified is outside this range, it is automatically capped.
Remarks
Of the file formats supported as standard, this setting only affects the detection of MP3/MP2/MP1 formats, but it may also be used by add-ons (see the documentation). The verification length excludes any tags that may be found at the start of the file. The default setting is 0, which means 25% of the \texttt{BASS\_CONFIG\_VERIFY} setting.

As well as internet streams, this config setting also applies to "buffered" user file streams created with \texttt{BASS\_StreamCreateFileUser}.
See also
BASS_GetConfig, BASS_SetConfig, BASS_StreamCreateFileUser,
BASS_StreamCreateURL, BASS_CONFIG_VERIFY
Enable speaker assignment with panning/balance control on Windows Vista and newer?

```c
BASS_SetConfig(
    BASS_CONFIG_VISTA_SPEAKERS,
    BOOL enable
);
```
**Parameters**

**enable**  If TRUE, speaker assignment with panning/balance control is enabled on Windows Vista and newer.
Remarks
Panning/balance control via the BASS_ATTRIB_PAN attribute is not available when speaker assignment is used on Windows due to the way that the speaker assignment needs to be implemented there. The situation is improved with Windows Vista, and speaker assignment can generally be done in a way that does permit panning/balance control to be used at the same time, but there may still be some drivers that it does not work properly with, so it is disabled by default and can be enabled via this config option. Changes only affect channels that are created afterwards, not any that already exist.
**Platform-specific**
This config option is only available on Windows. It is available on all Windows versions (not including CE), but only has effect on Windows Vista and newer. Speaker assignment with panning/balance control is always possible on other platforms, where BASS generates the final mix.
See also
BASS_GetConfig, BASS_SetConfig, BASS_ATTRIB_PAN
BASS_CONFIG_VISTA_TRUEPOS config option

Enable true play position mode on Windows Vista and newer?

```c
BASS_SetConfig(
    BASS_CONFIG_VISTA_TRUEPOS,
    BOOL truepos
);
```
Parameters

truepos If TRUE, DirectSound's "true play position" mode is enabled on Windows Vista and newer.
Remarks
Unless this option is enabled, the reported playback position will advance in 10ms steps on Windows Vista and newer. As well as affecting the precision of BASS_ChannelGetPosition, this also affects the timing of non-mixtime syncs. When this option is enabled, it allows finer position reporting but it also increases latency.

The default setting is enabled. Changes only affect channels that are created afterwards, not any that already exist. The latency and minbuf values in the BASS_INFO structure reflect the setting at the time of the device's BASS_Init call.
Platform-specific
This config option is only available on Windows. It is available on all Windows versions (not including CE), but only has effect on Windows Vista and newer.
See also
BASS_ChannelGetPosition, BASS_GetConfig, BASS_GetInfo,
BASS_SetConfig
BASS_PluginFree

Unplugs an add-on.

```c
BOOL BASS_PluginFree(
    HPLUGIN handle
);
```
Parameters

handle  The plugin handle... 0 = all plugins.
**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 valid.
Remarks
If there are streams created by a plugin in existence when it is being freed, the streams will automatically be freed too. Samples loaded by the plugin are unaffected as the plugin has nothing to do with them once they are loaded; the sample data is already fully decoded.
See also
BASS_PluginLoad
BASS_PluginGetInfo

Retrieves information on a plugin.

```c
BASS_PLUGININFO *BASS_PluginGetInfo(
    HPLUGIN handle
);
```
**Parameters**

handle  The plugin handle.
Return value
If successful, a pointer to the plugin info is returned, else NULL is returned. Use BASS_ErrorGetCode to get the error code.
Remarks
The plugin information does not change, so the returned pointer remains valid for as long as the plugin is loaded.
Error codes
BASS_ERROR_HANDLE  handle is not valid.
Example
List the formats supported by a plugin.

```c
BASS_PLUGININFO *info=BASS_PluginGetInfo(plugin); // get the plugin info
int a;
for (a=0; a<info->formatc; a++) { // display the array of formats...
    printf("ctype=%x name=%s exts=%s\n",
           info->formats[a].ctype, info->formats[a].name, info->formats[a].exts);
}
```
See also
BASS_PluginLoad, BASS_PLUGININFO structure
BASS_PluginLoad

Plugs an "add-on" into the standard stream and sample creation functions.

HPLUGIN BASS_PluginLoad(
    char *file,
    DWORD flags
);

**Parameters**

file     Filename of the add-on/plugin.
flags    A combination of these flags.
        BASS_UNICODE  file is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, and UTF-8 on other platforms.
**Return value**
If successful, the loaded plugin's handle is returned, else 0 is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes

BASS_ERROR_FILEOPEN  The file could not be opened.
BASS_ERROR_FILEFORM  The file is not a plugin.
BASS_ERROR_VERSION   The plugin requires a different BASS version. Due to the use of the "stdcall" calling-convention, and so risk of stack faults from unexpected API differences, an add-on won't load at all on Windows if the BASS version is unsupported, and a BASS_ERROR_FILEFORM error will be generated instead of this.
BASS_ERROR_ALREADY   The plugin is already loaded.
Remarks
There are 2 ways in which add-ons can provide support for additional formats. They can provide dedicated functions to create streams of the specific format(s) they support and/or they can plug into the standard stream creation functions: BASS_StreamCreateFile, BASS_StreamCreateURL, and BASS_StreamCreateFileUser. This function enables the latter method. Both methods can be used side by side. The obvious advantage of the plugin system is convenience, while the dedicated functions can provide extra options that are not possible via the shared function interfaces. See an add-on's documentation for more specific details on it.

As well as the stream creation functions, plugins also add their additional format support to BASS_SampleLoad.

Information on what file formats a plugin supports is available via the BASS_PluginGetInfo function.

When using multiple plugins, the stream/sample creation functions will try each of them in the order that they were loaded via this function, until one that accepts the file is found.

When an add-on is already loaded (eg. if you are using functions from it), the plugin system will use the same instance (the reference count will just be incremented); there will not be 2 copies of the add-on in memory.
Platform-specific
Dynamic libraries are not permitted on iOS, so add-ons are provided as static libraries instead, which means this function has to work a little differently. The add-on needs to be linked into the executable, and a "plugin" symbol declared and passed to this function (instead of a filename). See the example below.
Example
Plugin the FLAC add-on.

```c
#ifdef __WIN32  // Windows/CE
    BASS_PluginLoad("bassflac.dll", 0);
#else __linux__  // Linux
    BASS_PluginLoad("libbassflac.so", 0);
#elif TARGET_OS_IPHONE  // iOS
    extern void BASSFLACplugin;
    BASS_PluginLoad(&BASSFLACplugin, 0);
#else  // OSX
    BASS_PluginLoad("libbassflac.dylib", 0);
#endif
```
See also

BASS_PluginFree, BASS_PluginGetInfo
BASS_PLUGINFORM structure

Information on a plugin supported format.

typedef struct {
    DWORD ctype;
    char *name;
    char *exts;
} BASS_PLUGINFORM;
**Members**

cctype  The channel type, as would appear in the `BASS_CHANNELINFO` structure.

name    Format description.

exts    File extension filter, in the form of "*.ext1;*.ext2;...".
**Remarks**
The extension filter is for information only. A plugin will check the file contents rather than file extension, to verify that it is a supported format.
Platform-specific
On Windows CE, *name* and *exts* are in UTF-16 form ("WCHAR" rather than "char").
See also
BASS_PluginGetInfo, BASS_PLUGININFO structure
BASS_PLUGININFO structure

Used with BASS_PluginGetInfo to retrieve information on a plugin.

typedef struct {
    DWORD version;
    DWORD formatc;
    BASS_PLUGINFORM *formats;
} BASS_PLUGININFO;
**Members**

- **version**: Plugin version, in the same form as given by `BASS_GetVersion`.
- **formatc**: Number of supported formats.
- **formats**: The array of supported formats. The array contains `formatc` elements.
See also
BASS_PluginGetInfo, BASS_PLUGINFORM structure
BASS_ErrorGetCode

Retrieves the error code for the most recent BASS function call in the current thread.

```c
int BASS_ErrorGetCode();
```
**Return value**

If no error occurred during the last BASS function call then BASS_OK is returned, else one of the BASS_ERROR values is returned. See the function description for an explanation of what the error code means.

Error codes are stored for each thread. So if you happen to call 2 or more BASS functions at the same time, they will not interfere with each other's error codes.
**Error codes list**

0  BASS_OK
1  BASS_ERROR_MEM
2  BASS_ERROR_FILEOPEN
3  BASS_ERROR_DRIVER
4  BASS_ERROR_BUFLOST
5  BASS_ERROR_HANDLE
6  BASS_ERROR_FORMAT
7  BASS_ERROR_POSITION
8  BASS_ERROR_INIT
9  BASS_ERROR_START
10  BASS_ERROR_SSL
14  BASS_ERROR_ALREADY
18  BASS_ERROR_NOCHAN
19  BASS_ERROR_ILLTYPE
20  BASS_ERROR_ILLPARAM
21  BASS_ERROR_NO3D
22  BASS_ERROR_NOEAX
23  BASS_ERROR_DEVICE
24  BASS_ERROR_NOPLAY
25  BASS_ERROR_FREQ
27  BASS_ERROR_NOTFILE
29  BASS_ERROR_NOHW
31  BASS_ERROR_EMPTY
32  BASS_ERROR_NONET
33  BASS_ERROR_CREATE
34  BASS_ERROR_NOFX
37  BASS_ERROR_NOTAVAIL
38  BASS_ERROR_DECODE
39  BASS_ERROR_DX
40  BASS_ERROR_TIMEOUT
Add-ons may introduce additional error codes.
Frees all resources used by the output device, including all its samples, streams and MOD musics.

BOOL BASS_Free();
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use **BASS_ErrorGetCode** to get the error code.
**Error codes**

BASS_ERROR_INIT  BASS_Init has not been successfully called.
Remarks
This function should be called for all initialized devices before the program closes. It is not necessary to individually free the samples_streams_musics as these are all automatically freed by this function.

When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
See also

BASS_Init
BASS_GetCPU

Retrieves the current CPU usage of BASS.

```
float BASS_GetCPU();
```
**Return value**
The BASS CPU usage as a percentage.
Remarks
This function includes the time taken to render stream (HSTREAM) and MOD music (HMUSIC) channels during playback, and any DSP functions set on those channels. It also includes any FX that are not using the "with FX flag" DX8 effect implementation. The rendering of some add-on stream formats may not be entirely included, if they use additional decoding threads; see the add-on documentation for details.

This function does not strictly tell the CPU usage, but rather how timely the processing is. For example, if it takes 10ms to generate 100ms of data, that would be 10%. If the reported usage gets to 100%, that means the channel data is being played faster than it can be generated and buffer underruns are likely to occur.

If automatic updating is disabled, then the value returned by this function is only updated after each call to BASS_Update. BASS_ChannelUpdate usage is not included. The CPU usage of an individual channel is available via the BASS_ATTRIB_CPU attribute.
Platform-specific
On Windows, the CPU usage does not include sample channels (HCHANNEL), which are mixed by the output device/drivers (hardware mixing) or Windows (software mixing). On other platforms, the CPU usage does include sample playback as well as the generation of the final output mix.
See also
BASS_CONFIG_UPDATETHREADS, BASS_ATTRIB_CPU
BASS_GetDevice

Retrieves the device setting of the current thread.

DWORD BASS_GetDevice();
Return value
If successful, the device number is returned, else -1 is returned. Use BASS_ErrorGetCode to get the error code.
Error codes

BASS_ERROR_INIT  BASS_Init has not been successfully called; there are no initialized devices.
See also

BASS_ChannelGetDevice, BASS_Init, BASS_SetDevice
BASS_GetDeviceInfo

Retrieves information on an output device.

```c
BOOL BASS_GetDeviceInfo(
    DWORD device,
    BASS_DEVICEINFO *info
);
```
**Parameters**

- **device**  The device to get the information of... 0 = first.
- **info**    Pointer to a structure to receive the information.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_DEVICE  device is invalid.
Remarks
This function can be used to enumerate the available devices for a setup dialog. Device 0 is always the "no sound" device, so you should start at device 1 if you only want to list real output devices.
**Platform-specific**
On Linux, a "Default" device is hardcoded to device number 1, which uses the default output set in the ALSA config, and the real devices start at number 2. That is also the case on Windows when the **BASS_CONFIG_DEV_DEFAULT** option is enabled.

On OSX, the BASS_DEVICES_AIRPLAY flag can be used in the `device` parameter to enumerate Airplay receivers instead of soundcards. A shared buffer is used for the Airplay receiver `name` information, which gets overwritten each time Airplay receiver information is requested, so it should be copied if needed. The **BASS_CONFIG_AIRPLAY** config option can be used to change which of the receiver(s) are used.
**Example**

Get the total number of devices currently present.

```c
int a, count=0;
BASS_DEVICEINFO info;
for (a=0; BASS_GetDeviceInfo(a, &info); a++)
    if (info.flags&BASS_DEVICE_ENABLED) // device is enabled
        count++; // count it
```

List all Airplay receivers available on OSX.

```c
int a;
BASS_DEVICEINFO info;
for (a=0; BASS_GetDeviceInfo(a|BASS_DEVICES_AIRPLAY, &info); a++)
    printf("%d: name=[%s] flags=%x\n", a, di.name, di.flags);
```
See also
BASS_GetInfo, BASS_Init, BASS_DEVICEINFO structure
BASS_GetDSoundObject

Retrieves a pointer to a DirectSound object interface.

```c
void *BASS_GetDSoundObject(
    DWORD object
);
```
Parameters

object  The interface to retrieve. This can be a HCHANNEL, HMUSIC or HSTREAM handle, in which case an **IDirectSoundBuffer** interface is returned, or one of the following.

- BASS_OBJECT_DS  Retrieve the **IDirectSound** interface.
- BASS_OBJECT_DS3DL  Retrieve the **IDirectSound3DListener** interface.
**Return value**
If successful, then a pointer to the requested object is returned, otherwise NULL is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
BASS_ERROR_ILLPARAM  object is invalid.
BASS_ERROR_NOTAVAIL  The requested object is not available with the current device.
Remarks
This function allows those that are familiar with DirectSound to access the internal DirectSound object interfaces, so that extra external functionality can be "plugged" into BASS. If you create any objects through a retrieved interface, make sure you release the objects before calling BASS_Free.

See the DirectX SDK for information on the DirectSound interfaces.

When using multiple devices, and requesting either the BASS_OBJECT_DS or BASS_OBJECT_DS3DL object interfaces, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
**Platform-specific**

DirectSound in a Windows thing, so this function is not available on other platforms.
Example
Set the speaker configuration to "headphones".

```c
#include <dsound.h>
...
IDirectSound *ds=BASS_GetDSoundObject(BASSOBJECT_DS); // get object
IDirectSound_SetSpeakerConfig(ds, DSSPEAKER_HEADPHONE); // headphones
```
See also
BASS_Init
BASS_GetInfo

Retrieves information on the device being used.

```c
BOOL BASS_GetInfo(
    BASS_INFO *info
);
```
**Parameters**

info  Pointer to a structure to receive the information.
Return value
If successful, TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
Remarks
When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
**Example**
Check if the current device has DirectSound support.

BASS_INFO info;
BASS_GetInfo(&info);
if (info.flags&DSCAPS_EMULDRIVER) {
    // device does NOT have DirectSound support
}
See also
BASS_GetDeviceInfo, BASS_INFO structure
BASS_GetVersion

Retrieves the version of BASS that is loaded.

DWORD BASS_GetVersion();
**Return value**
The BASS version. For example, 0x02040103 (hex), would be version 2.4.1.3
Remarks
There is no guarantee that a previous or future version of BASS supports all the BASS functions that you are using, so you should always use this function to make sure the correct version is loaded. It is safe to assume that future revisions (indicated in the LOWORD) will be fully compatible.

The BASS API includes a BASSVERSION constant, which can be used to check that the loaded BASS.DLL matches the API version used, ignoring revisions.
**Example**

Check that the correct BASS version is loaded, ignoring the revision.

```c
if (HIWORD(BASS_GetVersion())!=BASSVERSION) {
    // incorrect version loaded!
}
```

Check that revision 1.0 (or above) of the correct BASS version is loaded.

```c
if (HIWORD(BASS_GetVersion())!=BASSVERSION // check the main version
    || LOWORD(BASS_GetVersion())<0x100) { // check the revision
    // incorrect version loaded!
}
```
BASS_GetVolume

Retrieves the current master volume level.

```c
float BASS_GetVolume();
```
Return value
If successful, the volume level is returned, else -1 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  There is no volume control when using the "no sound" device.
BASS_ERROR_UNKNOWN  Some other mystery problem!
Remarks
When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
See also
BASS_SetVolume
Initializes an output device.

```c
BOOL BASS_Init(
    int device,
    DWORD freq,
    DWORD flags,
    HWND win,
    GUID *clsid
);
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>The device to use... -1 = default device, 0 = no sound, 1 = first real output device. <strong>BASS_GetDeviceInfo</strong> can be used to enumerate the available devices.</td>
</tr>
<tr>
<td>freq</td>
<td>Output sample rate.</td>
</tr>
<tr>
<td>flags</td>
<td>A combination of these flags.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_8BITS</strong> Use 8-bit resolution, else 16-bit.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_MONO</strong> Use mono, else stereo.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_3D</strong> Enable 3D functionality.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_LATENCY</strong> Calculates the latency of the device, that is the delay between requesting a sound to play and it actually being heard. A recommended minimum buffer length is also calculated. Both values are retrievable in the <strong>BASS_INFO</strong> structure (<strong>latency</strong> &amp; <strong>minbuf</strong> members). These calculations can increase the time taken by this function by 1-3 seconds.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_CPSPEAKERS</strong> Use the Windows control panel setting to detect the number of speakers. Soundcards generally have their own control panel to set the speaker config, so the Windows control panel setting may not be accurate unless it matches that. This flag has no effect on Vista, as the speakers are already accurately detected.</td>
</tr>
<tr>
<td></td>
<td><strong>BASS_DEVICE_SPEAKERS</strong> Force the enabling of <strong>speaker assignment</strong>. With some devices/drivers, the number of speakers BASS detects may be 2, when the device in fact supports...</td>
</tr>
</tbody>
</table>
more than 2 speakers. This flag forces the enabling of assignment to 8 possible speakers. This flag has no effect with non-WDM drivers.

BASS_DEVICE_NOSPEAKER Ignore speaker arrangement. This flag tells BASS not to make any special consideration for speaker arrangements when using the SPEAKER flags, eg. swapping the CENLFE and REAR speaker channels in 5/7.1 speaker output. This flag should be used with plain multi-channel (rather than 5/7.1) devices.

BASS_DEVICE_FREQ Set the device's output rate to \textit{freq}, otherwise leave it as it is.

BASS_DEVICE_DMX Initialize the device using the ALSA "dmix" plugin, else initialize the device for exclusive access.

\begin{itemize}
\item \texttt{win} The application's main window... 0 = the desktop window (use this for console applications).
\item \texttt{clsid} Class identifier of the object to create, that will be used to initialize DirectSound... NULL = use default.
\end{itemize}
Return value
If the device was successfully initialized, TRUE is returned, else FALSE is returned. Use \texttt{BASS_ErrorGetCode} to get the error code.
**Error codes**

BASS_ERROR_DX  
DirectX (or ALSA on Linux or OpenSL ES on Android) is not installed.

BASS_ERROR_DEVICE  
*device* is invalid.

BASS_ERROR_ALREADY  
The device has already been initialized.  
**BASS_Free** must be called before it can be initialized again.

BASS_ERROR_DRIVER  
There is no available device driver. The device may already be in use.

BASS_ERROR_FORMAT  
The specified format is not supported by the device. Try changing the *freq* and *flags* parameters.

BASS_ERROR_MEM  
There is insufficient memory.

BASS_ERROR_NO3D  
Could not initialize 3D support.

BASS_ERROR_UNKNOWN  
Some other mystery problem!
Remarks
This function must be successfully called before using any sample, stream or MOD music functions. The recording functions may be used without having called this function.

Playback is not possible with the "no sound" device, but it does allow the use of "decoding channels", eg. to decode files.

Simultaneously using multiple devices is supported in the BASS API via a context switching system; instead of there being an extra "device" parameter in the function calls, the device to be used is set prior to calling the functions. BASS_SetDevice is used to switch the current device. When successful, BASS_Init automatically sets the current thread's device to the one that was just initialized.

When using the default device ($device = -1$), BASS_GetDevice can be used to find out which device it was mapped to.
Platform-specific
On Linux, a "Default" device is hardcoded to device number 1, which uses the default output set in the ALSA config; that could map directly to one of the other devices or it could use ALSA plugins. If the `BASS_CONFIG_DEV_DEFAULT` config option has been enabled, a "Default" device is also available on Windows, who's output will follow default device changes on Windows 7. In both cases, the "Default" device will also be the default device (`device = -1`)

The sample format specified in the `freq` and `flags` parameters has no effect on the device output on iOS or OSX, and not on Windows unless VxD drivers are used (on Windows 98/95); with WDM drivers (on Windows XP/2000/Me/98SE), the output format is automatically set depending on the format of what is played and what the device supports, while on Vista and newer, the output format is determined by the user's choice in the Sound control panel. On Linux, the output device will use the specified format if possible, but will otherwise use a format as close to it as possible. On Android, the device's native sample rate (as reported by the AudioTrack getNativeOutputSampleRate method) will be used unless the `BASS_DEVICE_FREQ` flag is specified, in which case the `freq` parameter will be used (this only affects BASS's output format, not the device's output format). If the `BASS_DEVICE_FREQ` flag is specified on iOS or OSX, then the device's output rate will be set to the `freq` parameter if possible. The `BASS_DEVICE_FREQ` flag has no effect on other platforms. `BASS_GetInfo` can be used to check what the output format actually is.

On Windows, when specifying a class identifier (`clsid`), `BASS_GetDSoundObject` can be used to retrieve the DirectSound object after successful initialization, and through that access any special interfaces that the object may provide.

The `win` and `clsid` parameters are only used on Windows and are ignored on other platforms. That applies to the `BASS_DEVICE_CPSPEAKERS` and `BASS_DEVICE_SPEAKERS` flags too, as the number of available speakers is always accurately detected on the other platforms. The `BASS_DEVICE_LATENCY` flag is also ignored on those other platforms, as latency information is available without it.

The `BASS_DEVICE_DMIX` flag is only available on Linux, and allows multiple applications to share the device (if they all use "dmix"). It may also be
possible for multiple applications to use exclusive access if the device is capable of hardware mixing. If exclusive access initialization fails, the BASS_DEVICE_DMIX flag will automatically be tried; if that happens, it can be detected via BASS_GetInfo and the initflags.

On Linux, Android, and Windows CE, the length of the device's buffer can be set via the BASS_CONFIG_DEV_BUFFER config option.
**Example**
Initialize BASS to use the default output device, and a nominal format of 44100 Hz stereo 16-bit.

```c
BASS_Init(-1, 44100, 0, hwnd, NULL);
```
See also
BASS_Free, BASS_GetCPU, BASS_GetDeviceInfo, BASS_GetDSoundObject, BASS_GetInfo, BASS_MusicLoad, BASS_SampleCreate, BASS_SampleLoad, BASS_SetConfig, BASS_SetDevice, BASS_StreamCreate, BASS_StreamCreateFile, BASS_StreamCreateURL, BASS_Update, BASS_CONFIG_BUFFER, BASS_CONFIG_DEV_BUFFER, BASS_CONFIG_DEV_DEFAULT, BASS_CONFIG_UPDATEPERIOD
**BASS_Pause**

Stops the output, pausing all musics/samples/streams on it.

```c
BOOL BASS_Pause();
```
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
**Remarks**
Use `BASS_Start` to resume the output and paused channels.

When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
See also
BASS_ChannelPause, BASS_Start, BASS_Stop,
BASS_CONFIG_PAUSE_NOPLAY
Sets the device to use for subsequent calls in the current thread.

```c
BOOL BASS_SetDevice(
    DWORD device
);
```
Parameters
device  The device to use... 0 = no sound, 1 = first real output device.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.
**Error codes**

BASS_ERROR_DEVICE  *device* is invalid.

BASS_ERROR_INIT   The device has not been initialized.
Remarks
Simultaneously using multiple devices is supported in the BASS API via a context switching system; instead of there being an extra "device" parameter in the function calls, the device to be used is set prior to calling the functions. The device setting is local to the current thread, so calling functions with different devices simultaneously in multiple threads is not a problem.

The functions that use the device selection are the following: BASS_Free, BASS_GetDSoundObject, BASS_GetInfo, BASS_Start, BASS_Stop, BASS_Pause, BASS_SetVolume, BASS_GetVolume, BASS_Set3DFactors, BASS_Get3DFactors, BASS_Set3DPosition, BASS_Get3DPosition, BASS_SetEAXParameters, BASS_GetEAXParameters. It also determines which device is used by a new sample/stream/music: BASS_MusicLoad, BASS_SampleLoad, BASS_StreamCreateFile, etc.

When one of the above functions (or BASS_GetDevice) is called, BASS will check the current thread's device setting, and if no device is selected (or the selected device is not initialized), BASS will automatically select the lowest device that is initialized. This means that when using a single device, there is no need to use this function; BASS will automatically use the device that is initialized. Even if you free the device, and initialize another, BASS will automatically switch to the one that is initialized.
Example
Create a stream of an MP3 file on device 2.

BASS_SetDevice(2); // select device 2
stream=BASS_StreamCreateFile(FALSE, "afile.mp3", 0, 0, 0); // create
See also
BASS_ChannelGetDevice, BASS_ChannelSetDevice, BASS_GetDevice, BASS_Init
BASS_SetVolume

Sets the output master volume.

```c
BOOL BASS_SetVolume(  
    float volume  
);
```
**Parameters**

**volume**  The volume level... 0 (silent) to 1 (max).
**Return value**
If successful, then TRUE is returned, else FALSE 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**: There is no volume control when using the "no sound" device.
- **BASS_ERROR_ILLPARAM**: volume is invalid.
- **BASS_ERROR_UNKNOWN**: Some other mystery problem!
**Remarks**
The actual volume level may not be exactly the same as requested, due to underlying precision differences. `BASS_GetVolume` can be used to confirm what the volume is.

This function affects the volume level of all applications using the same output device. If you wish to only affect the level of your application's sounds, the `BASS_ATTRIB_VOL` attribute and/or the `BASS_CONFIG_GVOL_MUSIC` / `BASS_CONFIG_GVOL_SAMPLE` / `BASS_CONFIG_GVOL_STREAM` config options should be used instead.

When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
See also
BASS_GetVolume, BASS_ATTRIB_VOL, BASS_CONFIG_CURVE_VOL,
BASS_CONFIG_GVOL_MUSIC, BASS_CONFIG_GVOL_SAMPLE,
BASS_CONFIG_GVOL_STREAM
Starts (or resumes) the output.

```c
BOOL BASS_Start();
```
Return value
If successful, then TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_INIT  **BASS_Init** has not been successfully called.
Remarks
The output is automatically started by BASS_Init, so there is no need to use this function unless you have stopped or paused the output.

When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
See also
BASS_Pause, BASS_Stop
BASS_Stop

Stops the output, stopping all musics/samples/streams on it.

BOOL BASS_Stop();
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use \texttt{BASS\_ErrorGetCode} to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
Remarks
This function can be used after BASS_Pause to stop the paused channels, so that they will not be resumed the next time BASS_Start is called.

When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
See also
BASS_ChannelStop, BASS_Pause, BASS_Start
BASS_Update

Updates the HSTREAM and HMUSIC channel playback buffers.

```c
BOOL BASS_Update(
    DWORD length
);
```
**Parameters**

length  The amount of data to render, in milliseconds.
Return value
If successful, TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_NOTAVAIL  Updating is already in progress.
Remarks
When automatic updating is disabled, this function or BASS_ChannelUpdate needs to be used to keep the playback buffers updated. The length parameter should include some safety margin, in case the next update cycle gets delayed. For example, if calling this function every 100ms, 200 would be a reasonable length parameter.
See also
BASS_ChannelUpdate, BASS_CONFIG_BUFFER,
BASS_CONFIG_UPDATETHREADS
BASS_DEVICEINFO structure

Used with BASS_GetDeviceInfo or BASS_RecordGetDeviceInfo to retrieve information on a device.

typedef struct {
    char *name;
    char *driver;
    DWORD flags;
} BASS_DEVICEINFO;
Members
name Description of the device.
driver The filename of the driver.
flags The device's current status... a combination of these flags.
  BASS_DEVICE_ENABLED The device is enabled. It will not be possible to initialize the device if this flag is not present.
  BASS_DEVICE_DEFAULT The device is the system default.
  BASS_DEVICE_INIT The device is initialized, ie. BASS_Init or BASS_RecordInit has been called.

The type of device may also be indicated in the high 8 bits (use BASS_DEVICE_TYPE_MASK to test), and can be one of the following.
  BASS_DEVICE_TYPE_DIGITAL An audio endpoint device that connects to an audio adapter through a connector for a digital interface of unknown type.
  BASS_DEVICE_TYPE_DISPLAYPORT An audio endpoint device that connects to an audio adapter through a DisplayPort connector.
  BASS_DEVICE_TYPE_HANDSET The part of a telephone that is held in the hand and that contains a speaker and a microphone for two-way communication.
  BASS_DEVICE_TYPE_HDMI An audio endpoint device that connects to an audio adapter through a High-Definition Multimedia Interface (HDMI) connector.
  BASS_DEVICE_TYPE_HEADPHONES A set of headphones.
**BASS_DEVICE_TYPE_HEADSET**  
An earphone or a pair of earphones with an attached mouthpiece for two-way communication.

**BASS_DEVICE_TYPE_LINE**  
An audio endpoint device that sends a line-level analog signal to a line-input jack on an audio adapter or that receives a line-level analog signal from a line-output jack on the adapter.

**BASS_DEVICE_TYPE_MICROPHONE**  
A microphone.

**BASS_DEVICE_TYPE_NETWORK**  
An audio endpoint device that the user accesses remotely through a network.

**BASS_DEVICE_TYPE_SPDIF**  
An audio endpoint device that connects to an audio adapter through a Sony/Philips Digital Interface (S/PDIF) connector.

**BASS_DEVICE_TYPE_SPEAKERS**  
A set of speakers.
Remarks
When a device is disabled/disconnected, it is still retained in the device list, but the BASS_DEVICE_ENABLED flag is removed from it. If the device is subsequently re-enabled, it may become available again with the same device number, or the system may add a new entry for it.

When a new device is connected, it can affect the other devices and result in the system moving them to new device entries. If an affected device is initialized, it will stop working and will need to be reinitialized using its new device number.
Platform-specific
On Windows, *driver* can reveal the type of driver being used on systems that support both VxD and WDM drivers (Windows Me/98SE). Further information can be obtained from the file via the GetFileVersionInfo function. On Vista and newer, the device's endpoint ID is given rather than its driver filename. On OSX, *driver* is the device's UID, and on Linux it is the ALSA device name. It is unused on other platforms. The device type is only available on Windows (Vista and newer) and OSX. On Windows, DisplayPort devices will have BASS_DEVICE_TYPE_HDMI rather than BASS_DEVICE_TYPE_DISPLAYPORT.

Depending on the **BASS_CONFIG_UNICODE** config setting, *name* and *driver* can be in ANSI or UTF-8 form on Windows. They are always in UTF-16 form ("WCHAR" rather than "char") on Windows CE, and UTF-8 on other platforms.
See also
BASS_GetDeviceInfo, BASS_RecordGetDeviceInfo,
BASS_CONFIG_UNICODE
BASS_INFO structure

Used with BASS_GetInfo to retrieve information on the current device.

typedef struct {
    DWORD flags;
    DWORD hwsize;
    DWORD hwfree;
    DWORD freesam;
    DWORD free3d;
    DWORD minrate;
    DWORD maxrate;
    BOOL eax;
    DWORD minbuf;
    DWORD dsver;
    DWORD latency;
    DWORD initflags;
    DWORD speakers;
    DWORD freq;
} BASS_INFO;
Members

flags The device's capabilities... a combination of the following flags.

DSCAPS_CONTINUOUSRATE The device supports all sample rates between \textit{minrate} and \textit{maxrate}.

DSCAPS_EMULDRIVER The device's drivers do NOT have DirectSound support, so it is being emulated. Updated drivers should be installed.

DSCAPS_CERTIFIED The device driver has been certified by Microsoft. This flag is always set on WDM drivers.

DSCAPS_SECONDARYMONO Mono samples are supported by hardware mixing.

DSCAPS_SECONDARYSTEREO Stereo samples are supported by hardware mixing.

DSCAPS_SECONDARY8BIT 8-bit samples are supported by hardware mixing.

DSCAPS_SECONDARY16BIT 16-bit samples are supported by hardware mixing.

hwszie The device's total amount of hardware memory.

hwfree The device's amount of free hardware memory.

freesam The number of free sample slots in the hardware.

free3d The number of free 3D sample slots in the hardware.

minrate The minimum sample rate supported by the hardware.

maxrate The maximum sample rate supported by the hardware.

eax The device supports EAX and has it enabled? The device's "Hardware acceleration" needs to be set to "Full" in its "Advanced Properties" setup, else EAX is disabled. This is always FALSE if \textsc{BASS\_DEVICE\_3D} was not specified when \textsc{BASS\_Init} was called.

minbuf The minimum buffer length (rounded up to the nearest millisecond) recommended for use (with the \textsc{BASS\_CONFIG\_BUFFER} config
option).

dsver  DirectSound version... 9 = DX9/8/7/5 features are available, 8 = DX8/7/5 features are available, 7 = DX7/5 features are available, 5 = DX5 features are available. 0 = none of the DX9/8/7/5 features are available.

latency  The average delay (rounded up to the nearest millisecond) for playback of HSTREAM/HMUSIC channels to start and be heard.

initflags  The flags parameter of the BASS_Init call.

speakers  The number of available speakers, which can be accessed via the speaker assignment flags.

freq  The device's current output sample rate.
**Platform-specific**

On Windows, it is possible for *speakers* to mistakenly be 2 with some devices/drivers when the device in fact supports more speakers. In that case, the BASS_DEVICE_CPSPEAKERS flag can be used (with `BASS_Init`) to use the Windows control panel setting, or the BASS_DEVICE_SPEAKERS flag can be used to force the enabling of speaker assignment to up to 8 speakers, even though the device may not really support that many speakers. The result of assigning channels to nonexistent speakers is undefined; they may be heard on other speakers or not heard at all.

The *flags*, *hwsize*, *hwfree*, *freesam*, *free3d*, *minrate*, *maxrate*, *eax*, and *dsver* members are only used on Windows, as DirectSound and hardware mixing are only available there. The *freq* member is not available on Windows prior to Vista.

On Windows, the availability of the *latency* and *minbuf* values depends on the BASS_DEVICE_LATENCY flag being used when `BASS_Init` was called.
See also
BASS_GetInfo
BASS_Apply3D

Applies changes made to the 3D system.

`void BASS_Apply3D();`
Remarks

This function must be called to apply any changes made with
BASS_Set3DFactors, BASS_Set3DPosition, BASS_ChannelSet3DAttributes or
BASS_ChannelSet3DPosition. This allows multiple changes to be synchronized,
and also improves performance.

This function applies 3D changes on all the initialized devices. There is no need
to re-call it for each individual device when using multiple devices.
See also
BASS_ChannelSet3DAttributes, BASS_ChannelSet3DPosition,
BASS_Set3DFactors, BASS_Set3DPosition
BASS_Get3DFactors

Retrieves the factors that affect the calculations of 3D sound.

```c
BOOL BASS_Get3DFactors(
    float *distf,
    float *rollf,
    float *doppf
);
```
**Parameters**

- **distf**  The distance factor... NULL = don't retrieve it.
- **rollf**  The rolloff factor... NULL = don't retrieve it.
- **doppf**  The doppler factor... NULL = don't retrieve it.
Return value
If successful, then TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
BASS_ERROR_NO3D  The device was not initialized with 3D support.
Remarks
When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
See also
BASS_Set3DFactors
BASS_Get3DPosition

Retrieves the position, velocity, and orientation of the listener.

```c
BOOL BASS_Get3DPosition(
    BASS_3DVECTOR *pos,
    BASS_3DVECTOR *vel,
    BASS_3DVECTOR *front,
    BASS_3DVECTOR *top
);
```
Parameters

pos  The position of the listener... NULL = don't retrieve it.
vel  The listener's velocity... NULL = don't retrieve it.
front  The direction that the listener's front is pointing... NULL = don't retrieve it.
top  The direction that the listener's top is pointing... NULL = don't retrieve it.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

BASS_ERROR_INIT  BASS_Init has not been successfully called.
BASS_ERROR_NO3D  The device was not initialized with 3D support.
**Remarks**
The *front* and *top* parameters must both be retrieved in a single call, they cannot be retrieved individually.

When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
See also
BASS_Set3DPosition, BASS_3DVECTOR structure
BASS_GetEAXParameters

Retrieves the current type of EAX environment and its parameters.

```c
BOOL BASS_GetEAXParameters(
    DWORD *env,
    float *vol,
    float *decay
    float *damp
);
```
Parameters

env  The EAX environment... NULL = don't retrieve it. See BASS_SetEAXParameters for a list of the possible environments.
vol  The volume of the reverb... NULL = don't retrieve it.
decay The decay duration... NULL = don't retrieve it.
amp  The damping... NULL = don't retrieve it.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
BASS_ERROR_NOEAX  The current device does not support EAX.
Remarks
When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
Platform-specific
EAX and this function are only available on Windows.
See also
BASS_SetEAXParameters
BASS_Set3DFactors

Sets the factors that affect the calculations of 3D sound.

BOOL BASS_Set3DFactors(
    float distf,
    float rollf,
    float doppf
);
Parameters

distf The distance factor... 0 or less = leave current... examples: 1.0 = use meters, 0.9144 = use yards, 0.3048 = use feet. By default BASS measures distances in meters, you can change this setting if you are using a different unit of measurement.

rollf The rolloff factor, how fast the sound quietens with distance... 0.0 (min) - 10.0 (max), less than 0.0 = leave current... examples: 0.0 = no rolloff, 1.0 = real world, 2.0 = 2x real.

doppf The doppler factor... 0.0 (min) - 10.0 (max), less than 0.0 = leave current... examples: 0.0 = no doppler, 1.0 = real world, 2.0 = 2x real. The doppler effect is the way a sound appears to change pitch when it is moving towards or away from you. The listener and sound velocity settings are used to calculate this effect, this doppf value can be used to lessen or exaggerate the effect.
Return value
If successful, then TRUE is returned, else FALSE is returned. Use
BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_INIT   BASS_Init has not been successfully called.
BASS_ERROR_NO3D   The device was not initialized with 3D support.
Remarks
As with all 3D functions, use `BASS_Apply3D` to apply the changes.

When using multiple devices, the current thread's device setting (as set with `BASS_SetDevice`) determines which device this function call applies to.
Example
Use yards as the distance measurement unit, while leaving the current rolloff and
doppler factors untouched.

```c
BASS_Set3DFactors(0.9144, -1.0, -1.0);
BASS_Apply3D(); // apply the change
```
See also
BASS_Apply3D, BASS_Get3DFactors
Sets the position, velocity, and orientation of the listener (ie. the player).

```c
BOOL BASS_Set3DPosition(
    BASS_3DVECTOR *pos,
    BASS_3DVECTOR *vel,
    BASS_3DVECTOR *front,
    BASS_3DVECTOR *top
);
```
Parameters

pos  The position of the listener... NULL = leave current.

vel  The listener's velocity in units (as set with BASS_Set3DFactors) per second... NULL = leave current. This is only used to calculate the doppler effects, and in no way affects the listener's position.

front  The direction that the listener's front is pointing... NULL = leave current. This is automatically normalized.

top  The direction that the listener's top is pointing... NULL = leave current. This is automatically normalized, and adjusted to be at a right-angle to the front vector if necessary.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_INIT  BASS_Init has not been successfully called.
BASS_ERROR_NO3D  The device was not initialized with 3D support.
Remarks
The *front* and *top* parameters must both be set in a single call, they cannot be set individually. As with all 3D functions, use **BASS_Apply3D** to apply the changes.

When using multiple devices, the current thread's device setting (as set with **BASS_SetDevice**) determines which device this function call applies to.
See also
BASS_Apply3D, BASS_Get3DPosition, BASS_Set3DFactors, BASS_3DVECTOR structure
BASS_SetEAXParameters

Sets the type of EAX environment and its parameters.

```c
BOOL BASS_SetEAXParameters(
    int env,
    float vol,
    float decay,
    float damp
);
```
Parameters

env  The EAX environment... -1 = leave current, or one of the following.
     EAX_ENVIRONMENT_GENERIC,
     EAX_ENVIRONMENT_PADDEDCELL,
     EAX_ENVIRONMENT_ROOM,
     EAX_ENVIRONMENT_BATHROOM,
     EAX_ENVIRONMENT_LIVINGROOM,
     EAX_ENVIRONMENT_STONEROOM,
     EAX_ENVIRONMENT_AUDITORIUM,
     EAX_ENVIRONMENT_CONCERTHALL,
     EAX_ENVIRONMENT_CAVE, EAX_ENVIRONMENT_ARENA,
     EAX_ENVIRONMENT_HANGAR,
     EAX_ENVIRONMENT_CARPETEDHALLWAY,
     EAX_ENVIRONMENT_HALLWAY,
     EAX_ENVIRONMENT_STONECORRIDOR,
     EAX_ENVIRONMENT_ALLEY, EAX_ENVIRONMENT_FOREST,
     EAX_ENVIRONMENT_CITY,
     EAX_ENVIRONMENT_MOUNTAINS,
     EAX_ENVIRONMENT_QUARRY, EAX_ENVIRONMENT.PLAIN,
     EAX_ENVIRONMENT_PARKINGLOT,
     EAX_ENVIRONMENT_SEWERPIPE,
     EAX_ENVIRONMENT_UNDERWATER,
     EAX_ENVIRONMENT_DRUGGED,
     EAX_ENVIRONMENT_DIZZY,
     EAX_ENVIRONMENT_PSYCHOTIC.

vol  The volume of the reverb... 0 (off) to 1 (max), less than 0 = leave current.

decay  The time in seconds it takes the reverb to diminish by 60 dB... 0.1 (min)
        to 20 (max), less than 0 = leave current.

damp  The damping, high or low frequencies decay faster... 0 = high decays
        quickest, 1 = low/high decay equally, 2 = low decays quickest, less than
        0 = leave current.
Return value
If successful, then TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
Error codes

BASS_ERROR_INIT  
BASS_Init has not been successfully called.

BASS_ERROR_NOEAX  
The output device does not support EAX.
Remarks
The use of EAX functions requires that the output device supports EAX. BASS_GetInfo can be used to check that. EAX only affects 3D channels, but EAX functions do not require BASS_Apply3D to apply the changes.

Presets are provided for all the EAX environments. To use a preset, simply call BASS_SetEAXParameters(preset), where preset is one of the following.

EAX_PRESET_GENERIC, EAX_PRESET_PADDEDCELL,
EAX_PRESET_ROOM, EAX_PRESET_BATHROOM,
EAX_PRESET_LIVINGROOM, EAX_PRESET_STONEROOM,
EAX_PRESET_AUDITORIUM, EAX_PRESET_CONCERTHALL,
EAX_PRESET_CAVE, EAX_PRESET_ARENA, EAX_PRESET_HANGAR,
EAX_PRESET_CARPETEDHALLWAY, EAX_PRESET_HALLWAY,
EAX_PRESET_STONECORRIDOR, EAX_PRESET_ALLEY,
EAX_PRESET_FOREST, EAX_PRESET_CITY,
EAX_PRESET_MOUNTAINS, EAX_PRESET_QUARRY,
EAX_PRESET_PLAIN, EAX_PRESET_PARKINGLOT,
EAX_PRESET_SEWERPIPE, EAX_PRESET_UNDERWATER,
EAX_PRESET_DRUGGED, EAX_PRESET_DIZZY,
EAX_PRESET_PSYCHOTIC.

When using multiple devices, the current thread's device setting (as set with BASS_SetDevice) determines which device this function call applies to.
Platform-specific
This function is only available on Windows.
Example
Use the EAX_PRESET_ARENA preset.

BASS_SetEAXParameters(EAX_PRESET_ARENA);
See also
BASS_GetEAXParameters, BASS_ATTRIB_EAXMIX
BASS_3DVECTOR structure

Structure used by the 3D functions to describe positions, velocities, and orientations.

```c
typedef struct {
    float x;
    float y;
    float z;
} BASS_3DVECTOR;
```
Members

x  +ve = right, -ve = left.
y  +ve = up, -ve = down.
z  +ve = front, -ve = behind.
Remarks
As can be seen above, the left-handed coordinate system is used.
BASS_SampleCreate

Creates a new sample.

```c
HSAMPLE BASS_SampleCreate(
    DWORD length,
    DWORD freq,
    DWORD chans,
    DWORD max,
    DWORD flags
);
```
**Parameters**

length  The sample's length, in bytes.

freq  The default sample rate.

chans  The number of channels... 1 = mono, 2 = stereo, etc.

max  Maximum number of simultaneous playbacks... 1 (min) - 65535 (max)... use one of the BASS_SAMPLE_OVER flags to choose the override decider, in the case of there being no free channel available for playback (ie. the sample is already playing max times).

flags  A combination of these flags.

BASS_SAMPLE_8BITS  Use 8-bit resolution. If neither this or the BASS_SAMPLE_FLOAT flags are specified, then the sample is 16-bit.

BASS_SAMPLE_FLOAT  Use 32-bit floating-point sample data. Not really recommended for samples as it (at least) doubles the memory usage.

BASS_SAMPLE_LOOP  Looped? Note that only complete sample loops are allowed; you cannot loop just a part of the sample. More fancy looping can be achieved via streaming.

BASS_SAMPLE_SOFTWARE  Force the sample to not use hardware mixing.

BASS_SAMPLE_VAM  Enables the DX7 voice allocation and management features on the sample, which allows the sample to be played in software or hardware. This flag is ignored if the BASS_SAMPLE_SOFTWARE flag is also specified.

BASS_SAMPLE_3D  Enable 3D functionality. This requires that the BASS_DEVICE_3D flag was
specified when calling `BASS_Init`, and the sample must be mono (`chans=1`).

- **BASS_SAMPLE_MUTEMAX**: Mute the sample when it is at (or beyond) its max distance (software-mixed 3D samples only).
- **BASS_SAMPLE_OVER_VOL**: Override: the channel with the lowest volume is overridden.
- **BASS_SAMPLE_OVER_POS**: Override: the longest playing channel is overridden.
- **BASS_SAMPLE_OVER_DIST**: Override: the channel furthest away (from the listener) is overridden (3D samples only).
**Return value**
If successful, the new sample's handle is returned, else 0 is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td>BASS_Init has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>Sample functions are not available when using the &quot;no sound&quot; device.</td>
</tr>
<tr>
<td>BASS_ERROR_ILLPARAM</td>
<td>max is invalid.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>The sample format is not supported by the device/drivers.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_NO3D</td>
<td>Could not initialize 3D support.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
The sample's initial content is undefined. **BASS_SampleSetData** should be used to set the sample's data.

Unless the BASS_SAMPLESOFTWARE flag is used, the sample will use hardware mixing if hardware resources are available. Use **BASS_GetInfo** to see if there are hardware mixing resources available, and which sample formats are supported by the hardware. The BASS_SAMPLE_VAM flag allows a sample to be played by both hardware and software, with the decision made when the sample is played rather than when it is loaded. A sample's VAM options are set via **BASS_SampleSetInfo**.

To play a sample, first a channel must be obtained using **BASS_SampleGetChannel**, which can then be played using **BASS_ChannelPlay**.

If you want to play a large or one-off sample, then it would probably be better to stream it instead with **BASS_StreamCreate**.
Platform-specific
The BASS_SAMPLE_VAM flag requires DirectX 7 (or above). Away from Windows, all mixing is done in software (by BASS), so the BASS_SAMPLESOFTWARE flag is unnecessary.
Example
Create a 440 Hz sine wave sample.

```c
HSAMPLE sample = BASS_SampleCreate(256, 28160, 1, 1, BASS_SAMPLE_LOOP);
short data[128]; // data buffer
int a;
for (a=0; a<128; a++)
    data[a] = (short)(32767.0*sin((double)a*6.283185/64)); // sine wave
BASS_SampleSetData(sample, data); // set the sample's data
```
See also
BASS_SampleLoad, BASS_SampleSetData, BASS_StreamCreate
BASS_SampleFree

Frees a sample's resources.

BOOL BASS_SampleFree(
    HSAMPLE handle
);

**Parameters**

`handle`  The sample 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 not valid.
See also
BASS_SampleCreate, BASS_SampleLoad
BASS_SampleGetChannel

Creates/initializes a playback channel for a sample.

```c
HCHANNEL BASS_SampleGetChannel(
    HSAMPLE handle,
    BOOL onlynew
);
```
**Parameters**

- **handle**: Handle of the sample to play.
- **onlynew**: Do not recycle/override one of the sample's existing channels?
**Return value**
If successful, the handle of the new channel is returned, else NULL is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

- **BASS_ERROR_HANDLE**: `handle` is not a valid sample handle.
- **BASS_ERROR_NOCHAN**: The sample has no free channels... the maximum number of simultaneous playbacks has been reached, and no BASS_SAMPLE_OVER flag was specified for the sample or `onlynew = TRUE`.
- **BASS_ERROR_TIMEOUT**: The sample's minimum time gap ([BASS_SAMPLE](#)) has not yet passed since the last channel was created.
Remarks
Use BASS_SampleGetInfo and BASS_SampleSetInfo to set a sample's default attributes, which are used when creating a channel. After creation, a channel's attributes can be changed via BASS_ChannelSetAttribute, BASS_ChannelSet3DAttributes and BASS_ChannelSet3DPosition. BASS_Apply3D should be called before starting playback of a 3D sample, even if you just want to use the default settings.

If a sample has a maximum number of simultaneous playbacks of 1 (the max parameter was 1 when calling BASS_SampleLoad or BASS_SampleCreate), then the HCHANNEL handle returned will be identical to the HSAMPLE handle. That means you can use the HSAMPLE handle with functions that usually require a HCHANNEL handle, but you must still call this function first to initialize the channel.

When channel overriding has been enabled via a BASS_SAMPLE_OVER flag and there are multiple candidates for overriding (eg. with identical volume), the oldest of them will be chosen to make way for the new channel.

A sample channel is automatically freed when it's overridden by a new channel, or when stopped by BASS_ChannelStop, BASS_SampleStop or BASS_Stop. If you wish to stop a channel and re-use it, BASS_ChannelPause should be used to pause it instead. Determining whether a channel still exists can be done by trying to use the handle in a function call. A list of all the sample's existing channels can also be retrieved via BASS_SampleGetChannels.

The new channel will have an initial state of being paused (BASS_ACTIVE_PAUSED). This prevents the channel being claimed by another call of this function before it has been played, unless it gets overridden due to a lack of free channels.

All of a sample's channels share the same sample data, and just have their own individual playback state information (volume/position/etc).
Example
Play a sample with its default settings.

```c
HCHANNEL channel=BASS_SampleGetChannel(sample, FALSE); // get a sample
BASS_ChannelPlay(channel, FALSE); // play it
```
See also
BASS_ChannelPlay, BASS_ChannelSet3DAttributes,
BASS_ChannelSet3DPosition, BASS_ChannelSetAttribute,
BASS_SampleCreate, BASS_SampleGetChannels, BASS_SampleLoad,
BASS_SampleStop, BASS_CONFIG_SRC_SAMPLE
BASS_SampleGetChannels

Retrieves all a sample's existing channels.

```cpp
DWORD BASS_SampleGetChannels(
    HSAMPLE handle,
    HCHANNEL *channels
);
```
**Parameters**

**handle**  The sample handle.

**channels**  An array to put the sample's channel handles in. The array should be the same size as the sample's `max` setting when the sample was created, which can be retrieved using `BASS_SampleGetInfo`. NULL can be used to just check how many channels exist.
**Return value**
If successful, the number of existing channels is returned, else -1 is returned. Use
[BASS_ErrorGetCode](#) to get the error code.
**Error codes**

BASS_ERROR_HANDLE   *handle* is not a valid sample handle.
Remarks
To determine whether a particular sample channel still exists, it is simplest to just try it in a function call.
Example
Set the sample rate of all a sample's channels to 10000 Hz.

```c
BASS_SAMPLE info;
HCHANNEL *channels;
DWORD a, count;
BASS_SampleGetInfo(sample, &info); // get sample info for "max" value
channels=malloc(info.max*sizeof(HCHANNEL)); // allocate channels array
count=BASS_SampleGetChannels(sample, channels); // get the channels
for (a=0; a<count; a++) // go through them all and...
    BASS_ChannelSetAttribute(channels[a], BASS_ATTRIB_FREQ, 10000);
free(channels); // free the channels array
```
See also
BASS_SampleGetChannel, BASS_SampleGetInfo
BASS_SampleGetData

Retrieves a copy of a sample's data.

```c
BOOL BASS_SampleGetData(
    HSAMPLE handle,
    void *buffer
);
```
Parameters

handle  The sample handle.
buffer  Pointer to a buffer to receive the data.
**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 valid.
BASS_ERROR_UNKNOWN
Some other mystery problem!
Remarks
The buffer must be big enough to receive the sample's data, the size of which can be retrieved via BASS_SampleGetInfo.
See also
BASS_ChannelGetData, BASS_SampleSetData
BASS_SampleGetInfo

Retrieves a sample's default attributes and other information.

```c
BOOL BASS_SampleGetInfo(
    HSAMPLE handle,
    BASS_SAMPLE *info
);
```
**Parameters**

- **handle**  The sample handle.
- **info**  Pointer to a structure to receive the sample information.
**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 handle is invalid.
**Remarks**
Use this function and `BASS_SampleSetInfo` to edit a sample's default attributes.
See also
BASS_SampleSetInfo, BASS_SAMPLE structure
BASS_SampleLoad

Loads a WAV, AIFF, MP3, MP2, MP1, OGG or plugin supported sample.

```
HSAMPLE BASS_SampleLoad(
    BOOL mem,
    void *file,
    QWORD offset,
    DWORD length,
    DWORD max,
    DWORD flags
);
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>TRUE = load the sample from memory.</td>
</tr>
<tr>
<td>file</td>
<td>Filename (mem = FALSE) or a memory location (mem = TRUE).</td>
</tr>
<tr>
<td>offset</td>
<td>File offset to load the sample from (only used if mem = FALSE).</td>
</tr>
<tr>
<td>length</td>
<td>Data length... 0 = use all data up to the end of file (if mem = FALSE). If length over-runs the end of the file, it will automatically be lowered to the end of the file.</td>
</tr>
<tr>
<td>max</td>
<td>Maximum number of simultaneous playbacks... 1 (min) - 65535 (max). Use one of the BASS_SAMPLE_OVER flags to choose the override decider, in the case of there being no free channel available for playback (ie. the sample is already playing max times).</td>
</tr>
<tr>
<td>flags</td>
<td>A combination of these flags.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_FLOAT</td>
<td>Use 32-bit floating-point sample data. Not really recommended for samples as it (at least) doubles the memory usage.</td>
</tr>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>Looped? Note that only complete sample loops are allowed, you cannot loop just a part of the sample. More fancy looping can be achieved by streaming the file.</td>
</tr>
<tr>
<td>BASS_SAMPLE_MONO</td>
<td>Convert the sample (MP3/MP2/MP1 only) to mono, if it is not already. This flag is automatically applied if BASS_DEVICE_MONO was specified when calling BASS_Init.</td>
</tr>
<tr>
<td>BASS_SAMPLE_SOFTWARE</td>
<td>Force the sample to not use hardware mixing.</td>
</tr>
</tbody>
</table>
| BASS_SAMPLE_VAM       | Enables the DX7 voice allocation and management features on the sample, which allows the sample to be played in software or hardware. This flag is ignored if the BASS_SAMPLE_SOFTWARE flag
<table>
<thead>
<tr>
<th>Codepoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_3D</td>
<td>Enable 3D functionality. This requires that the BASS_DEVICE_3D flag was specified when calling BASS_Init, and the sample must be mono.</td>
</tr>
<tr>
<td>BASS_SAMPLE_MUTEMAX</td>
<td>Mute the sample when it is at (or beyond) its max distance (software-mixed 3D samples only).</td>
</tr>
<tr>
<td>BASS_SAMPLE_OVER_VOL</td>
<td>Override: the channel with the lowest volume is overridden.</td>
</tr>
<tr>
<td>BASS_SAMPLE_OVER_POS</td>
<td>Override: the longest playing channel is overridden.</td>
</tr>
<tr>
<td>BASS_SAMPLE_OVER_DIST</td>
<td>Override: the channel furthest away (from the listener) is overridden (3D samples only).</td>
</tr>
<tr>
<td>BASS_UNICODE</td>
<td><em>file</em> is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, and UTF-8 on other platforms.</td>
</tr>
</tbody>
</table>
Return value
If successful, the loaded sample's handle is returned, else 0 is returned. Use BASS_ErrorGetCode to get the error code.
### Error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td>BASS_Init has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>Sample functions are not available when using the &quot;no sound&quot; device.</td>
</tr>
<tr>
<td>BASS_ERROR_ILLPARAM</td>
<td>max and/or length is invalid. The length must be specified when loading from memory.</td>
</tr>
<tr>
<td>BASS_ERROR_FILEOPEN</td>
<td>The file could not be opened.</td>
</tr>
<tr>
<td>BASS_ERROR_FILEFORM</td>
<td>The file's format is not recognised/supported.</td>
</tr>
<tr>
<td>BASS_ERROR_CODEC</td>
<td>The file uses a codec that is not available/supported. This can apply to WAV and AIFF files, and also MP3 files when using the &quot;MP3-free&quot; BASS version.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>The sample format is not supported by the device/drivers. If the sample is more than stereo or the BASS_SAMPLE_FLOAT flag is used, it could be that they are not supported.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_NO3D</td>
<td>Could not initialize 3D support.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>


Remarks
This function supports the same file formats as \texttt{BASS\_StreamCreateFile} does, including those supported via the plugin system.

Unless the \texttt{BASS\_SAMPLE\_SOFTWARE} flag is used, the sample will use hardware mixing if hardware resources are available. Use \texttt{BASS\_GetInfo} to see if there are hardware mixing resources available, and which sample formats are supported by the hardware. The \texttt{BASS\_SAMPLE\_VAM} flag allows a sample to be played by both hardware and software, with the decision made when the sample is played rather than when it is loaded. A sample's VAM options are set via \texttt{BASS\_SampleSetInfo}.

To play a sample, first a channel must be obtained using \texttt{BASS\_SampleGetChannel}, which can then be played using \texttt{BASS\_ChannelPlay}.

After loading a sample from memory (\texttt{mem = TRUE}), the memory can safely be discarded, as a copy is made.

If you want to play a large or one-off sample, then it would probably be better to stream it instead with \texttt{BASS\_StreamCreateFile}.
**Platform-specific**
The BASS_SAMPLE_VAM flag requires DirectX 7 (or above). Away from Windows, all mixing is done in software (by BASS), so the BASS_SAMPLE_SOFTWARE flag is unnecessary.
See also
BASS_SampleCreate, BASS_SampleFree, BASS_SampleGetChannel,
BASS_SampleGetInfo, BASS_StreamCreateFile
BASS_SampleSetData

Sets a sample's data.

```c
BOOL BASS_SampleSetData(
    HSAMPLE handle,
    void *buffer
);
```
**Parameters**

- **handle**  The sample handle.
- **buffer**  Pointer to the data.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_HANDLE</td>
<td><code>handle</code> is not valid.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
**Remarks**
The required length and format of the data can be retrieved via `BASS_SampleGetInfo`.

A sample's data can be set at any time, including during playback.
See also
BASS_SampleCreate, BASS_SampleGetData
Sets a sample's default attributes.

```c
BOOL BASS_SampleSetInfo(
    HSAMPLE handle,
    BASS_SAMPLE *info
);
```
**Parameters**

handle  The sample handle.

info     Pointer to the sample information structure.
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 handle is invalid.
BASS_ERROR_ILLPARAM     The BASS_SAMPLE max value is invalid.
Remarks
Use this function and BASS_SampleGetInfo to edit a sample's default attributes. Changing a sample's default attributes does not affect any existing channels, it only affects channels subsequently created via BASS_SampleGetChannel. The exception is the VAM settings, changes to that apply to all the sample's channels at their next playback (BASS_ChannelPlay). Use BASS_ChannelSetAttribute and BASS_ChannelSet3DAtributes to change the attributes of an existing sample channel.

The sample's maximum number of simultaneous playbacks can be changed via the max member of the BASS_SAMPLE structure. If the new maximum is lower than the existing number of channels, the channels will remain existing until they are stopped.

The length, origres and chans members of the BASS_SAMPLE structure cannot be modified; any changes are ignored. The BASS_SAMPLE_8BITS, BASS_SAMPLE_MONO, BASS_SAMPLE_3D, BASS_SAMPLE_MUTEMAX, BASS_SAMPLE_SOFTWARE and BASS_SAMPLE_VAM flags also cannot be changed.
Example
Set a sample's default volume to 0.5.

BASS_SAMPLE info;
BASS_SampleGetInfo(sample, &info); // get the sample's current info
info.volume=0.5; // set the default volume to 0.5
BASS_SampleSetInfo(sample, &info); // set the updated info
See also
BASS_ChannelSet3DAttributes, BASS_ChannelSetAttribute,
BASS_SampleGetInfo, BASS_SAMPLE structure
BASS_SampleStop

Stops all instances of a sample.

```c
BOOL BASS_SampleStop(
    HSAMPLE handle
);
```
Parameters

handle  The sample 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 not a valid sample.
**Remarks**
If a sample is playing simultaneously multiple times, calling this function will stop them all, which is obviously simpler than calling `BASS_ChannelStop` multiple times.
See also
BASS_ChannelStop
BASS_SAMPLE structure

Used with BASS_SampleGetInfo and BASS_SampleSetInfo to retrieve and set the default playback attributes of a sample.

typedef struct {
    DWORD freq;
    float volume;
    float pan;
    DWORD flags;
    DWORD length;
    DWORD max;
    DWORD origres;
    DWORD chans;
    DWORD mingap;
    DWORD mode3d;
    float mindist;
    float maxdist;
    DWORD iangle;
    DWORD oangle;
    float outvol;
    DWORD vam;
    DWORD priority;
} BASS_SAMPLE;
Members
freq
Default sample rate.
volume
Default volume... 0 (silent) to 1 (full).
pan
Default panning position... -1 (full left) to +1 (full right), 0 = centre.
flags
A combination of these flags.

BASS_SAMPLE_8BITS
8-bit resolution. If neither this or the BASS_SAMPLE_FLOAT flags are present, then the sample is 16-bit.

BASS_SAMPLE_FLOAT
32-bit floating-point.

BASS_SAMPLE_LOOP
Looped?

BASS_SAMPLE_3D
The sample has 3D functionality enabled.

BASS_SAMPLE_MUTE_MAX
Mute the sample when it is at (or beyond) its max distance (3D samples only).

BASS_SAMPLE_SOFTWARE
The sample is not using hardware mixing.

BASS_SAMPLE_VAM
DX7 voice allocation and management features are enabled (see below).

BASS_SAMPLE_OVER_VOL
Override: the channel with the lowest volume is overridden.

BASS_SAMPLE_OVER_POS
Override: the longest playing channel is overridden.

BASS_SAMPLE_OVER_DIST
Override: the channel furthest away (from the listener) is overridden (3D samples only).

length
The length in bytes.
max
Maximum number of simultaneous playbacks.
origres
The original resolution (bits per sample)... 0 = undefined.
chans
Number of channels... 1 = mono, 2 = stereo, etc.
mimgap
Minimum time gap in milliseconds between creating channels using
**BASS_SampleGetChannel.** This can be used to prevent flanging effects caused by playing a sample multiple times very close to each other. The default setting, after loading/creating a sample, is 0 (disabled).

**The following are the sample's default 3D attributes (if the sample is 3D).**

- **mode3d**  The 3D processing mode... one of these flags.
  - **BASS_3DMODE_NORMAL**  Normal 3D processing.
  - **BASS_3DMODE_RELATIVE**  The sample's 3D position (position/velocity/orientation) is relative to the listener. When the listener's position/velocity/orientation is changed with **BASS_Set3DPosition**, the sample's position relative to the listener does not change.
  - **BASS_3DMODE_OFF**  Turn off 3D processing on the sample, the sound will be played in the centre.

- **mindist**  The minimum distance. The sample's volume is at maximum when the listener is within this distance.
- **maxdist**  The maximum distance. The sample's volume stops decreasing when the listener is beyond this distance.
- **iangle**  The angle of the inside projection cone in degrees... 0 (no cone) to 360 (sphere).
- **oangle**  The angle of the outside projection cone in degrees... 0 (no cone) to 360 (sphere).
- **outvol**  The delta-volume outside the outer projection cone... 0 (silent) to 1 (full).

**The following are the sample's DX7 voice allocation/management settings (if VAM is enabled).**

- **vam**  voice allocation/management flags... a combination of these
  - **BASS_VAM_HARDWARE**  Play the sample in hardware. If no hardware voices are available then the **play** call will fail.
BASS_VAM_SOFTWARE  Play the sample in software (ie. non-accelerated). No other VAM flags may be used together with this flag.

The following flags enable hardware resource stealing... if the hardware has no available voices, a currently playing buffer will be stopped to make room for the new buffer. Only samples with VAM enabled are considered for termination.

BASS_VAM_TERM_TIME  If there are no free hardware voices, the buffer to be terminated will be the one with the least time left to play.

BASS_VAM_TERM_DIST  If there are no free hardware voices, the buffer to be terminated will be one that was loaded/created with the BASS_SAMPLE_MUTEMAX flag and is beyond its max distance (maxdist). If there are no buffers that match this criteria, then the play call will fail.

BASS_VAM_TERM_PRIO  If there are no free hardware voices, the buffer to be terminated will be the one with the lowest priority. This flag may be used with the TERM_TIME or TERM_DIST flag, if multiple voices have the same priority then the time or distance is used to decide which to terminate.

priority  Priority, used with the BASS_VAM_TERM_PRIO flag... 0 (min) to 0xFFFFFFFF (max).
Remarks
When a sample has 3D functionality, the `iangle` and `oangle` angles decide how wide the sound is projected around the orientation angle (as set via `BASS_ChannelSet3DPosition`). Within the inside angle the volume level is the level set in the `volume` member (or the `BASS_ATTRIB_VOL` attribute when the sample is playing). Outside the outer angle, the volume changes according to the `outvol` value. Between the inner and outer angles, the volume gradually changes between the inner and outer volume levels. If the inner and outer angles are 360 degrees, then the sound is transmitted equally in all directions.

When VAM is enabled, and neither the `BASS_VAM_HARDWARE` or `BASS_VAM_SOFTWARE` flags are specified, then the sample will be played in hardware if resources are available, and in software if no hardware resources are available.
See also
BASS_SampleCreate, BASS_SampleGetInfo, BASS_SampleLoad, BASS_SampleSetInfo
BASS_StreamCreate

Creates a user sample stream.

```
HSTREAM BASS_StreamCreate(
    DWORD freq,  
    DWORD chans,  
    DWORD flags,  
    STREAMPROC *proc, 
    void *user
);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>freq</strong></td>
<td>The default sample rate. The sample rate can be changed using <strong>BASS_ChannelSetAttribute</strong>.</td>
</tr>
<tr>
<td><strong>chans</strong></td>
<td>The number of channels... 1 = mono, 2 = stereo, 4 = quadraphonic, 6 = 5.1, 8 = 7.1.</td>
</tr>
<tr>
<td><strong>flags</strong></td>
<td>A combination of these flags.</td>
</tr>
<tr>
<td><strong>BASS_SAMPLE_8BITS</strong></td>
<td>Use 8-bit resolution. If neither this or the BASS_SAMPLE_FLOAT flags are specified, then the stream is 16-bit.</td>
</tr>
<tr>
<td><strong>BASS_SAMPLE_FLOAT</strong></td>
<td>Use 32-bit floating-point sample data. See Floating-point channels for info.</td>
</tr>
<tr>
<td><strong>BASS_SAMPLE_SOFTWARE</strong></td>
<td>Force the stream to not use hardware mixing.</td>
</tr>
<tr>
<td><strong>BASS_SAMPLE_3D</strong></td>
<td>Enable 3D functionality. This requires that the BASS_DEVICE_3D flag was specified when calling <strong>BASS_Init</strong>, and the stream must be mono (chans=1). The SPEAKER flags cannot be used together with this flag.</td>
</tr>
<tr>
<td><strong>BASS_SAMPLE_FX</strong></td>
<td>Enable the old implementation of DirectX 8 effects. See the DX8 effect implementations section for details. Use <strong>BASS_ChannelSetFX</strong> to add effects to the stream.</td>
</tr>
<tr>
<td><strong>BASS_STREAM_AUTOFREE</strong></td>
<td>Automatically free the stream when playback ends.</td>
</tr>
<tr>
<td><strong>BASS_STREAM_DECODE</strong></td>
<td>Decode the sample data, without playing it. Use <strong>BASS_ChannelGetData</strong> to retrieve decoded sample data. The BASS_SAMPLE_3D, BASS_STREAM_AUTOFREE and</td>
</tr>
</tbody>
</table>
SPEAKER flags cannot be used together with this flag. The BASS_SAMPLE_SOFTWARE and BASS_SAMPLE_FX flags are also ignored.

**BASS_SPEAKER***<xxx>*** Speaker assignment flags. These flags have no effect when the stream is more than stereo.

**proc** The user defined stream writing function, or one of the following.

**STREAMPROC_DUMMY** Create a "dummy" stream. A dummy stream does not have any sample data of its own, but a decoding dummy stream (with BASS_STREAM_DECODE flag) can be used to apply DSP/FX processing to any sample data, by setting DSP/FX on the stream and feeding the data through **BASS_ChannelGetData**. The dummy stream should have the same sample format as the data being fed through it.

**STREAMPROC_PUSH** Create a "push" stream. Instead of BASS pulling data from a **STREAMPROC** function, data is pushed to BASS via **BASS_StreamPutData**.

**user** User instance data to pass to the callback function. Unused when creating a dummy or push stream.
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td>BASS_Init has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>Only decoding channels (BASS_STREAM_DECODE) are allowed when using the &quot;no sound&quot; device. The BASS_STREAM_AUTOFREE flag is also unavailable to decoding channels.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_SPEAKER</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_NO3D</td>
<td>Could not initialize 3D support.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
Sample streams allow any sample data to be played through BASS, and are particularly useful for playing a large amount of sample data without requiring a large amount of memory. If you wish to play a sample format that BASS does not support, then you can create a stream and decode the sample data into it.

BASS can automatically stream MP3, MP2, MP1, OGG, WAV and AIFF files, using `BASS_StreamCreateFile`, and also from HTTP and FTP servers, using `BASS_StreamCreateURL`. `BASS_StreamCreateFileUser` allows streaming from other sources too.
**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_ChannelPlay, BASS_ChannelSetAttribute, BASS_ChannelSetDSP, BASS_ChannelSetFX, BASS_ChannelSetLink, BASS_StreamCreateFile, BASS_StreamCreateFileUser, BASS_StreamCreateURL, BASS_StreamFree, BASS_StreamPutData, STREAMPROC callback, BASS_CONFIG_BUFFER
**BASS_StreamCreateFile**

Creates a sample stream from an MP3, MP2, MP1, OGG, WAV, AIFF or plugin supported file.

```c
HSTREAM BASS_StreamCreateFile(
    BOOL mem,
    void *file,
    QWORD offset,
    QWORD length,
    DWORD flags
);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mem</strong></td>
<td>TRUE = stream the file from memory.</td>
</tr>
<tr>
<td><strong>file</strong></td>
<td>Filename (mem = FALSE) or a memory location (mem = TRUE).</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>File offset to begin streaming from (only used if mem = FALSE).</td>
</tr>
<tr>
<td><strong>length</strong></td>
<td>Data length... 0 = use all data up to the end of the file (if mem = FALSE)</td>
</tr>
<tr>
<td><strong>flags</strong></td>
<td>A combination of these flags.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_FLOAT</td>
<td>Use 32-bit floating-point sample data. <a href="#">Floating-point channels</a> for info.</td>
</tr>
<tr>
<td>BASS_SAMPLE_MONO</td>
<td>Decode/play the stream (MP3/MP2/M only) in mono, reducing the CPU usage (if it was originally stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling <strong>BASS_Init</strong>.</td>
</tr>
<tr>
<td>BASS_SAMPLE_SOFTWARE</td>
<td>Force the stream to not use hardware mixing.</td>
</tr>
<tr>
<td>BASS_SAMPLE_3D</td>
<td>Enable 3D functionality. This requires that the BASS_DEVICE_3D flag was specified when calling <strong>BASS_Init</strong>, and the stream must be mono. The SPEAKER flags cannot be used together with this flag.</td>
</tr>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>Loop the file. This flag can be toggled any time using <strong>BASS_ChannelFlags</strong>.</td>
</tr>
<tr>
<td>BASS_SAMPLE_FX</td>
<td>Enable the old implementation of DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details. Use <strong>BASS_ChannelSetFX</strong> to add effects to the stream.</td>
</tr>
</tbody>
</table>
| BASS_STREAM_PRESCAN | Pre-scan the file for accurate seek points and length reading in MP3/MP2/MP1 files and chained OGG files (has no effect on normal OGG files). This can significantly increase the time taken to
create the stream, particularly with a large file and/or slow storage media. **BASS_ChannelSetPosition** can be used to scan the file after stream creation instead.

**BASS_STREAM_AUTOFREE** 
Automatically free the stream when playback ends.

**BASS_STREAM_DECODE** 
Decode the sample data, without playing it. Use **BASS_ChannelGetData** to retrieve decoded sample data. The **BASS_SAMPLE_3D**, **BASS_STREAM_AUTOFREE** and **SPEAKER** flags cannot be used together with this flag. The **BASS_SAMPLESOFTWARE** and **BASS_SAMPLEFX** flags are also ignored.

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

**BASS_ASYNCFILE** 
Read the file asynchronously. When enabled, the file is read and buffered in parallel with the decoding, to reduce the chances of the decoder being affected by I/O delays. This can be particularly useful with slow storage media and/or low latency output. The size of the file buffer is determined by the **BASS_CONFIG_ASYNCFILE_BUF** config option. This flag is ignored when streaming from memory (mem = TRUE).

**BASS_UNICODE** 
*File* is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, a UTF-8 on other platforms.
**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_ILLPARAM**  
The length must be specified when streaming from memory.

**BASS_ERROR_FILEOPEN**  
The file could not be opened.

**BASS_ERROR_FILEFORM**  
The file's format is not recognised/supported.

**BASS_ERROR_CODEC**  
The file uses a codec that is not available/supported. This can apply to WAV and AIFF files, and also MP3 files when using the "MP3-free" BASS version.

**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
BASS has built-in support for MPEG, OGG, WAV and AIFF files. Support for additional formats is available through BASS_PluginLoad.

MPEG 1.0, 2.0 and 2.5 layer 3 (MP3) files are supported, layers 1 (MP1) and 2 (MP2) are also supported. Standard RIFF and RF64 WAV files are supported. All PCM formats from 8 to 32-bit are supported in WAV and AIFF files, but the output will be restricted to 16-bit unless the BASS_SAMPLE_FLOAT flag is used. 64-bit floating-point WAV and AIFF files are also supported, but they are rendered in 16-bit or 32-bit floating-point depending on the flags. The file's original resolution is available from BASS_ChannelGetInfo.

Chained OGG files containing multiple logical bitstreams are supported, but seeking within them is only fully supported if the BASS_STREAM_PRESCAN flag is used (or the BASS_CONFIG_OGG_PRESCAN option is enabled) to have them pre-scanned. Without pre-scanning, seeking will only be possible back to the start. The BASS_POS_OGG "mode" can be used with BASS_ChannelGetLength to get the number of bitstreams and with BASS_ChannelSetPosition to seek to a particular one. A BASS_SYNC_OGG_CHANGE sync can be set via BASS_ChannelSetSync to be informed of when a new bitstream begins during decoding/playback.

Multi-channel (ie. more than stereo) OGG, WAV and AIFF files are supported.

Use BASS_ChannelGetInfo to retrieve information on the format (sample rate, resolution, channels) of the stream. The playback length of the stream can be retrieved using BASS_ChannelGetLength.

If length = 0 (use all data up to the end of the file), and the file length increases after creating the stream (ie. the file is still being written), then BASS will play the extra data too, but the length returned by BASS_ChannelGetLength will not be updated until the end is reached. The BASS_StreamGetPosition return values will be updated during playback of the extra data though.

When streaming from memory (mem = TRUE), the memory must not be freed before the stream is freed. There may be exceptions to that with some add-ons (see the documentation).
To stream a file from the internet, use `BASS_StreamCreateURL`. To stream from other locations, see `BASS_StreamCreateFileUser`.
**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.

On Windows and Windows CE, ACM codecs are supported with compressed WAV files. Media Foundation codecs are also supported on Windows 7 and updated versions of Vista, including support for AAC and WMA. On iOS and OSX, CoreAudio codecs are supported, including support for AAC and ALAC. Media Foundation and CoreAudio codecs are only tried after the built-in decoders and any plugins have rejected the file. Built-in support for IMA and Microsoft ADPCM WAV files is provided on Linux/Android/Windows CE, while they are supported via ACM and CoreAudio codecs on Windows and OSX/iOS.
Example
Create a stream from an MP3 file.

HSTREAM stream=BASS_StreamCreateFile(FALSE, "afile.mp3", 0, 0, 0);
See also
BASS_ChannelGetInfo, BASS_ChannelGetLength, BASS_ChannelGetTags,
BASS_ChannelPlay, BASS_ChannelSetAttribute, BASS_ChannelSetDSP,
BASS_ChannelSetFX, BASS_ChannelSetLink, BASS_StreamCreate,
BASS_StreamCreateFileUser, BASS_StreamCreateURL, BASS_StreamFree,
BASS_StreamGetFilePosition, BASS_CONFIG_VERIFY
BASS_StreamCreateFileUser

Creates a sample stream from an MP3, MP2, MP1, OGG, WAV, AIFF or plugin supported file via user callback functions.

```
HSTREAM BASS_StreamCreateFileUser(
    DWORD system,
    DWORD flags,
    BASS_FILEPROCS *procs,
    void *user
);
```
Parameters

**system**  File system to use, one of the following.
- **STREAMFILE_NOBUFFER**  Unbuffered.
- **STREAMFILE_BUFFER**  Buffered.
- **STREAMFILE_BUFFERPUSH**  Buffered, with the data pushed to BASS via `BASS_StreamPutFileData`.

**flags**  A combination of these flags.
- **BASS_SAMPLE_FLOAT**  Use 32-bit floating-point sample data. See [Floating-point channels](#) for info.
- **BASS_SAMPLE_MONO**  Decode/play the stream (MP3/MP2/M only) in mono, reducing the CPU usage (if it was originally stereo). This flag automatically applied if `BASS_DEVICE_MONO` was specified when calling `BASS_Init`.
- **BASS_SAMPLE_SOFTWARE**  Force the stream to not use hardware mixing.
- **BASS_SAMPLE_3D**  Enable 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 cannot be used together with this flag.
- **BASS_SAMPLE_LOOP**  Loop the file. This flag can be toggled at any time using `BASS_ChannelFlags`. This flag is ignored when streaming in blocks (BASS_STREAM_BLOCK).
- **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_PRESCAN**  Pre-scan the file for accurate seek point and length reading in MP3/MP2/MP1
files and chained OGG files (has no effect on normal OGG files). This can significantly increase the time taken to create the stream, particularly with a large file and/or slow storage media. This flag only applies when using the STREAMFILE_NOBUFFER system.

**BASS_STREAM_RESTRATE**
Restrict the "download" rate of the file to the rate required to sustain playback. If this flag is not used, then the file will downloaded as quickly as possible. This flag only has effect when using the STREAMFILE_BUFFER system.

**BASS_STREAM_BLOCK**
Download and play the file in smaller chunks. Uses a lot less memory than otherwise, but it is not possible to seek or loop the stream; once it has ended, the file must be opened again to play it again. This flag will automatically be applied when the file length is unknown. This flag also has the effect of restricting the download rate. This flag has no effect when using the STREAMFILE_NOBUFFER system.

**BASS_STREAM_AUTOFREE**
Automatically free the stream when playback ends.

**BASS_STREAM_DECODE**
Decode the sample data, without playing it. Use **BASS_ChannelGetData** to retrieve decoded sample data. The BASS_SAMPLE_3D, BASS_STREAM_AUTOFREE and SPEAKER flags cannot be used together with this flag. The BASS_SAMPLE_SOFTWARE and BASS_SAMPLE_FX flags are also ignored.
BASS_SPEAKER_

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

BASS_ASYNCFILE

Read the file asynchronously. When enabled, the file is read and buffered in parallel with the decoding, to reduce the chances of the decoder being affected by I/O delays. This can be particularly useful with slow storage media and/or low latency output. The size of the file buffer is determined by the BASS_CONFIG_ASYNCFILE_BUFFER config option. This flag only applies to using the STREAMFILE_NOBUFFER system.

procs The user defined file functions.

user User instance data to pass to the callback functions.
**Return value**
If successful, the new stream's handle is returned, else 0 is returned. Use [BASS_ErrorGetCode](#) to get the error code.
<table>
<thead>
<tr>
<th>Error codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td>BASS_Init has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>Only decoding channels (BASS_STREAM_DECODE) are allowed when using the &quot;no sound&quot; device. The BASS_STREAM_AUTOFREE flag is also unavailable to decoding channels.</td>
</tr>
<tr>
<td>BASS_ERROR_ILLPARAM</td>
<td>system is not valid.</td>
</tr>
<tr>
<td>BASS_ERROR_FILEFORM</td>
<td>The file's format is not recognised/supported.</td>
</tr>
<tr>
<td>BASS_ERROR_CODEC</td>
<td>The file uses a codec that is not available/supported. This can apply to WAV and AIFF files, and also MP3 files when using the &quot;MP3-free&quot; BASS version.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_SPEAKER</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_NO3D</td>
<td>Could not initialize 3D support.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
The buffered file system (STREAMFILE_BUFFER) is what is used by BASS_StreamCreateURL. As the name suggests, data from the file is buffered so that it is readily available for decoding; BASS creates a thread dedicated to "downloading" the data. This is ideal for when the data is coming from a source that has high latency, like the internet. It is not possible to seek in buffered file streams, until the download has reached the requested position; it is not possible to seek at all if it is being streamed in blocks.

The push buffered file system (STREAMFILE_BUFFERPUSH) is the same, except that instead of the file data being pulled from the FILEREADPROC function in a "download" thread, the data is pushed to the stream via BASS_StreamPutFileData. A FILEREADPROC function is still required, to get the initial data used in the creation of the stream.

The unbuffered file system (STREAMFILE_NOBUFFER) is what is used by BASS_StreamCreateFile. In this system, BASS does not do any intermediate buffering; it simply requests data from the file as and when it needs it. This means that reading (FILEREADPROC) must be quick, otherwise the decoding will be delayed and playback buffer underruns (old data repeated) are a possibility. It is not so important for seeking (FILESEEKPROC) to be fast, as that is generally not required during decoding, except when looping a file.

In all cases, BASS will automatically stall playback of the stream when insufficient data is available, and resume it when enough data does become available.

A copy is made of the procs callback function table, so it does not need to persist beyond this function call.
**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.

On Windows and Windows CE, ACM codecs are supported with compressed WAV files. Media Foundation codecs are also supported on Windows 7 and updated versions of Vista, including support for AAC and WMA. On iOS and OSX, CoreAudio codecs are supported, including support for AAC and ALAC. Media Foundation and CoreAudio codecs are only tried after the built-in decoders and any plugins have rejected the file. Built-in support for IMA and Microsoft ADPCM WAV files is provided on Linux/Android/Windows CE, while they are supported via ACM and CoreAudio codecs on Windows and OSX/iOS.
See also
BASS_ChannelGetInfo, BASS_ChannelGetLength, BASS_ChannelGetTags, BASS_ChannelPlay, BASS_ChannelSetAttribute, BASS_ChannelSetDSP, BASS_ChannelSetFX, BASS_ChannelSetLink, BASS_StreamCreateFile, BASS_StreamCreateURL, BASS_StreamFree, BASS_StreamGetPosition, BASS_StreamPutFileData, BASS_FILEPROCS structure, BASS_CONFIG_NET_BUFFER
BASS_StreamCreateURL

Creates a sample stream from an MP3, MP2, MP1, OGG, WAV, AIFF or plugin supported file on the internet, optionally receiving the downloaded data in a callback function.

HSTREAM BASS_StreamCreateURL(
    char *url,
    DWORD offset,
    DWORD flags,
    DOWNLOADPROC *proc,
    void *user
);


Parameters

url URL of the file to stream. Should begin with "http://" or "https://" or "ftp://", or another add-on supported protocol. The URL can be followed by custom HTTP request headers to be sent to the server; the URL and each header should be terminated with a carriage return and line feed ("\r\n").

offset File position to start streaming from. This is ignored by some servers, specifically when the length is unknown/undefined.

flags A combination of these flags.

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_FLOAT</td>
<td>Use 32-bit floating-point sample data. See <a href="#">Floating-point channels</a> for info.</td>
</tr>
<tr>
<td>BASS_SAMPLE_MONO</td>
<td>Decode/play the stream (MP3/MP2/MP1 only) in mono, reducing the CPU usage (if it was originally stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling BASS_Init.</td>
</tr>
<tr>
<td>BASS_SAMPLE_SOFTWARE</td>
<td>Force the stream to not use hardware mixing.</td>
</tr>
<tr>
<td>BASS_SAMPLE_3D</td>
<td>Enable 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 cannot be used together with this flag.</td>
</tr>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>Loop the file. This flag can be toggled at any time using BASS_ChannelFlags. This flag is ignored when streaming in blocks (BASS_STREAM_BLOCK).</td>
</tr>
<tr>
<td>BASS_SAMPLE_FX</td>
<td>Enable the old implementation of DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details.</td>
</tr>
</tbody>
</table>
Use **BASS_ChannelSetFX** to add effects to the stream.

**BASS_STREAM_RESTRATE**  Restrict the download rate of the file to the rate required to sustain playback. If this flag is not used, then the file will be downloaded as quickly as the user's internet connection allows.

**BASS_STREAM_BLOCK**  Download and play the file in smaller chunks, instead of downloading the entire file to memory. Uses a lot less memory than otherwise, but it is not possible to seek or loop the stream; once it has ended, the file must be opened again to play it again. This flag will automatically be applied when the file length is unknown, for example with Shout/Icecast streams. This flag also has the effect of restricting the download rate.

**BASS_STREAM_STATUS**  Pass status info (HTTP/ICY tags) from the server to the **DOWNLOADPROC** callback during connection. This can be useful to determine the reason for a failure.

**BASS_STREAM_AUTOFREE**  Automatically free the stream when playback ends.

**BASS_STREAM_DECODE**  Decode the sample data, without playing it. Use **BASS_ChannelGetData** to retrieve decoded sample data. The **BASS_SAMPLE_3D**, **BASS_STREAM_AUTOFREE** and **SPEAKER** flags cannot be used together with this flag. The
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_SOFTWARE</td>
<td>BASS SAMPLE SOFTWARE and BASS SAMPLE FX flags are also ignored.</td>
</tr>
<tr>
<td>BASS_SAMPLE_FX</td>
<td></td>
</tr>
<tr>
<td>BASS_SPEAKER_xxx</td>
<td>Speaker assignment flags. These flags have no effect when the stream is more than stereo.</td>
</tr>
<tr>
<td>BASS_UNICODE</td>
<td>url is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, and UTF-8 on other platforms.</td>
</tr>
</tbody>
</table>

| proc                | Callback function to receive the file as it is downloaded... NULL = no callback. |
| user                | User instance data to pass to the callback function.                        |
Return value
If successful, the new stream's handle is returned, else 0 is returned. Use \texttt{BASS_ErrorGetCode} to get the error code.
**Error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td><strong>BASS_Init</strong> has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>Only decoding channels (BASS_STREAM_DECODE) are allowed when using the &quot;no sound&quot; device. The BASS_STREAM_AUTOFREE flag is also unavailable to decoding channels.</td>
</tr>
<tr>
<td>BASS_ERROR_NONET</td>
<td>No internet connection could be opened. Can be caused by a bad <strong>proxy</strong> setting.</td>
</tr>
<tr>
<td>BASS_ERROR_I LLPARAM</td>
<td><strong>url</strong> is not a valid URL.</td>
</tr>
<tr>
<td>BASS_ERROR_SSL</td>
<td>SSL/HTTPS support is not available.</td>
</tr>
<tr>
<td>BASS_ERROR_TIMEOUT</td>
<td>The server did not respond to the request within the timeout period, as set with the <strong>BASS_CONFIG_NET_TIMEOUT</strong> config option.</td>
</tr>
<tr>
<td>BASS_ERROR_FILEOPEN</td>
<td>The file could not be opened.</td>
</tr>
<tr>
<td>BASS_ERROR_FILEFORM</td>
<td>The file's format is not recognised/supported.</td>
</tr>
<tr>
<td>BASS_ERROR_CODEC</td>
<td>The file uses a codec that is not available/supported. This can apply to WAV and AIFF files, and also MP3 files when using the &quot;MP3-free&quot; BASS version.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_SPEAKER</td>
<td>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.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_NO3D</td>
<td>Could not initialize 3D support.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
Use `BASS_ChannelGetInfo` to retrieve information on the format (sample rate, resolution, channels) of the stream. The playback length of the stream can be retrieved using `BASS_ChannelGetLength`.

When playing the stream, BASS will stall the playback if there is insufficient data to continue playing. Playback will automatically be resumed when sufficient data has been downloaded. `BASS_ChannelIsActive` can be used to check if the playback is stalled, and the progress of the file download can be checked with `BASS_StreamGetFilePosition`.

When streaming in blocks (BASS_STREAM_BLOCK flag), be careful not to stop/pause the stream for too long, otherwise the connection may timeout due to there being no activity and the stream will end prematurely.

When streaming from Shoutcast servers, metadata (track titles) may be sent by the server. The data can be retrieved with `BASS_ChannelGetTags`. A `BASS_SYNC_META` sync can also be set via `BASS_ChannelSetSync` to be informed when metadata is received. A `BASS_SYNC_OGG_CHANGE` sync can be used to be informed of when a new logical bitstream begins in an Icecast/OGG stream.

When using an offset, the file length returned by `BASS_StreamGetFilePosition` can be used to check that it was successful by comparing it with the original file length. Another way to check is to inspect the HTTP headers retrieved with `BASS_ChannelGetTags`.

Custom HTTP request headers may be ignored by some plugins, notably `BASSWMA`. 
**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.

On Windows and Windows CE, ACM codecs are supported with compressed WAV files. Media Foundation codecs are also supported on Windows 7 and updated versions of Vista, including support for AAC and WMA. On iOS and OSX, CoreAudio codecs are supported, including support for AAC and ALAC. Media Foundation and CoreAudio codecs are only tried after the built-in decoders and any plugins have rejected the file. Built-in support for IMA and Microsoft ADPCM WAV files is provided on Linux/Android/Windows CE, while they are supported via ACM and CoreAudio codecs on Windows and OSX/iOS.
**Example**
Stream an MP3 file.

```c
HSTREAM stream=BASS_StreamCreateURL("http://www.asite.com/afile.mp3'
```

Stream an MP3 file with a cookie sent to the server.

```c
HSTREAM stream=BASS_StreamCreateURL("http://www.asite.com/afile.mp3'
```
See also
BASS_ChannelGetInfo, BASS_ChannelGetLength, BASS_ChannelGetTags,
BASS_ChannelPlay, BASS_ChannelSetAttribute, BASS_ChannelSetDSP,
BASS_ChannelSetFX, BASS_StreamCreateFile, BASS_StreamCreateFileUser,
BASS_StreamFree, BASS_StreamGetFilePosition, DOWNLOADPROC
callback, BASS_ATTRIB_NET_RESUME, BASS_CONFIG_NET_AGENT,
BASS_CONFIG_NET_BUFFER, BASS_CONFIG_NET_PASSIVE,
BASS_CONFIG_NET_PREBUF, BASS_CONFIG_NET_PROXY,
BASS_CONFIG_NET_READTIMEOUT, BASS_CONFIG_NET_TIMEOUT,
BASS_CONFIG_VERIFY_NET
BASS_StreamFree

Frees a sample stream's resources, including any sync/DSP/FX it has.

```c
BOOL BASS_StreamFree(
    HSTREAM handle
);
```
Parameters

handle  The stream 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 not valid.
BASS_StreamGetFilePosition

Retrieves the file position/status of a stream.

QWORD BASS_StreamGetFilePosition(
    HSTREAM handle,
    DWORD mode
);

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The stream handle.</td>
</tr>
<tr>
<td>mode</td>
<td>The file position/status to retrieve. One of the following.</td>
</tr>
<tr>
<td>BASS_FILEPOS_ASYNCBUF</td>
<td>The amount of data in the asynchronous file reading buffer. This requires</td>
</tr>
<tr>
<td></td>
<td>that the BASS_ASYNCFILE flag was used at the stream's creation.</td>
</tr>
<tr>
<td>BASS_FILEPOS_BUFFER</td>
<td>The amount of data in the buffer of an internet file stream or &quot;buffered&quot;</td>
</tr>
<tr>
<td></td>
<td>user file stream. Unless streaming in blocks, this is the same as</td>
</tr>
<tr>
<td></td>
<td>BASS_FILEPOS_DOWNLOAD.</td>
</tr>
<tr>
<td>BASS_FILEPOS_CONNECTED</td>
<td>Internet file stream or &quot;buffered&quot; user file stream is still connected?</td>
</tr>
<tr>
<td></td>
<td>0 = no, 1 = yes.</td>
</tr>
<tr>
<td>BASS_FILEPOS_CURRENT</td>
<td>Position that is to be decoded for playback next. This will be a bit</td>
</tr>
<tr>
<td></td>
<td>ahead of the position actually being heard due to buffering.</td>
</tr>
<tr>
<td>BASS_FILEPOS_DOWNLOAD</td>
<td>Download progress of an internet file stream or &quot;buffered&quot; user file stream.</td>
</tr>
<tr>
<td>BASS_FILEPOS_END</td>
<td>End of audio data. When streaming in blocks (the BASS_STREAM_BLOCK flag is</td>
</tr>
<tr>
<td></td>
<td>in effect), the download buffer length is given.</td>
</tr>
<tr>
<td>BASS_FILEPOS_SIZE</td>
<td>Total size of the file.</td>
</tr>
<tr>
<td>BASS_FILEPOS_START</td>
<td>Start of audio data.</td>
</tr>
</tbody>
</table>

*other modes may be supported by add-ons, see the documentation.*
**Return value**
If successful, then the requested file position/status is returned, else -1 is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

BASS_ERROR_HANDLE  \( handle \) is not valid.
BASS_ERROR_NOTFILE  The stream is not a file stream.
BASS_ERROR_NOTAVAIL The requested file position/status is not available.
Remarks
ID3 tags (both v1 and v2) and WAVE headers, as well as any other rubbish at the start of the file, are excluded from the BASS_FILEPOS_CURRENT, BASS_FILEPOS_DOWNLOAD, and BASS_FILEPOS_END positions. This is useful for average bitrate calculations, but it means that they may not be actual file positions. The BASS_FILEPOS_START position can be added to get the actual file position.

When streaming a file from the internet or a "buffered" user file stream, the entire file is downloaded even if the audio data ends before that, in case there are tags to be read. This means that the BASS_FILEPOS_DOWNLOAD position may go beyond the BASS_FILEPOS_END position.

It is unwise to use the BASS_FILEPOS_CURRENT position for syncing purposes because it gives the position that is being decoded, not the position that is being heard. Use BASS_ChannelGetPosition and/or BASS_ChannelSetSync instead.
Example
Get the average bitrate of a file.

```c
float time = BASS_ChannelBytes2Seconds(stream, BASS_ChannelGetLength(stream));
DWORD len = BASS_StreamGetFilePosition(stream, BASS_FILEPOS_END); // file length
DWORD bitrate = (DWORD)(len / (125 * time) + 0.5); // bitrate (Kbps)
```

Get the percentage downloaded of an internet file stream, or the buffer level when streaming in blocks.

```c
QWORD len = BASS_StreamGetFilePosition(stream, BASS_FILEPOS_END); //
QWORD buf = BASS_StreamGetFilePosition(stream, BASS_FILEPOS_BUFFER);
float progress = buf * 100.0 / len; // percentage of buffer filled
```
See also
BASS_ChannelGetPosition, BASS_ChannelGetLength
BASS_StreamPutData

Adds sample data to a "push" stream.

```c
DWORD BASS_StreamPutData(
    HSTREAM handle,
    void *buffer,
    DWORD length
);
```
**Parameters**

- **handle**  The stream handle.
- **buffer**  Pointer to the sample data... NULL = allocate space in the queue buffer so that there is at least *length* bytes of free space.
- **length**  The amount of data in bytes, optionally using the BASS_STREAMPROC_END flag to signify the end of the stream. 0 can be used to just check how much data is queued.
Return value
If successful, the amount of queued data is returned, else -1 is returned. Use BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_HANDLE  
*handle* is not valid.

BASS_ERROR_NOTAVAIL  
The stream is not using the push system.

BASS_ERROR_ILLPARAM  
*length* is not valid, it must equate to a whole number of samples.

BASS_ERROR_ENDED  
The stream has ended.

BASS_ERROR_MEM  
There is insufficient memory.
Remarks
As much data as possible will be placed in the stream's playback buffer, and any remainder will be queued for when more space becomes available, i.e. as the buffered data is played. With a decoding channel, there is no playback buffer, so all data is queued in that case. There is no limit to the amount of data that can be queued (besides available memory); the queue buffer will be automatically enlarged as required to hold the data, but it can also be enlarged in advance. The queue buffer is freed when the stream ends or is reset, e.g. via BASS_ChannelPlay (with restart = TRUE) or BASS_ChannelSetPosition (with pos = 0).

DSP/FX are applied when the data reaches the playback buffer, or the BASS_ChannelGetData call in the case of a decoding channel.

Data should be provided at a rate sufficient to sustain playback. If the buffer gets exhausted, BASS will automatically stall playback of the stream, until more data is provided. BASS_ChannelGetData (BASS_DATA_AVAILABLE) can be used to check the buffer level, and BASS_ChannelIsActive can be used to check if playback has stalled. A BASS_SYNC_STALL sync can also be set via BASS_ChannelSetSync, to be triggered upon playback stalling or resuming.
See also
BASS_StreamCreate, STREAMPROC callback
**BASS_StreamPutFileData**

Adds data to a "push buffered" user file stream's buffer.

```c
DWORD BASS_StreamPutFileData(
    HSTREAM handle,
    void *buffer,
    DWORD length
);
```
**Parameters**

- handle: The stream handle.
- buffer: Pointer to the file data.
- length: The amount of data in bytes, or `BASS_FILEDATA_END` to end the file.
**Return value**
If successful, the number of bytes read from *buffer* is returned, else -1 is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_HANDLE  handle is not valid.
BASS_ERROR_NOTAVAIL  The stream is not using the STREAMFILE_BUFFERPUSH file system.
BASS_ERROR_ENDED  The file has ended.
Remarks
If there is not enough space in the stream's file buffer to receive all of the data, then only the amount that will fit is read from buffer. **BASS_StreamGetFilePosition** can be used to check the amount of space in the buffer.

File data should be provided at a rate sufficient to sustain playback. If there is insufficient file data, and the playback buffer is subsequently exhausted, BASS will automatically stall playback of the stream, until more data is available. A BASS_SYNC_STALL sync can be set via **BASS_ChannelSetSync**, to be triggered upon playback stalling or resuming.
See also
BASS_StreamCreateFileUser, BASS_StreamGetPosition
DOWNLOADPROC callback

Internet stream download callback function.

```c
void CALLBACK DownloadProc(
    const void *buffer,
    DWORD length,
    void *user
);
```
**Parameters**

- **buffer**: Pointer to the downloaded data... NULL = finished downloading.
- **length**: The number of bytes in the buffer... 0 = HTTP or ICY tags.
- **user**: The user instance data given when `BASS_StreamCreateURL` was called.
Remarks
The callback will be called before the `BASS_StreamCreateURL` call returns (if it's successful), with the initial downloaded data. So any initialization (eg. creating the file if writing to disk) needs to be done either before the call, or in the callback function.

When the BASS_STREAM_STATUS flag is specified in the `BASS_StreamCreateURL` call, HTTP and ICY tags may be passed to the callback during connection, before any stream data is received. The tags are given exactly as would be returned by `BASS_ChannelGetTags`. You can distinguish between HTTP and ICY tags by checking what the first string starts with: "HTTP" or "ICY".

A download callback function could be used in conjunction with a BASS_SYNC_META sync set via `BASS_ChannelSetSync` to save individual tracks to disk from a Shoutcast stream.
Example
Stream an MP3 file, and save a local copy.

```c
FILE *file=NULL;
...
void CALLBACK MyDownloadProc(const void *buffer, DWORD length, void *user)
{
    if (!file) file=fopen("afile.mp3", "wb"); // create the file
    if (!buffer) fclose(file); // finished downloading
    else fwrite(buffer, 1, length, file);
}
...
HSTREAM stream=BASS_StreamCreateURL("http://www.asite.com/afile.mp3",
```
See also
BASS_StreamCreateURL
FILECLOSEPROC callback

User file stream close callback function.

```c
void CALLBACK FileCloseProc(
    void *user
);
```
**Parameters**

**user**  The user instance data given when `BASS_StreamCreateFileUser` was called.
Remarks
With a buffered file stream, this function is called as soon as reading reaches the end of the file. If the stream is freed before then, this function could be called while its `FILEREADPROC` function is in progress. If that happens, the `FILEREADPROC` function call should be immediately cancelled.
See also
BASS_StreamCreateFileUser, BASS_FILEPROCS structure
User file stream length callback function.

```c
QWORD CALLBACK FileLenProc(
    void *user
);
```
Parameters
user  The user instance data given when BASS_StreamCreateFileUser was called.
**Return value**
The length of the file in bytes. Returning 0 for a buffered file stream makes BASS stream the file in blocks and is equivalent to using the BASS_STREAM_BLOCK flag in the [BASS_StreamCreateFileUser](https://www.bass-club.de/Docs/BASS_StreamCreateFileUser) call.
Remarks
This function is called first thing, and only the once with buffered streams. With unbuffered streams, it may be called again when testing for EOF (end of file), allowing the file to grow in size.
See also
BASS_StreamCreateFileUser, BASS_FILEPROCS structure
FILEREADPROC callback

User file stream read callback function.

```c
DWORD CALLBACK FileReadProc(
    void *buffer,
    DWORD length,
    void *user
);
```
**Parameters**

- **buffer**: Pointer to the buffer to put the data in.
- **length**: Maximum number of bytes to read.
- **user**: The user instance data given when [BASS_StreamCreateFileUser](https://www.bass-context.com/doc.php?doc=api/BASS_StreamCreateFileUser) was called.
**Return value**
The number of bytes read... -1 = end of file, 0 = end of file (buffered file stream only).
**Remarks**
During creation of the stream, this function should try to return the amount of data requested. After that, it can just return whatever is available up to the requested amount.

For an unbuffered file stream, this function should be as quick as possible during playback; any delays will not only affect the decoding of the current stream, but can also affect other streams and MOD musics that are playing. It is better to return less data (even none) rather than wait for more data. A buffered file stream is not affected by delays like this, as this function runs in its own thread then.
See also
BASS_StreamCreateFileUser, BASS_FILEPROCS structure
FILESEEKPROC callback

User file stream seek callback function.

```c
BOOL CALLBACK FileSeekProc(
    QWORD offset,
    void *user
);
```
**Parameters**

offset  Position in bytes to seek to.

user  The user instance data given when BASS_StreamCreateFileUser was called.
Return value
TRUE if successful, else FALSE.
See also
BASS_StreamCreateFileUser, BASS_FILEPROCS structure
STREAMPROC callback

User stream writing callback function.

```c
DWORD CALLBACK StreamProc(
    HSTREAM handle,
    void *buffer,
    DWORD length,
    void *user
);
```
Parameters

- **handle**: The stream that needs writing.
- **buffer**: Pointer to the buffer to write the sample data in. The data should be as follows: 8-bit samples are unsigned, 16-bit samples are signed, 32-bit floating-point samples range from -1 to +1.
- **length**: The maximum number of bytes to write.
- **user**: The user instance data given when BASS_StreamCreate was called.
**Return value**
The number of bytes written by the function, optionally using the BASS_STREAMPROC_END flag to signify that the end of the stream is reached.
**Remarks**
A stream writing function should be as quick as possible, because other streams (and MOD musics) cannot be updated until it has finished. It is better to return less data quickly, rather than spending a long time delivering exactly the amount requested.

Although a STREAMPROC may return less data than BASS requests, be careful not to do so by too much, too often. If the buffer gets exhausted, BASS will automatically stall playback of the stream, until more data is provided. **BASS_ChannelGetData** *(BASS_DATA_AVAILABLE)* can be used to check the buffer level, and **BASS_ChannelIsActive** can be used to check if playback has stalled. A **BASS_SYNC_STALL** sync can also be set via **BASS_ChannelSetSync**, to be triggered upon playback stalling or resuming. If you do return less than the requested amount of data, the number of bytes should still equate to a whole number of samples.

Some functions can cause problems if called from within a stream (or DSP) function. Do not call **BASS_Stop** or **BASS_Free** from within a stream callback, and do not call **BASS_ChannelStop** or **BASS_StreamFree** with the same handle as received by the callback.

When streaming multi-channel sample data, the channel order of each sample is as follows.

- **3 channels** left-front, right-front, center.
- **4 channels** left-front, right-front, left-rear/side, right-rear/side.
- **5 channels** left-front, right-front, center, left-rear/side, right-rear/side.
- **6 channels** left-front, right-front, center, LFE, left-rear/side, right-rear/side.
- **(5.1)**
- **8 channels** left-front, right-front, center, LFE, left-rear/side, right-rear/side, left-rear center, right-rear center.
Example
A callback function to stream a file, in 44100 Hz stereo 16-bit.

```c
FILE *file;
...

// the stream writing callback
DWORD CALLBACK MyStreamProc(HSTREAM handle, void *buf, DWORD len, void *user)
{
  DWORD c=fread(buf, 1, len, file); // read the file into the buffer
  if (feof(file)) c|=BASS_STREAMPROC_END; // end of the file/stream
  return c;
}
...

HSTREAM stream=BASS_StreamCreate(44100, 2, 0, MyStreamProc, 0); // create the stream
```
See also
BASS_StreamCreate, BASS_StreamPutData
BASS_FILEPROCS structure

Table of callback functions used with `BASS_StreamCreateFileUser`.

typedef struct {
    FILECLOSEPROC *close;
    FILELENPROC  *length;
    FILEREADPROC *read;
    FILESEEKPROC *seek;
} BASS_FILEPROCS;
**Members**

- close: Callback function to close the file.
- length: Callback function to get the file length.
- read: Callback function to read from the file.
- seek: Callback function to seek in the file. Not used by buffered file streams.
Example
Stream a file from disk via an "unbuffered" user file stream.

```c
void CALLBACK MyFileCloseProc(void *user)
{
    fclose(user); // close the file
}

QWORD CALLBACK MyFileLenProc(void *user)
{
    struct stat s;
    fstat(fileno(user), &s);
    return s.st_size; // return the file length
}

DWORD CALLBACK MyFileReadProc(void *buffer, DWORD length, void *user)
{
    return fread(buffer, 1, length, user); // read from file
}

BOOL CALLBACK MyFileSeekProc(QWORD offset, void *user)
{
    return fseek(user, offset, SEEK_SET); // seek to offset
}

...

FILE *file=fopen("a_file.mp3", "rb"); // open the file
stream=BASS_StreamCreateFileUser(STREAMFILE_NOBUFFER, 0, &fileprocs, file, 0);
```

NOTE: This is just an example. It is simpler to use `BASS_StreamCreateFile` to stream a file from disk.
See also
BASS_StreamCreateFileUser, FILECLOSEPROC callback, FILELENPROC callback, FILEREADPROC callback, FILESEEKPROC callback
BASS_MusicFree

Frees a MOD music's resources, including any sync/DSP/FX it has.

BOOL BASS_MusicFree(
    HMUSIC handle
);
**Parameters**

handle  The MOD music 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 not valid.
See also
BASS_MusicLoad
BASS_MusicLoad

Loads a MOD music file.

HMUSIC BASS_MusicLoad(
    BOOL mem,
    void *file,
    QWORD offset,
    DWORD length,
    DWORD flags,
    DWORD freq
);

Parameters

**mem**  TRUE = load the MOD music from memory.

**file**  Filename (mem = FALSE) or a memory location (mem = TRUE).

**offset**  File offset to load the MOD music from (only used if mem = FALSE).

**length**  Data length... 0 = use all data up to the end of file (if mem = FALSE).

**flags**  A combination of these flags.

- **BASS_SAMPLE_8BITS**  Use 8-bit resolution. If neither this or the **BASS_SAMPLE_FLOAT** flags are specified, then the sample data will be 16-bit.

- **BASS_SAMPLE_FLOAT**  Use 32-bit floating-point sample data. See [Floating-point channels](#) for info.

- **BASS_SAMPLE_MONO**  Decode/play the MOD music in mono (uses less CPU than stereo). This flag is automatically applied if **BASS_DEVICE_MONO** was specified when calling **BASS_Init**.

- **BASS_SAMPLE_SOFTWARE**  Force the MOD music to not use hardware mixing.

- **BASS_SAMPLE_3D**  Enable 3D functionality. This requires that the **BASS_DEVICE_3D** flag was specified when calling **BASS_Init**. 3D channels must also be mono, so **BASS_SAMPLE_MONO** is automatically applied. The **SPEAKER** flags cannot 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 music.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>Loop the music.</td>
</tr>
<tr>
<td>BASS_MUSIC_NONINTER</td>
<td>Use non-interpolated sample mixing. This generally reduces the sound quality, but can be good for chip-tunes.</td>
</tr>
<tr>
<td>BASS_MUSIC_SINCINTER</td>
<td>Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. If neither this or the BASS_MUSIC_NONINTER flag is specified, linear interpolation is used.</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMP</td>
<td>Use &quot;normal&quot; ramping (as in FastTracker 2).</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMPS</td>
<td>Use &quot;sensitive&quot; ramping.</td>
</tr>
<tr>
<td>BASS_MUSIC_SURROUND</td>
<td>Apply XMPlay's surround sound to the music. This is ignored if the BASS_SAMPLE_MONO flag is also specified.</td>
</tr>
<tr>
<td>BASS_MUSIC_SURROUND2</td>
<td>Apply XMPlay's surround sound mode 2 to the music. This is ignored if the BASS_SAMPLE_MONO flag is also specified.</td>
</tr>
<tr>
<td>BASS_MUSIC_FT2MOD</td>
<td>Play .MOD file as FastTracker 2 would.</td>
</tr>
<tr>
<td>BASS_MUSIC_PT1MOD</td>
<td>Play .MOD file as ProTracker 1 would.</td>
</tr>
<tr>
<td>BASS_MUSIC_POSRESET</td>
<td>Stop all notes when seeking (BASS_ChannelSetPosition).</td>
</tr>
<tr>
<td>BASS_MUSIC_POSRESETEX</td>
<td>Stop all notes and reset bpm/etc when seeking.</td>
</tr>
<tr>
<td>BASS_MUSIC_STOPBACK</td>
<td>Stop the music when a backward jump effect is played. This stops musics that never reach the end from going into endless loops. Some</td>
</tr>
</tbody>
</table>
MOD musics are designed to jump all over the place, so this flag would cause those to be stopped prematurely. If this flag is used together with the BASS_SAMPLE_LOOP flag, then the music would not be stopped but any BASS_SYNC_END sync would be triggered.

**BASS_MUSIC_PRESCAN**

MOD musics are designed to jump all over the place, so this flag would cause those to be stopped prematurely. If this flag is used together with the BASS_SAMPLE_LOOP flag, then the music would not be stopped but any BASS_SYNC_END sync would be triggered.

**BASS_MUSIC_NOSAMPLE**

Do not load the samples. This reduces the time (and memory) taken to load the music, notably with MO3 files, which is useful if you just want to get the text and/or length of the music without playing it.

**BASS_MUSIC_AUTOFREE**

Automatically free the music when playback ends. Note that some musics have infinite loops, so never actually end on their own.

**BASS_MUSIC_DECODE**

Decode/render the sample data, without playing it. Use BASS_ChannelGetData to retrieve decoded sample data. The BASS_SAMPLE_3D, BASS_STREAM_AUTOFREE and SPEAKER flags cannot be used.
together with this flag. The BASS_SAMPLESOFTWARE and BASS_SAMP
LE_FX flags are also ignored.

BASS_SPEAKER_*** Speaker assignment flags. The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.

BASS_UNICODE file is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, and UTF-8 on other platforms.

freq Sample rate to render/play the MOD music at... 0 = the rate specified in the BASS_Init call, 1 = the device's current output rate (or the BASS_Init rate if that is not available).
Return value
If successful, the loaded MOD music'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**: The BASS_MUSIC_AUTOFREE flag is unavailable to decoding channels.
- **BASS_ERROR_FILEOPEN**: The file could not be opened.
- **BASS_ERROR_FILEFORM**: The file's format is not recognised/supported.
- **BASS_ERROR_FORMAT**: The sample format is not supported by the device/drivers. If using the BASS_SAMPLE_FLOAT flag, it could be that floating-point channels are not supported.
- **BASS_ERROR_SPEAKER**: The specified SPEAKER flags are invalid. The device/drivers do not support them 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
BASS uses the same code as XMPlay for its MOD music support, giving an accurate reproduction of the MO3 / IT / XM / S3M / MTM / MOD / UMX formats.

MO3s are treated and used in exactly the same way as normal MOD musics. The advantage of MO3s is that they can be a lot smaller with virtually identical quality. Playing a MO3 does not use any more CPU power than playing the original MOD version does. The only difference is a slightly longer load time as the samples are being decoded. MO3 files are created using the MO3 encoder available at the BASS website.

DMO effects (the same as available with BASS_ChannelSetFX) can be used in IT and XM files (and MO3 versions of them) created with Modplug Tracker. This allows effects to be added to a track without having to resort to an MP3 or OGG version, so it can remain small and still sound fancy. Of course, the effects require some CPU, so should not be used carelessly if performance is key.

"Ramping" does not take a lot of extra processing and improves the sound quality by removing clicks, by ramping/smoothing volume and pan changes. The start of a sample may also be ramped-in. That is always the case with XM files (or MOD files in FT2 mode) when using normal ramping, and possibly with all formats when using sensitive ramping; sensitive ramping will only ramp-in when necessary to avoid a click. Generally, normal ramping is recommended for XM files, and sensitive ramping for the other formats, but some XM files may also sound better using sensitive ramping.

After loading a MOD music from memory (mem = TRUE), the memory can safely be discarded.
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.

DMO effects are not supported in MOD music on Windows CE, and DirectX 8 (or above) is required on Windows. They are always available on other platforms, except for the following: Compressor, Gargle, and I3DL2Reverb. When a DMO effect is unavailable, the MOD music can still be played, but the effect will be disabled.
See also
BASS_ChannelGetInfo, BASS_ChannelGetLength, BASS_ChannelGetTags,
BASS_ChannelPlay, BASS_ChannelSetAttribute, BASS_ChannelSetDSP,
BASS_ChannelSetFX, BASS_ChannelSetLink, BASS_MusicFree,
BASS_CONFIG_MUSIC_VIRTUAL
Frees all resources used by the recording device.

BOOL BASS_RecordFree();
Return value
If successful, then TRUE is returned, else FALSE is returned. Use \texttt{BASS_ErrorGetCode} to get the error code.
Error codes
BASS_ERRORINIT  BASS_RecordInit has not been successfully called.
**Remarks**
This function should be called for all initialized recording devices before your program exits.

When using multiple recording devices, the current thread's device setting (as set with `BASS_RecordSetDevice`) determines which device this function call applies to.
See also
BASS_RecordInit
BASS_RecordGetDevice

Retrieves the recording device setting of the current thread.

DWORD BASS_RecordGetDevice();
Return value
If successful, the device number is returned, else -1 is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_INIT  BASS_RecordInit has not been successfully called; there are no initialized devices.
See also
BASS_ChannelGetDevice, BASS_RecordInit, BASS_RecordSetDevice
BASS_RecordGetDeviceInfo

Retrieves information on a recording device.

```c
BOOL BASS_RecordGetDeviceInfo(
    DWORD device,
    BASS_DEVICEINFO *info
);
```
**Parameters**

**device**  The device to get the information of... 0 = first.

**info**  Pointer to a structure to receive the information.
Return value
If successful, then TRUE is returned, else FALSE is returned. Use
BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_DX  A sufficient version of DirectX is not installed.

BASS_ERRORDEVICE  device is invalid.
Remarks
This function can be used to enumerate the available devices for a setup dialog.
**Platform-specific**
Recording support requires DirectX 5 (or above) on Windows.

On Linux, a "Default" device is hardcoded to device number 0, which uses the default input set in the ALSA config.
Example
Get the total number of devices currently present.

```c
int a, count=0;
BASS_DEVICEINFO info;
for (a=0; BASS_RecordGetDeviceInfo(a, &info); a++)
    if (info.flags&BASS_DEVICE_ENABLED) // device is enabled
        count++; // count it
```

Find a microphone.

```c
int a;
BASS_DEVICEINFO info;
for (a=0; BASS_RecordGetDeviceInfo(a, &info); a++)
    if ((info.flags&BASS_DEVICE_ENABLED) && (info.flags&BASS_DEVICE_TYPE_MASK) == BASS_DEVICE_TYPE_MICROPHONE)
        // do something
```
See also
BASS_RecordGetInfo, BASS_RecordInit, BASS_DEVICEINFO structure
BASS_RecordGetInfo

Retrieves information on the recording device being used.

BOOL BASS_RecordGetInfo(
    BASS_RECORDINFO *info
);

Parameters

info  Pointer to a structure to receive the information.
Return value
If successful, TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
Error codes
BASS_ERROR_INIT  BASS_RecordInit has not been successfully called.
Example
Check if the current device can have multiple inputs enabled.

BASS_RECORDINFO info;
BASS_RecordGetInfo(&info);
if (!info.singlein) {
    // device does allow multiple inputs to be enabled
}
See also
BASS_RecordGetDeviceInfo, BASS_RECORDINFO structure
**BASS_RecordGetInput**

Retrieves the current settings of a recording input source.

```
DWORD BASS_RecordGetInput(
    int input,
    float *volume
);
```
**Parameters**

input  The input to get the settings of... 0 = first, -1 = master.
volume Pointer to a variable to receive the volume... NULL = don't retrieve the volume.
**Return value**

If an error occurs, -1 is returned, use `BASS_ErrorGetCode` to get the error code. If successful, then the settings are returned. The BASS_INPUT_OFF flag will be set if the input is disabled, otherwise the input is enabled. The type of input is also indicated in the high 8 bits (use BASS_INPUT_TYPE_MASK to test) of the return value, and can be one of the following. If the volume is requested but not available, `volume` will receive -1.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_INPUT_TYPE_DIGITAL</td>
<td>Digital input source, for example, a DAT or audio CD.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_LINE</td>
<td>Line-in. On some devices, &quot;Line-in&quot; may be combined with other analog sources into a single BASS_INPUT_TYPE_ANALOG input.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_MIC</td>
<td>Microphone.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_SYNTH</td>
<td>Internal MIDI synthesizer.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_CD</td>
<td>Analog audio CD.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_PHONE</td>
<td>Telephone.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_SPEAKER</td>
<td>PC speaker.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_WAVE</td>
<td>The device's WAVE/PCM output.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_AUX</td>
<td>Auxiliary. Like &quot;Line-in&quot;, &quot;Aux&quot; may be combined with other analog sources into a single BASS_INPUT_TYPE_ANALOG input on some devices.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_ANALOG</td>
<td>Analog, typically a mix of all analog sources.</td>
</tr>
<tr>
<td>BASS_INPUT_TYPE_UNDEF</td>
<td>Anything that is not covered by the other types.</td>
</tr>
</tbody>
</table>
Error codes
BASS_ERROR_INIT  BASS_RecordInit has not been successfully called.
BASS_ERROR_ILLPARAM  input is invalid.
BASS_ERROR_NOTAVAIL  A master input is not available.
BASS_ERROR_UNKNOWN  Some other mystery problem!
Platform-specific
The input type information is only available on Windows. There is no "what you hear" type of input defined; if the device has one, it will typically come under BASS_INPUT_TYPE_ANALOG or BASS_INPUT_TYPE_UNDEF.

On OSX, there is no master input (-1), and only the currently enabled input has its volume setting available (if it has a volume control).
Example
List all available input sources, with their current status.

```c
int n;
char *name;
for (n=0; name=BASS_RecordGetInputName(n); n++) {
    float vol;
    int s=BASS_RecordGetInput(n, &vol);
    printf("%s [%s : %g]\n", name, s&BASS:_INPUT_OFF?"off":"on", vol);
}
```

Find a microphone input.

```c
int mic=-1, n, flags;
for (n=0; (flags=BASS_RecordGetInput(n, NULL))!=-1; n++) {
    if ((flags&BASS:_INPUT_TYPE_MASK)==BASS_INPUT_TYPE_MIC) { // found the mic!
        mic=n;
        break;
    }
}
if (mic!=-1) printf("Found a microphone at input %d\n", mic);
else printf("No microphone found\n");
```
See also
BASS_RecordGetInfo, BASS_RecordGetInputName, BASS_RecordSetInput
BASS_RecordGetInputName

Retrieves the text description of a recording input source.

```c
char *BASS_RecordGetInputName(
    int input
);
```
**Parameters**

*input*  The input to get the description of... 0 = first, -1 = master.
**Return value**
If successful, then a pointer to the description is returned, else NULL is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

BASS_ERROR_INIT  
BASS_RecordInit has not been successfully called.

BASS_ERROR_ILLPARAM  
Input is invalid.

BASS_ERROR_NOTAVAIL  
A master input is not available.
**Platform-specific**
The returned string is in ANSI or UTF-8 form on Windows, depending on the `BASS_CONFIG_UNICODE` setting. It is in UTF-16 form ("WCHAR" rather than "char") on Windows CE, and in UTF-8 form on other platforms.

There is no master input (-1) on OSX.
See also
BASS_RecordGetInfo, BASS_RecordGetInput, BASS_RecordSetInput
BASS_RecordInit

Initializes a recording device.

```c
BOOL BASS_RecordInit(
    int device
);
```
Parameters

device  The device to use... -1 = default device, 0 = first.

BASS_RecordGetDeviceInfo can be used to enumerate the available devices.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
### Error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_DX</td>
<td>A sufficient version of DirectX (or ALSA) is not installed.</td>
</tr>
<tr>
<td>BASS_ERROR_DEVICE</td>
<td><code>device</code> is invalid.</td>
</tr>
<tr>
<td>BASS_ERROR_ALREADY</td>
<td>The device has already been initialized.</td>
</tr>
<tr>
<td></td>
<td><code>BASS_RecordFree</code> must be called before it can be initialized again.</td>
</tr>
<tr>
<td>BASS_ERROR_DRIVER</td>
<td>There is no available device driver.</td>
</tr>
</tbody>
</table>
Remarks
This function must be successfully called before using the recording features.

Simultaneously using multiple devices is supported in the BASS API via a context switching system; instead of there being an extra "device" parameter in the function calls, the device to be used is set prior to calling the functions. BASS_RecordSetDevice is used to switch the current recording device. When successful, BASS_RecordInit automatically sets the current thread's device to the one that was just initialized.

When using the default device (device = -1), BASS_RecordGetDevice can be used to find out which device it was mapped to.
**Platform-specific**
Recording support requires DirectX 5 (or above) on Windows.

On Linux, a "Default" device is hardcoded to device number 0, which uses the default input set in the ALSA config; that could map directly to one of the other devices or it could use ALSA plugins.
See also
BASS_RecordFree, BASS_RecordGetInfo, BASS_RecordGetInput,
BASS_RecordSetDevice, BASS_RecordSetInput, BASS_RecordStart
BASS_RecordSetDevice

Sets the recording device to use for subsequent calls in the current thread.

```c
BOOL BASS_RecordSetDevice(
    DWORD device
);
```
Parameters

device   The device to use... 0 = first.
**Return value**
If successful, then TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

BASS_ERROR_DEVICE  *device* is invalid.
BASS_ERROR_INIT    The device has not been initialized.
Remarks
Simultaneously using multiple devices is supported in the BASS API via a context switching system; instead of there being an extra "device" parameter in the function calls, the device to be used is set prior to calling the functions. The device setting is local to the current thread, so calling functions with different devices simultaneously in multiple threads is not a problem.

The functions that use the recording device selection are the following: `BASS_RecordFree`, `BASS_RecordGetInfo`, `BASS_RecordGetInput`, `BASS_RecordGetInputName`, `BASS_RecordSetInput`, `BASS_RecordStart`.

When one of the above functions (or `BASS_RecordGetDevice`) is called, BASS will check the current thread's recording device setting, and if no device is selected (or the selected device is not initialized), BASS will automatically select the lowest device that is initialized. This means that when using a single device, there is no need to use this function; BASS will automatically use the device that's initialized. Even if you free the device, and initialize another, BASS will automatically switch to the one that is initialized.
Example
Start recording on device 2.

BASS_RecordSetDevice(2); // select device 2
record = BASS_RecordStart(44100, 2, 0, MyRecordProc, 0); // start recording
See also
BASS_ChannelGetDevice, BASS_RecordGetDevice, BASS_RecordInit
BASS_RecordSetInput

Adjusts the settings of a recording input source.

```c
BOOL BASS_RecordSetInput(
    int input,
    DWORD flags,
    float volume
);
```
**Parameters**

- **input**  The input to adjust the settings of... 0 = first, -1 = master.
- **flags**  The new setting... a combination of these flags.
  - **BASS_INPUT_OFF**  Disable the input. This flag cannot be used when the device supports only one input at a time.
  - **BASS_INPUT_ON**  Enable the input. If the device only allows one input at a time, then any previously enabled input will be disabled by this.
- **volume**  The volume level... 0 (silent) to 1 (max), less than 0 = leave current.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use **BASS_ErrorGetCode** to get the error code.
**Error codes**

BASS_ERROR_INIT  
* BASS_RecordInit* has not been successfully called.

BASS_ERROR_ILLPARAM  
*input* or *volume* is invalid.

BASS_ERROR_NOTAVAIL  
The input does not have the necessary controls to apply the *flags* and/or *volume*. If attempting to set both at the same time, try separating them to determine which is unavailable.

BASS_ERROR_UNKNOWN  
Some other mystery problem!
Remarks
The actual volume level may not be exactly the same as requested, due to underlying precision differences. BASS_RecordGetInput can be used to confirm what the volume is.

The volume curve used by this function is always linear; the BASS_CONFIG_CURVE_VOL config option setting has no effect on this.

Changes made by this function are system-wide, i.e. other software using the device will be affected by it.
Platform-specific
On OSX, there is no master input (-1), and only the currently enabled input may have its volume set (if it has a volume control).
**Example**
Enable the first input, and set its volume level to 50%.

```c
BASS_RecordSetInput(0, BASS_INPUT_ON, 0.5);
```
See also
BASS_RecordGetInfo, BASS_RecordGetInput
BASS_RecordStart

Starts recording.

HRECORD BASS_RecordStart(
    DWORD freq,
    DWORD chans,
    DWORD flags,
    RECORDPROC *proc
    void *user
);
**Parameters**

freq  The sample rate to record at.

chans The number of channels... 1 = mono, 2 = stereo, etc.

flags A combination of these flags.

- **BASS_SAMPLE_8BITS** Use 8-bit resolution. If neither this or the BASS_SAMPLE_FLOAT flag are specified, then the recorded data is 16-bit.

- **BASS_SAMPLE_FLOAT** Use 32-bit floating-point sample data. See [Floating-point channels](#) for info.

- **BASS_RECORD_PAUSE** Start the recording paused. Use [BASS_ChannelPlay](#) to resume it.

The HIWORD - use MAKELONG(flags,period) - can be used to set the period (in milliseconds) between calls to the callback function. The minimum period is 5ms, the maximum is half the BASS_CONFIG_REC_BUFFER setting. If the period specified is outside this range, it is automatically capped. The default is 100ms.

proc The user defined function to receive the recorded sample data... can be NULL if you do not wish to use a callback.

user User instance data to pass to the callback function.
**Return value**
If successful, the new recording's handle is returned, else FALSE is returned. Use **BASS_ErrorGetCode** to get the error code.
### Error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_INIT</td>
<td>BASS_RecordInit has not been successfully called.</td>
</tr>
<tr>
<td>BASS_ERROR_BUSY</td>
<td>The device is busy. An existing recording may need to be stopped before starting another one.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>The recording device is not available. Another application may already be recording with it, or it could be a half-duplex device that is currently being used for playback.</td>
</tr>
<tr>
<td>BASS_ERROR_FORMAT</td>
<td>The requested format is not supported. If using the BASS_SAMPLE_FLOAT flag, it could be that floating-point recording is not supported.</td>
</tr>
<tr>
<td>BASS_ERROR_MEM</td>
<td>There is insufficient memory.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
Use \texttt{BASS\_ChannelStop} to stop the recording and free its resources. \texttt{BASS\_ChannelPause} can be used to pause the recording; it can also be started in a paused state via the \texttt{BASS\_RECORD\_PAUSE} flag, which allows DSP/FX to be set on it before any data reaches the callback function.

The sample data will generally arrive from the recording device in blocks rather than in a continuous stream, so when specifying a very short period between callbacks, some calls may be skipped due to there being no new data available since the last call.

When not using a callback (\texttt{proc = NULL}), the recorded data is instead retrieved via \texttt{BASS\_ChannelGetData}. To keep latency at a minimum, the amount of data in the recording buffer should be monitored (also done via \texttt{BASS\_ChannelGetData}, with the \texttt{BASS\_DATA\_AVAILABLE} flag) to check that there is not too much data; freshly recorded data will only be retrieved after the older data in the buffer is.
**Platform-specific**

Multiple simultaneous recordings can be made from the same device on Windows XP and later, but generally not on older Windows. Multiple simultaneous recordings are possible on iOS and OSX, but may not always be on Linux or Windows CE.

On OSX and iOS, the device is instructed (when possible) to deliver data at the period set in the HIWORD of flags, even when a callback function is not used. On other platforms, it is up the system when data arrives from the device.
**Example**
Start recording at 44100 Hz stereo 16-bit.

```c
HRECORD record=BASS_RecordStart(44100, 2, 0, MyRecordProc, 0);
```
See also
BASS_ChannelGetData, BASS_ChannelGetLevel, BASS_ChannelPause,
BASS_ChannelStop, BASS_RecordInit, RECORDPROC callback,
BASS_CONFIG_REC_BUFFER
User defined callback function to process recorded sample data.

```c
BOOL CALLBACK RecordProc(
    HRECORD handle,
    const void *buffer,
    DWORD length,
    void *user
);
```
**Parameters**

handle  The recording that the data is from.

buffer  Pointer to the recorded sample data. The sample data is in standard Windows PCM format, that is 8-bit samples are unsigned, 16-bit samples are signed, 32-bit floating-point samples range from -1 to +1.

length  The number of bytes in the buffer.

user  The user instance data given when BASS_RecordStart was called.
**Return value**
Return FALSE to stop recording, and anything else to continue recording.
Remarks

*BASS_RecordFree* should not be used to free the recording device within a recording callback function. Nor should *BASS_ChannelStop* be used to stop the recording; return FALSE to do that instead.
Example
A callback function to write the recorded data to disk.

```c
BOOL CALLBACK MyRecordProc(HRECORD handle, const void *buffer, DWORD length, void *user)
{
    fwrite(buffer, 1, length, (FILE*)user); // write the buffer to file
    return TRUE; // continue recording
}

HRECORD record=BASS_RecordStart(44100, 2, 0, MyRecordProc, file); // start recording
```
See also
BASS_RecordStart
BASS_RECORDINFO structure

Used with BASS_RecordGetInfo to retrieve information on the current recording device.

typedef struct {
    DWORD flags;
    DWORD formats;
    DWORD inputs;
    BOOL singlein;
    DWORD freq;
} BASS_RECORDINFO;
Members

flags The device's capabilities... a combination of these flags.
- DSCCAPS_EMULDRIVER The device's drivers do NOT have DirectSound recording support, so it is being emulated.
- DSCCAPS_CERTIFIED The device driver has been certified by Microsoft.

formats The standard formats supported by the device... a combination of these flags.
- WAVE_FORMAT_1M08 11025 Hz, Mono, 8-bit
- WAVE_FORMAT_1S08 11025 Hz, Stereo, 8-bit
- WAVE_FORMAT_1M16 11025 Hz, Mono, 16-bit
- WAVE_FORMAT_1S16 11025 Hz, Stereo, 16-bit
- WAVE_FORMAT_2M08 22050 Hz, Mono, 8-bit
- WAVE_FORMAT_2S08 22050 Hz, Stereo, 8-bit
- WAVE_FORMAT_2M16 22050 Hz, Mono, 16-bit
- WAVE_FORMAT_2S16 22050 Hz, Stereo, 16-bit
- WAVE_FORMAT_4M08 44100 Hz, Mono, 8-bit
- WAVE_FORMAT_4S08 44100 Hz, Stereo, 8-bit
- WAVE_FORMAT_4M16 44100 Hz, Mono, 16-bit
- WAVE_FORMAT_4S16 44100 Hz, Stereo, 16-bit
The number of channels supported by the device is in the high 8 bits.

inputs The number of input sources available to the device.
singlein TRUE = only one input may be active at a time.

freq The device's current input sample rate. Recording at this rate will give the best quality and performance, as no resampling is required.
**Platform-specific**
The *flags* member is only used on Windows. The *formats* member is only used on Windows/OSX/iOS, and only for the device's channel count in the case of OSX and iOS. On Windows, it does not necessarily represent all of the formats supported by the device, just the "standard" ones. *freq* is also only available on Windows/OSX/iOS, but not on Windows prior to Vista.
See also
BASS_RecordGetInfo
A "channel" can be a sample playback channel (HCHANNEL), a sample stream (HSTREAM), a MOD music (HMUSIC), or a recording (HRECORD). Each "Channel" function can be used with one or more of these channel types.
The types of channel

HCHANNEL  Returned by BASS_SampleGetChannel.

HSTREAM  Returned by BASS_StreamCreate, BASS_StreamCreateFile, BASS_StreamCreateURL, BASS_StreamCreateFileUser. Also add-on provided functions.

HMUSIC  Returned by BASS_MusicLoad.

HRECORD  Returned by BASS_RecordStart.

A sample stream (HSTREAM) or MOD music (HMUSIC) that has been created with the BASS_STREAM_DECODE or BASS_MUSIC_DECODE flag is sometimes referred to as a "decoding channel".
BASS_ChannelBytes2Seconds

Translates a byte position into time (seconds), based on a channel's format.

double BASS_ChannelBytes2Seconds(
    DWORD handle,
    QWORD pos
);
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD. HSAMPLE handles may also be used.

**pos**  The position to translate.
**Return value**
If successful, then the translated length is returned, else a negative value is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**
BASS_ERROR_HANDLE  *handle* is not a valid channel.
Remarks
The translation is based on the channel's initial sample rate, when it was created.
See also
BASS_ChannelGetInfo, BASS_ChannelGetPosition,
BASS_ChannelSeconds2Bytes
BASS_ChannelFlags

Modifies and retrieves a channel's flags.

```c
DWORD BASS_ChannelFlags(
    DWORD handle,
    DWORD flags,
    DWORD mask
);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The channel handle... a HCHANNEL, HMUSIC, HSTREAM.</td>
</tr>
<tr>
<td>flags</td>
<td>A combination of these flags.</td>
</tr>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>Loop the channel.</td>
</tr>
<tr>
<td>BASS_STREAM_AUTOFREE</td>
<td>Automatically free the channel when playback ends. Note that the BASS_MUSIC_AUTOFREE flag is identical to this flag. (HSTREAM/HMUSIC only)</td>
</tr>
<tr>
<td>BASS_STREAM_RESTRATE</td>
<td>Restrict the download rate.                                                (HSTREAM)</td>
</tr>
<tr>
<td>BASS_MUSIC_NONINTER</td>
<td>Use non-interpolated sample mixing. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_SINCINTER</td>
<td>Use sinc interpolated sample mixing. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMP</td>
<td>Use &quot;normal&quot; ramping. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMPS</td>
<td>Use &quot;sensitive&quot; ramping. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_SURROUND</td>
<td>Use surround sound. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_SURROUND2</td>
<td>Use surround sound mode 2. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_FT2MOD</td>
<td>Use FastTracker 2 .MOD playback. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_PT1MOD</td>
<td>Use ProTracker 1 .MOD playback. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_POSRESET</td>
<td>Stop all notes when seeking.                                               (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_POSRESETEX</td>
<td>Stop all notes and reset BPM/etc when seeking. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_STOPBACK</td>
<td>Stop when a backward jump effect is played. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_SPEAKER_***</td>
<td><strong>Speaker assignment flags.</strong> (HSTREAM/HMUSIC)</td>
</tr>
</tbody>
</table>

Other flags may be supported by add-ons, see the documentation.
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   handle is not a valid channel.
**Remarks**

Some flags may not be adjustable in some circumstances, so the return value should be checked to confirm any changes. The flags listed above are just the flags that can be modified, and there may be additional flags present in the return value. See the `BASS_CHANNELINFO` documentation for a full list of flags.

Streams that are created by add-ons may have additional flags available. There is a limited number of possible flag values though, so some add-ons may use the same flag value for different things. This means that when using add-on specific flags with a stream created via the plugin system, it is a good idea to first confirm that the add-on is handling the stream, by checking its `ctype` via `BASS_ChannelGetInfo`.

During playback, the effects of flag changes are not heard instantaneously, due to buffering. To reduce the delay, use the `BASS_CONFIG_BUFFER` config option to reduce the buffer length.
Example
Toggle looping on a channel.

```c
if (BASS_ChannelFlags(channel, 0, 0)&BASS_SAMPLE_LOOP) {
    // looping is enabled, so...
    BASS_ChannelFlags(channel, 0, BASS_SAMPLE_LOOP);
} else {
    // looping is disabled, so...
    BASS_ChannelFlags(channel, BASS_SAMPLE_LOOP, BASS_SAMPLE_LOOP);
}
```
See also
BASS_ChannelGetAttribute, BASS_ChannelGetInfo,
BASS_ChannelSetAttribute, BASS_MusicLoad
BASS_ChannelGet3DAttributes

Retrieves the 3D attributes of a sample, stream, or MOD music channel with 3D functionality.

```c
BOOL BASS_ChannelGet3DAttributes(
    DWORD handle,
    DWORD *mode,
    float *min,
    float *max,
    DWORD *iangle,
    DWORD *oangle,
    float *outvol
);
```
**Parameters**

- **handle**: The channel handle... a HCHANNEL, HMUSIC, HSTREAM.
- **mode**: The 3D processing mode... NULL = don't retrieve it.
- **min**: The minimum distance... NULL = don't retrieve it.
- **max**: The maximum distance... NULL = don't retrieve it.
- **iangle**: The angle of the inside projection cone... NULL = don't retrieve it.
- **oangle**: The angle of the outside projection cone... NULL = don't retrieve it.
- **outvol**: The delta-volume outside the outer projection cone... NULL = don't retrieve it.
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  *handle* is not a valid channel.

BASS_ERROR_NO3D  The channel does not have 3D functionality.
Remarks
The "iangle and oangle parameters must both be retrieved in a single call to this function; one cannot be retrieved without the other. See BASS_ChannelSet3DAttributes for an explanation of the parameters.
See also
BASS_ChannelGet3DPosition, BASS_ChannelGetAttribute,
BASS_ChannelSet3DAttributes, BASS_ATTRIB_EAXMIX
BASS_ChannelGet3DPosition

Retrieves the 3D position of a sample, stream, or MOD music channel with 3D functionality.

```c
BOOL BASS_ChannelGet3DPosition(
    DWORD handle,
    BASS_3DVECTOR *pos,
    BASS_3DVECTOR *orient,
    BASS_3DVECTOR *vel
);
```
Parameters

handle   The channel handle... a HCHANNEL, HMUSIC, HSTREAM.
pos     Position of the sound... NULL = don't retrieve it.
orient   Orientation of the sound... NULL = don't retrieve it.
vel      Velocity of the sound... NULL = don't retrieve it.
**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  *handle* is not a valid channel.

BASS_ERROR_NO3D  The channel does not have 3D functionality.
See also
BASS_ChannelGet3DAttributes, BASS_ChannelGetAttribute,
BASS_ChannelSet3DPosition, BASS_Get3DFactors, BASS_Get3DPosition,
BASS_3DVECTOR structure
BASS_ChannelGetAttribute

Retrieves the value of a channel's attribute.

BOOL BASS_ChannelGetAttribute(
    DWORD handle,
    DWORD attrib,
    float *value
);
**Parameters**

**handle** The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HREC

**attrib** The attribute to get the value of... one of the following.

- BASS_ATTRIB_EAXMIX EAX wet/dry mix. (HCHANNEL/HMUSIC/STREAM only)
- BASS_ATTRIB_CPU CPU usage. (HMUSIC/HS)
- BASS_ATTRIB_FREQ Sample rate.
- BASS_ATTRIB_MUSIC_ACTIVE Active channel count. (HMUSIC)
- BASS_ATTRIB_MUSIC_AMPLIFY Amplification level. (HMUSIC)
- BASS_ATTRIB_MUSIC_BPM BPM. (HMUSIC)
- BASS_ATTRIB_MUSIC_PANSEP Pan separation level. (HMUSIC)
- BASS_ATTRIB_MUSIC_PSCALER Position scaler. (HMUSIC)
- BASS_ATTRIB_MUSIC_SPEED Speed. (HMUSIC)
- BASS_ATTRIB_MUSIC_VOL_CHAN A channel volume level. (HMUSIC)
- BASS_ATTRIB_MUSIC_VOL_GLOBAL Global volume level. (HMUSIC)
- BASS_ATTRIB_MUSIC_VOL_INST An instrument/sample volume level. (HMUSIC)
- BASS_ATTRIB_NET_RESUME Buffer level to resume stalled playback. (HSTREAM)
- BASS_ATTRIB_NOBUFFER Playback buffering switch. (HMUSIC/HSTREAM)
- BASS_ATTRIB_PAN Panning/balance position.
- BASS_ATTRIB_SRC Sample rate conversion quality.
- BASS_ATTRIB_VOL Volume level.

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

**value** Pointer to a variable to receive the attribute value.
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  handle is not a valid channel.
BASS_ERROR_NOTAVAIL  The attribute is not available.
BASS_ERROR_ILLTYPE  attrib is not valid.

some attributes may have additional error codes, see the documentation.
See also
BASS_ChannelGet3DAttributes, BASS_ChannelGetAttributeEx,
BASS_ChannelGetInfo, BASS_ChannelSetAttribute
BASS_ChannelGetAttributeEx

Retrieves the value of a channel's attribute.

```c
DWORD BASS_ChannelGetAttributeEx(
    DWORD handle,
    DWORD attrib,
    void *value,
    DWORD size
);
```
**Parameters**

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

**attrib** The attribute to get the value of... one of the following.

- **BASS_ATTRIB_SCANINFO** Scanned info. (HSTREAM only)

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

**value** Pointer to a buffer to receive the attribute data.

**size** The size of the attribute data... 0 = get the size of the attribute without getting the data.
Return value
If successful, the size of the attribute data is returned, else 0 is returned. Use BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_HANDLE   *handle* is not a valid channel.
BASS_ERROR_NOTAVAIL The attribute is not available.
BASS_ERROR_ILLTYPE   *attrib* is not valid.
BASS_ERROR_ILLPARAM  *size* is not valid.

some attributes may have additional error codes, see the documentation.
Remarks
This function also supports the floating-point attributes supported by
BASS_ChannelGetAttribute.
See also
BASS_ChannelGetAttribute, BASS_ChannelGetInfo,
BASS_ChannelSetAttributeEx
BASS_ChannelGetData

Retrieves the immediate sample data (or an FFT representation of it) of a sample channel, stream, MOD music, or recording channel.

```c
DWORD BASS_ChannelGetData(
    DWORD handle,
    void *buffer,
    DWORD length
);
```
**Parameters**

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

**buffer** Pointer to a buffer to receive the data... can be NULL when handle is a recording channel (HRECORD), to discard the requested amount of data from the recording buffer.

**length** Number of bytes wanted (up to 268435455 or 0xFFFFFFFF), and/or the following flags.

- **BASS_DATA_FLOAT** Return floating-point sample data.
- **BASS_DATA_FIXED** Return 8.24 fixed-point data.
- **BASS_DATA_FFT256** 256 sample FFT (returns 128 values).
- **BASS_DATA_FFT512** 512 sample FFT (returns 256 values).
- **BASS_DATA_FFT1024** 1024 sample FFT (returns 512 values).
- **BASS_DATA_FFT2048** 2048 sample FFT (returns 1024 values).
- **BASS_DATA_FFT4096** 4096 sample FFT (returns 2048 values).
- **BASS_DATA_FFT8192** 8192 sample FFT (returns 4096 values).
- **BASS_DATA_FFT16384** 16384 sample FFT (returns 8192 values).
- **BASS_DATA_FFT_COMPLEX** Return the complex FFT result rather than the magnitudes. This increases the amount of data returned (as listed above) fourfold, as it returns real and imaginary parts and the full FFT result (not only the first half). The real and imaginary parts are interleaved in the returned data.
- **BASS_DATA_FFT_INDIVIDUAL** Perform a separate FFT for each
channel, rather than a single combined FFT. The size of the data returned (as listed above) is multiplied by the number of channels.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_DATA_FFT_NOWINDOW</td>
<td>Prevent a Hann window being applied to the sample data when performing an FFT.</td>
</tr>
<tr>
<td>BASS_DATA_FFT_REMOVEDC</td>
<td>Remove any DC bias from the sample data when performing an FFT.</td>
</tr>
<tr>
<td>BASS_DATAAVAILABLE</td>
<td>Query the amount of data the channel has buffered for playback, or from recording. This flag cannot be used with decoding channels as they do not have playback buffers. buffer is ignored when using this flag.</td>
</tr>
</tbody>
</table>
**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 or BASS_DATA_FIXED flag). When using the BASS_DATAAVAILABLE flag, the number of bytes in the channel's buffer is returned.
**Error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_HANDLE</td>
<td><code>handle</code> is not a valid channel.</td>
</tr>
<tr>
<td>BASS_ERROR_ENDED</td>
<td>The channel has reached the end.</td>
</tr>
<tr>
<td>BASS_ERROR_NOTAVAIL</td>
<td>The BASS_DATAAVAILABLE flag was used with a decoding channel.</td>
</tr>
<tr>
<td>BASS_ERROR_BUFLOST</td>
<td>Should not happen... check that a valid window handle was used with BASS_Init.</td>
</tr>
</tbody>
</table>
Remarks
Unless the channel is a decoding channel, this function can only return as much data as has been written to the channel's playback buffer, so it may not always be possible to get the amount of data requested, especially if it is a large amount. If large amounts are needed, the buffer length (BASS_CONFIG_BUFFER) can be increased. The BASS_DATA_AVAILABLE flag can be used to check how much data a channel's buffer contains at any time, including when stopped or stalled.

When requesting data from a decoding channel, data is decoded directly from the channel's source (no playback buffer) and as much data as the channel has available can be decoded at a time.

When retrieving sample data, 8-bit samples are unsigned (0 to 255), 16-bit samples are signed (-32768 to 32767), 32-bit floating-point samples range from -1 to +1 (not clipped, so can actually be outside this range). That is unless the BASS_DATA_FLOAT flag is used, in which case the sample data will be converted to 32-bit floating-point (if it is not already), or if the BASS_DATA_FIXED flag is used, in which case the data will be converted to 8.24 fixed-point.

Unless complex data is requested via the BASS_DATA_FFT_COMPLEX flag, the magnitudes of the first half of an FFT result are returned. For example, with a 2048 sample FFT, there will be 1024 floating-point values returned. If the BASS_DATA_FIXED flag is used, then the FFT values will be in 8.24 fixed-point form rather than floating-point. Each value, or "bin", ranges from 0 to 1 (can actually go higher if the sample data is floating-point and not clipped). The 1st bin contains the DC component, the 2nd contains the amplitude at 1/2048 of the channel's sample rate, followed by the amplitude at 2/2048, 3/2048, etc. A Hann window is applied to the sample data to reduce leakage, unless the BASS_DATA_FFT_NOWINDOW flag is used. When a window is applied, it causes the DC component to leak into the next bin, but that can be removed (reduced to 0) by using the BASS_DATA_FFT_REMOVEDC flag. Doing so slightly increases the processing required though, so it should only be done when needed, which is when a window is applied and the 2nd bin value is important.

Channels that have 2 or more sample channels (ie. stereo or above) may have FFT performed on each individual channel, using the BASS_DATA_FFT_INDIVIDUAL flag. Without this flag, all of the channels
are combined, and a single mono FFT is performed. Performing the extra individual FFTs of course increases the amount of processing required. The return values are interleaved in the same order as the channel's sample data, eg. stereo = left,right,left,etc.

This function is most useful if you wish to visualize (eg. spectrum analyze) the sound.
Platform-specific
The BASS_DATA_FIXED flag is only available on Android and Windows CE.
**Example**

Perform a 1024 sample FFT on a channel and list the result.

```c
float fft[512]; // fft data buffer
BASS_ChannelGetData(channel, fft, BASS_DATA_FFT1024);
for (int a=0; a<512; a++)
    printf("%d: %f\n", a, fft[a]);
```

Perform a 1024 sample FFT on a channel and list the complex result.

```c
float fft[2048]; // fft data buffer
BASS_ChannelGetData(channel, fft, BASS_DATA_FFT1024|BASS_DATA_FFT_COMPLEX);
for (int a=0; a<1024; a++)
    printf("%d: (%f, %f)\n", a, fft[a*2], fft[a*2+1]);
```
See also
BASS_ChannelGetLevel, BASS_ChannelIsActive
BASS_ChannelGetDevice

Retrieves the device that a channel is using.

```
DWORD BASS_ChannelGetDevice(
    DWORD handle
);
```
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD. HSAMPLE handles may also be used.
**Return value**
If successful, the device number is returned, else -1 is returned. Use `BASS_ErrorGetCode` to get the error code.
Error codes
BASS_ERROR_HANDLE  \textit{handle} is not a valid channel.
Remarks
Recording devices are indicated by the HIWORD of the return value being 1, when this function is called with a HRECORD channel.
See also
BASS_ChannelSetDevice, BASS_GetDevice, BASS_SetDevice,
BASS_RecordGetDevice, BASS_RecordSetDevice
BASS_ChannelGetInfo

Retrieves information on a channel.

```c
BOOL BASS_ChannelGetInfo(
    DWORD handle,
    BASS_CHANNELINFO *info
);
```
**Parameters**

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

info    Pointer to structure to receive the channel information.
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 valid.
See also
BASS_ChannelFlags, BASS_ChannelGetAttribute, BASS_ChannelGetTags, BASS_CHANNELINFO structure
BASS_ChannelGetLength

Retrieves the playback length of a channel.

QWORD BASS_ChannelGetLength(
    DWORD handle,
    DWORD mode
);
### Parameters

| **handle** | The channel handle... a HCHANNEL, HMUSIC, HSTREAM. HSAMPLE handles may also be used. |
| **mode**   | How to retrieve the length. One of the following. |
|           | BASS_POS_BYTE  | Get the length in bytes. |
|           | BASS_POS_MUSIC_ORDER | Get the length in orders. (HMUSIC only) |
|           | BASS_POS_OGG  | Get the number of bitstreams in an OGG file. |

*other modes may be supported by add-ons, see the documentation.*
**Return value**
If successful, then the channel's length is returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.
**Error codes**

BASS_ERROR_HANDLE  
*handle* is not valid.

BASS_ERROR_NOTAVAIL  
The requested length is not available.
Remarks
The exact length of a stream will be returned once the whole file has been streamed, but until then it is not always possible to 100% accurately estimate the length. The length is always exact for MP3/MP2/MP1 files when the BASS_STREAM_PRESCAN flag is used in the BASS_StreamCreateFile call, otherwise it is an (usually accurate) estimation based on the file size. The length returned for OGG files will usually be exact (assuming the file is not corrupt), but when streaming from the internet (or "buffered" user file), it can be a very rough estimation until the whole file has been downloaded. It will also be an estimate for chained OGG files that are not pre-scanned.

Unless an OGG file contains a single bitstream, the number of bitstreams it contains will only be available if it was pre-scanned at the stream's creation.

Retrieving the byte length of a MOD music requires that the BASS_MUSIC_PRESCAN flag was used in the BASS_MusicLoad call.
**Example**
Get the duration (in seconds) of a channel.

```c
QWORD len=BASS_ChannelGetLength(channel, BASS_POS_BYTE); // the length in bytes
double time=BASS_ChannelBytes2Seconds(channel, len); // the length in seconds
```
See also
BASS_ChannelBytes2Seconds, BASS_ChannelGetPosition,
BASS_ChannelSetPosition, BASS_StreamGetPosition
BASS_ChannelGetLevel

Retrieves the level (peak amplitude) of a sample, stream, MOD music, or recording channel.

```
DWORD BASS_ChannelGetLevel(
    DWORD handle
);
```
Parameters

handle  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD.
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  handle is not a valid channel.
BASS_ERROR_NOPLAY  The channel is not playing.
BASS_ERROR_ENDED   The decoding channel has reached the end.
BASS_ERROR_BUFLOST  Should not happen... check that a valid window handle was used with BASS_Init.
Remarks
This function measures the level of the channel's sample data, not its level in the final output mix, so the channel's volume (BASS_ATTRIB_VOL attribute) and panning (BASS_ATTRIB_PAN) does not affect it. The effect of any DSP/FX set on the channel is present in the measurement, except for DX8 effects when using the "With FX flag" DX8 effect implementation.

For channels that are more than stereo, the left level will include all left channels (eg. front-left, rear-left, center), and the right will include all right (front-right, rear-right, LFE). If there are an odd number of channels then the left and right levels will include all channels. If the level of each individual channel is required, that is available from BASS_ChannelGetLevelEx.

20ms of data is inspected to calculate the level. When used with a decoding channel, that means 20ms of data needs to be decoded from the channel in order to calculate the level, and that data is then gone, eg. it is not available to a subsequent BASS_ChannelGetData call.

More flexible level retrieval is available with BASS_ChannelGetLevelEx.
**Example**
Get the left and right levels of a stereo channel.

```c
DWORD level, left, right;
level=BASS_ChannelGetLevel(channel);
left=LOWORD(level); // the left level
right=HIWORD(level); // the right level
```
See also
BASS_ChannelGetData, BASS_ChannelGetLevelEx, BASS_ChannelIsActive
BASS_ChannelGetLevelEx

Retrieves the level of a sample, stream, MOD music, or recording channel.

```c
BOOL BASS_ChannelGetLevelEx(
    DWORD handle,
    float *levels,
    float length,
    DWORD flags
);
```
**Parameters**

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

**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 channel's playback buffer 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.

- **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 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 valid channel.

BASS_ERROR_ILLPARAM  
length is not valid.

BASS_ERROR_NOPLAY  
The channel is not playing.

BASS_ERROR_ENDED  
The decoding channel has reached the end.

BASS_ERROR_BUFLOST  
Should not happen... check that a valid window handle was used with BASS_Init.
Remarks
This function operates in the same way as BASS_ChannelGetLevel but has greater flexibility on how the level is measured. The levels are not clipped, so may exceed +/-1.0 on floating-point channels.
**Example**

Replicate `BASS_ChannelGetLevel` but with floating-point levels.

```c
float levels[2];
BASS_ChannelGetLevelEx(handle, levels, 0.02, BASS_LEVEL_STEREO);
```

Get a mono RMS level reading in decibels using 50ms of data.

```c
float level;
BASS_ChannelGetLevelEx(handle, &level, 0.05, BASS_LEVEL_MONO|BASS_LEVEL_RMS);
float dblevel=(level>0?20*log10(level):-1000); // translate it to dB
```

Get a peak level reading for each channel using 20ms of data.

```c
BASS_CHANNELINFO ci;
BASS_ChannelGetInfo(handle, &ci);
float *levels=(float*)malloc(ci.chans*sizeof(float)); // allocate an array for each channel's level
BASS_ChannelGetLevelEx(handle, levels, 0.02, 0); // get the levels
```
See also
BASS_ChannelGetData, BASS_ChannelGetLevel, BASS_ChannelIsActive
BASS_ChannelGetPosition

Retrieves the playback position of a sample, stream, or MOD music. Can also be used with a recording channel.

QWORD BASS_ChannelGetPosition(
    DWORD handle,
    DWORD mode
);

Parameters

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

**mode**  How to retrieve the position. One of the following, with optional flags.

- **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)*

- **BASS_POS_DECODE**  Flag: Get the decoding/rendering position, which may be ahead of the playback position due to buffering. This flag is unnecessary with decoding channels because the decoding position will always be given for them anyway, as they do not have playback buffers.

*other modes & flags 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 a valid channel.
BASS_ERROR_NOTAVAIL  The requested position is not available.
BASS_ERROR_UNKNOWN   Some other mystery problem!
See also
BASS_ChannelBytes2Seconds, BASS_ChannelGetLength,
BASS_ChannelIsActive, BASS_ChannelGetPosition, BASS_ChannelSetSync,
BASS_StreamGetPosition

BASS_ChannelGetTags

Retrieves tags/headers from a channel.

```c
char *BASS_ChannelGetTags(
    DWORD handle,
    DWORD tags
);
```
**Parameters**

**handle**  
The channel handle... a HMUSIC or HSTREAM.

**tags**  
The tags/headers wanted... one of the following.

- **BASS_TAG_APE**  
  APE (v1 or v2) tags. A pointer to a series of null-terminated UTF-8 strings is returned, the final string ending with a double null. Each string is in the form of "key=value", or "key=value1/value2/..." if there are multiple values.

- **BASS_TAG_APE_BINARY**  
  APE binary tag. A pointer to a TAP_APE_BINARY structure is returned.

  + **tag number (0=first)**

- **BASS_TAG_CA_CODEC**  
  CoreAudio codec information. A pointer to a TAG_CA_CODEC structure is returned.

- **BASS_TAG_HTTP**  
  HTTP headers, only available when streaming from a HTTP server. A pointer to a series of null-terminated strings is returned, the final string ending with a double null.

- **BASS_TAG_ICY**  
  ICY (Shoutcast) tags. A pointer to a series of null-terminated strings is returned, the final string ending with a double null.

- **BASS_TAG_ID3**  
  ID3v1 tags. A pointer to a TAG_ID3 structure is returned.

- **BASS_TAG_ID3V2**  
  ID3v2 tags. A pointer to a variable length block is returned. ID3v2 tags are supported at both the start and end of the file, and in designated RIFF/AIFF chunks. See [www.id3.org](http://www.id3.org) for details of the
<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_TAG_LYRICS3</td>
<td>Lyrics3v2 tag. A single string is returned, containing the Lyrics3v2 information. See <a href="http://www.id3.org/Lyrics3v2">www.id3.org/Lyrics3v2</a> for details of its format.</td>
</tr>
<tr>
<td>BASS_TAG_META</td>
<td>Shoutcast metadata. A single string is returned, containing the current stream title and url (usually omitted). The format of the string is: StreamTitle='xxx';StreamUrl='xxx'</td>
</tr>
<tr>
<td>BASS_TAG_MF</td>
<td>Media Foundation metadata. A pointer to a series of null-terminated UTF-8 strings is returned, the final string ending with a double null.</td>
</tr>
<tr>
<td>BASS_TAG_MP4</td>
<td>MP4/iTunes metadata. A pointer to a series of null-terminated UTF-8 strings is returned, the final string ending with a double null.</td>
</tr>
<tr>
<td>BASS_TAG_MUSIC_INST</td>
<td>MOD instrument name. Only available with formats that have instruments, eg. IT and XM (and MO3).</td>
</tr>
<tr>
<td>+ instrument number (0=first)</td>
<td></td>
</tr>
<tr>
<td>BASS_TAG_MUSIC_MESSAGE</td>
<td>MOD message text.</td>
</tr>
<tr>
<td>BASS_TAG_MUSIC_NAME</td>
<td>MOD music title.</td>
</tr>
<tr>
<td>BASS_TAG_MUSIC_ORDERS</td>
<td>MOD music order list. A pointer to a byte array is returned, with each byte being the pattern number played at that order position. Pattern number 254 is &quot;+++&quot; (skip order) and 255 is &quot;---&quot; (end song).</td>
</tr>
<tr>
<td>BASS_TAG_MUSIC_SAMPLE</td>
<td>MOD sample name.</td>
</tr>
<tr>
<td>+ sample number (0=first)</td>
<td></td>
</tr>
<tr>
<td>Tag Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BASS_TAG_OGG</td>
<td>OGG comments. A pointer to a series of null-terminated UTF-8 strings is returned, the final string ending with a double null.</td>
</tr>
<tr>
<td>BASS_TAG_RIFF_BEXT</td>
<td>RIFF/BWF &quot;bext&quot; chunk tags. A pointer to a TAG_BEXT structure is returned.</td>
</tr>
<tr>
<td>BASS_TAG_RIFF_CART</td>
<td>RIFF/BWF &quot;cart&quot; chunk tags. A pointer to a TAG_CART structure is returned.</td>
</tr>
<tr>
<td>BASS_TAG_RIFF_DISP</td>
<td>RIFF &quot;DISP&quot; chunk text (CF_TEXT) tag. A single string is returned.</td>
</tr>
<tr>
<td>BASS_TAG_RIFF_INFO</td>
<td>RIFF &quot;INFO&quot; chunk tags. A pointer to a series of null-terminated strings is returned, the final string ending with a double null. The tags are in the form of &quot;XXXX=text&quot;, where &quot;XXXX&quot; is the chunk ID.</td>
</tr>
<tr>
<td>BASS_TAG_VENDOR</td>
<td>OGG encoder. A single UTF-8 string is returned.</td>
</tr>
<tr>
<td>BASS_TAG_WAVEFORMAT</td>
<td>WAVE &quot;fmt&quot; chunk contents. A pointer to a WAVEFORMATEX structure is returned. As well as WAVE files, this is also provided by Media Foundation codecs.</td>
</tr>
</tbody>
</table>

*other tags may be supported by add-ons, see the documentation.*
**Return value**
If successful, the requested tags are returned, else NULL is returned. Use [BASS_ErrorGetCode](#) to get the error code.
Error codes

BASS_ERROR_HANDLE  \( handle \) is not valid.
BASS_ERROR_NOTAVAIL  The requested tags are not available.
Remarks
Some tags (eg. ID3v1) are located at the end of the file, so when streaming a file from the internet, the tags will not be available until the download is complete. A BASS_SYNC_DOWNLOAD sync can be set via `BASS_ChannelSetSync`, to be informed of when the download is complete. A BASS_SYNC_META sync can be used to be informed of new Shoutcast metadata, and a BASS_SYNC_OGG_CHANGE sync for when a new logical bitstream begins in a chained OGG stream, which generally brings new OGG tags.

In a chained OGG file containing multiple bitstreams, each bitstream will have its own tags. To get the tags from a particular one, `BASS_ChannelSetPosition` can be first used to seek to it.

When a Media Foundation codec is in use, the BASS_TAG_WAVEFORMAT tag can be used to find out what the source format is, eg. via the WAVEFORMATEX structure's wFormatTag member. Some typical wFormatTag examples are: 0x0161 = WMA, 0x0162 = WMA pro, 0x0163 = WMA lossless, 0x1610 = AAC.
Example
List an OGG stream's comments.

```c
char *comments=BASS_ChannelGetTags(channel, BASS_TAG_OGG); // get a
if (comments)
    while (*comments) {
        printf("%s\n", comments); // display the comment
        comments+=strlen(comments)+1; // move on to next comment
    }
```

List a MOD music's samples.

```c
char *text;
int n=0;
while (text=BASS_ChannelGetTags(channel, BASS_TAG_MUSIC_SAMPLE+n)) {
    printf("sample %d = %s\n", n+1, text); // display the sample text
    n++;// move on to next sample
}
```
See also
BASS_ChannelGetInfo, BASS_ChannelSetSync
BASS_ChannelIsActive

Checks if a sample, stream, or MOD music is active (playing) or stalled. Can also check if a recording is in progress.

```
DWORD BASS_ChannelIsActive(
    DWORD handle
);
```
**Parameters**

handle  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD.
**Return value**
The return value is one of the following.

- **BASS_ACTIVE_STOPPED**: The channel is not active, or *handle* is not a valid channel.
- **BASS_ACTIVE_PLAYING**: The channel is playing (or recording).
- **BASS_ACTIVE_PAUSED**: The channel is paused.
- **BASS_ACTIVE_STALLED**: Playback of the stream has been stalled due to a lack of sample data. The playback will automatically resume once there is sufficient data to do so.
Remarks
When using this function with a decoding channel, BASS_ACTIVE_PLAYING will be returned while there is still data to decode. Once the end has been reached, BASS_ACTIVE_STOPPED will be returned. BASS_ACTIVE_STALLED is never returned for decoding channels; you can tell a decoding channel is stalled if BASS_ChannelGetData returns less data than requested, and this function still returns BASS_ACTIVE_PLAYING.
See also
BASS_ChannelGetLevel, BASS_ChannelPlay,
BASS_ATTRIB_NET_RESUME
BASS_ChannelIsSliding

Checks if an attribute (or any attribute) of a sample, stream, or MOD music is sliding.

```c
BOOL BASS_ChannelIsSliding(
    DWORD handle,
    DWORD attrib
);
```
Parameters

handle  The channel handle... a HCHANNEL, HSTREAM or HMUSIC.
attrib  The attribute to check for sliding... one of the following, or 0 for any attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ATTRIB_EAXMIX</td>
<td>EAX wet/dry mix.</td>
</tr>
<tr>
<td>BASS_ATTRIB_FREQ</td>
<td>Sample rate.</td>
</tr>
<tr>
<td>BASS_ATTRIB_PAN</td>
<td>Panning/balance position.</td>
</tr>
<tr>
<td>BASS_ATTRIB_FREQ</td>
<td>Sample rate.</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_AMPLIFY</td>
<td>Amplification level. (HMUSIC only)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_BPM</td>
<td>BPM. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_PANSEP</td>
<td>Pan separation level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_PSCALER</td>
<td>Position scaler. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_SPEED</td>
<td>Speed. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_CHAN</td>
<td>A channel volume level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_GLOBAL</td>
<td>Global volume level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_INST</td>
<td>An instrument/sample volume level. (HMUSIC)</td>
</tr>
</tbody>
</table>

Other attributes may be supported by add-ons, see the documentation.
Return value
If the attribute is sliding, then TRUE is returned, else FALSE is returned.
See also
BASS_ChannelSlideAttribute
BASS_ChannelLock

Locks a stream, MOD music or recording channel to the current thread.

```c
BOOL BASS_ChannelLock(
    DWORD handle,
    BOOL lock
);
```
Parameters

handle  The channel handle... a HMUSIC, HSTREAM or HRECORD.
lock    If FALSE, unlock the channel, else lock it.
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 valid channel.
**Remarks**
Locking a channel prevents other threads from performing most functions on it, including buffer updates. Other threads wanting to access a locked channel will block until it is unlocked, so a channel should only be locked very briefly. A channel must be unlocked in the same thread that it was locked.
Example
Lock a channel to ensure that 2 DSP functions start together.

```c
BASS_ChannelLock(channel, TRUE); // lock channel
BASS_ChannelSetDSP(channel, DspProc1, NULL, 0); // set 1st DSP
BASS_ChannelSetDSP(channel, DspProc2, NULL, 0); // set 2nd DSP
BASS_ChannelLock(channel, FALSE); // unlock channel
```
BASS_ChannelPause

Pauses a sample, stream, MOD music, or recording.

```c
BOOL BASS_ChannelPause(
    DWORD handle
);
```
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use `BASS_ErrorGetCode` to get the error code.
**Error codes**

**BASS_ERROR_NOPLAY**  The channel is not playing (or *handle* is not a valid channel).

**BASS_ERROR_DECODE**  The channel is not playable; it is a "decoding channel".

**BASS_ERROR_ALREADY**  The channel is already paused.
Remarks
Use `BASS_ChannelPlay` to resume a paused channel. `BASS_ChannelStop` can be used to stop a paused channel.
See also
BASS_ChannelIsActive, BASS_ChannelPlay, BASS_ChannelStop
BASS_ChannelPlay

Starts (or resumes) playback of a sample, stream, MOD music, or recording.

```c
BOOL BASS_ChannelPlay(
    DWORD handle,
    BOOL restart
);
```
**Parameters**

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

**restart**  Restart playback from the beginning? If *handle* is a user stream (created with [BASS_StreamCreate](https://www.bass.ca/doc/BASS_StreamCreate)), its current buffer contents are cleared. If it is a MOD music, its BPM/etc are reset to their initial values.
**Return value**
If successful, TRUE is returned, else FALSE is returned. Use [BASS_ErrorGetCode](#) to get the error code.
### Error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_HANDLE</td>
<td><em>handle</em> is not a valid channel.</td>
</tr>
<tr>
<td>BASS_ERROR_START</td>
<td>The output is paused/stopped, use <a href="#">BASS_Start</a> to start it.</td>
</tr>
<tr>
<td>BASS_ERROR_DECODE</td>
<td>The channel is not playable; it is a &quot;decoding channel&quot;.</td>
</tr>
<tr>
<td>BASS_ERROR_BUFLOST</td>
<td>Should not happen... check that a valid window handle was used with <a href="#">BASS_Init</a>.</td>
</tr>
<tr>
<td>BASS_ERROR_NOHW</td>
<td>No hardware voices are available (HCHANNEL only). This only occurs if the sample was loaded/created with the BASS_SAMPLE_VAM flag and BASS_VAM_HARDWARE is set in the sample's VAM mode, and there are no hardware voices available to play it.</td>
</tr>
</tbody>
</table>
Remarks
When streaming in blocks (BASS_STREAM_BLOCK), the restart parameter is ignored as it is not possible to go back to the start. The restart parameter is also of no consequence with recording channels.
See also
BASS_ChannelFlags, BASS_ChannelGetLevel, BASS_ChannelGetPosition, BASS_ChannelIsActive, BASS_ChannelPause, BASS_ChannelSetPosition, BASS_ChannelSetSync, BASS_ChannelStop, BASS_ChannelUpdate
BASS_ChannelRemoveDSP

Removes a DSP function from a stream, MOD music, or recording channel.

```c
BOOL BASS_ChannelRemoveDSP(
    DWORD handle,
    HDSP dsp
);
```
**Parameters**

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

**dsp**  Handle of the DSP function to remove from the channel. This can also be an HFX handle to remove an effect.
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 dsp is not valid.
See also
BASS_ChannelSetDSP
BASS_ChannelRemoveFX

Removes an effect on a stream, MOD music, or recording channel.

```c
BOOL BASS_ChannelRemoveFX(
    DWORD handle,
    HFX fx
);
```
**Parameters**

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

fx      Handle of the effect to remove from the channel.
**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 fx is not valid.
Remarks
Depending on the DX8 effect implementation being used by the channel, the channel may have to be stopped before removing a DX8 effect from it. If necessary, that is done automatically and the channel is resumed afterwards.

BASS_ChannelRemoveDSP can also be used to remove effects.
See also
BASS_ChannelSetFX, DX8 effect implementations
BASS_ChannelRemoveLink

Removes a links between two MOD music or stream channels.

BOOL BASS_ChannelRemoveLink(
    DWORD handle,
    DWORD chan
);
**Parameters**

handle The channel handle... a HMUSIC or HSTREAM.
chan The handle of the channel to have unlinked with it... a HMUSIC or HSTREAM.
**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  handle is not a valid channel.
BASS_ERROR_ALREADY Either chan is not a valid channel, or it is already not linked to handle.
See also
BASS_ChannelSetLink
BASS_ChannelRemoveSync

Removes a synchronizer from a MOD music, stream or recording channel.

```c
BOOL BASS_ChannelRemoveSync(
    DWORD handle,
    HSYNC sync
);
```
**Parameters**

- **handle**: The channel handle... a HMUSIC, HSTREAM or HRECORD.
- **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.
See also
BASS_ChannelSetSync
BASS_ChannelSeconds2Bytes

Translates a time (seconds) position into bytes, based on a channel's format.

QWORD BASS_ChannelSeconds2Bytes(
    DWORD handle,
    double pos
);
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD. HSAMPLE handles may also be used.

**pos**  The position to translate.
**Return value**
If successful, then the translated length is returned, else -1 is returned. Use [BASS_ErrorGetCode](#) to get the error code.
**Error codes**

BASS_ERROR_HANDLE  *handle* is not a valid channel.
Remarks
The translation is based on the channel's initial sample rate, when it was created.

The return value is rounded down to the position of the nearest sample.
See also
BASS_ChannelGetInfo, BASS_ChannelSetPosition,
BASS_ChannelBytes2Seconds
BASS_ChannelSet3DAttributes

Sets the 3D attributes of a sample, stream, or MOD music channel with 3D functionality.

```c
BOOL BASS_ChannelSet3DAttributes(
    DWORD handle,
    int mode,
    float min,
    float max,
    int iangle,
    int oangle,
    float outvol
);
```
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM.

**mode**  The 3D processing mode... one of these flags, -1 = leave current.
- **BASS_3DMODE_NORMAL**  Normal 3D processing.
- **BASS_3DMODE_RELATIVE**  The channel's 3D position (position/velocity/orientation) is relative to the listener. When the listener's position/velocity/orientation is changed with `BASS_Set3DPosition`, the channel's position relative to the listener does not change.
- **BASS_3DMODE_OFF**  Turn off 3D processing on the channel, the sound will be played in the centre.

**min**  The minimum distance. The channel's volume is at maximum when the listener is within this distance... 0 or less = leave current.

**max**  The maximum distance. The channel's volume stops decreasing when the listener is beyond this distance... 0 or less = leave current.

**iangle**  The angle of the inside projection cone in degrees... 0 (no cone) to 360 (sphere), -1 = leave current.

**oangle**  The angle of the outside projection cone in degrees... 0 (no cone) to 360 (sphere), -1 = leave current.

**outvol**  The delta-volume outside the outer projection cone... 0 (silent) to 1 (same as inside the cone), less than 0 = leave current.
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     handle is not a valid channel.
BASS_ERROR_NO3D       The channel does not have 3D functionality.
BASS_ERROR_ILLPARAM   One or more of the attribute values is invalid.
**Remarks**
The *iangle* and *oangle* parameters must both be set in a single call to this function; one cannot be set without the other. The *iangle* and *oangle* angles decide how wide the sound is projected around the orientation angle. Within the inside angle the volume level is the channel volume (*BASS_ATTRIB_VOL* attribute). Outside the outer angle, the volume changes according to the *outvol* value. Between the inner and outer angles, the volume gradually changes between the inner and outer volume levels. If the inner and outer angles are 360 degrees, then the sound is transmitted equally in all directions.

As with all 3D functions, use *BASS_Apply3D* to apply the changes made.
See also
BASS_ChannelGet3DAttributes, BASS_ChannelSet3DPosition,
BASS_ChannelSetAttribute, BASS_ATTRIB_EAXMIX
BASS_ChannelSet3DPosition

Sets the 3D position of a sample, stream, or MOD music channel with 3D functionality.

```c
BOOL BASS_ChannelSet3DPosition(
    DWORD handle,
    BASS_3DVECTOR *pos,
    BASS_3DVECTOR *orient,
    BASS_3DVECTOR *vel
);
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The channel handle... a HCHANNEL, HMUSIC, HSTREAM.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the sound... NULL = leave current.</td>
</tr>
<tr>
<td>orient</td>
<td>Orientation of the sound... NULL = leave current. This is automatically normalized.</td>
</tr>
<tr>
<td>vel</td>
<td>Velocity of the sound... NULL = leave current. This is only used to calculate the doppler effect, and has no effect on the sound's position.</td>
</tr>
</tbody>
</table>
**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  \textit{handle} is not a valid channel.

BASS_ERROR_NO3D  The channel does not have 3D functionality.
Remarks
As with all 3D functions, BASS_Apply3D must be called to apply the changes made.
See also
BASS_Apply3D, BASS_ChannelGet3DPosition,
BASS_ChannelSet3DAttributes, BASS_ChannelSetAttribute,
BASS_Set3DFactors, BASS_Set3DPosition, BASS_3DVECTOR structure
Sets the value of a channel's attribute.

```c
BOOL BASS_ChannelSetAttribute(
    DWORD handle,
    DWORD attrib,
    float value
);
```
Parameters

handle  The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HREC
attrib  The attribute to set the value of... one of the following.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ATTRIB_EAXMIX</td>
<td>EAX wet/dry mix. (HCHANNEL/HMUSIC/ONLY)</td>
</tr>
<tr>
<td>BASS_ATTRIB_FREQ</td>
<td>Sample rate.</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_AMPLIFY</td>
<td>Amplification level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_BPM</td>
<td>BPM. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_PANSEP</td>
<td>Pan separation level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_PSCALER</td>
<td>Position scaler. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_SPEED</td>
<td>Speed. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_CHAN</td>
<td>A channel volume level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_GLOBAL</td>
<td>Global volume level. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_MUSIC_VOL_INST</td>
<td>An instrument/sample volume. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_ATTRIB_NET_RESUME</td>
<td>Buffer level to resume stalled playback. (HSTREAM)</td>
</tr>
<tr>
<td>BASS_ATTRIB_NOBUFFER</td>
<td>Playback buffering switch. (HMUSIC/HSTREAM)</td>
</tr>
<tr>
<td>BASS_ATTRIB_PAN</td>
<td>Panning/balance position.</td>
</tr>
<tr>
<td>BASS_ATTRIB_SRC</td>
<td>Sample rate conversion quality.</td>
</tr>
<tr>
<td>BASS_ATTRIB_VOL</td>
<td>Volume level.</td>
</tr>
</tbody>
</table>

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

value  The new attribute value. See the attribute's documentation for details on possible values.
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`  
  *handle* is not a valid channel.

- `BASS_ERROR_ILLTYPE`  
  *attrib* is not valid.

- `BASS_ERROR_ILLPARAM`  
  *value* is not valid. See the attribute's documentation for the valid range of values.

*Some attributes may have additional error codes, see the documentation.*
Remarks
The actual attribute value may not be exactly the same as requested, due to precision differences. For example, an attribute might only allow whole number values. BASS_ChannelGetAttribute can be used to confirm what the value is.
See also
BASS_ChannelFlags, BASS_ChannelGetAttribute,
BASS_ChannelSetAttributeEx, BASS_ChannelSet3DAttributes,
BASS_ChannelSlideAttribute
BASS_ChannelSetAttributeEx

Sets the value of a channel's attribute.

```c
BOOL BASS_ChannelSetAttributeEx(
    DWORD handle,
    DWORD attrib,
    void *value,
    DWORD size
);
```
**Parameters**

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

**attrib**  The attribute to set the value of... one of the following.

- **BASS_ATTRIB_SCANINFO**  Scanned info. (HSTREAM only)

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

**value**  The new attribute data.

**size**  The size of the attribute data.
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 valid channel.
BASS_ERROR_ILLTYPE       attrib is not valid.
BASS_ERROR_ILLPARAM      The value content or size is not valid.

Some attributes may have additional error codes, see the documentation.
Remarks
This function also supports the floating-point attributes supported by 
BASS_ChannelGetAttribute.
See also
BASS_ChannelGetAttributeEx, BASS_ChannelSetAttribute
BASS_ChannelSetDevice

Changes the device that a stream, MOD music or sample is using.

```c
BOOL BASS_ChannelSetDevice(
    DWORD handle,
    DWORD device
);
```
**Parameters**

**handle**  The channel or sample handle... a HMUSIC, HSTREAM or HSAMPLE.

**device**  The device to use... 0 = no sound, 1 = first real output device.
Return value
If successful, TRUE is returned, else FALSE is returned and the channel remains on its current device. Use BASS_ErrorGetCode to get the error code.
**Error codes**

BASS_ERROR_HANDLE  
*handle* is not a valid channel.

BASS_ERROR_DEVICE  
*device* is invalid.

BASS_ERROR_INIT  
The requested device has not been initialized.

BASS_ERROR_ALREADY  
The channel is already using the requested device.

BASS_ERROR_NOTAVAIL  
Only decoding channels are allowed to use the "no sound" device.

BASS_ERROR_FORMAT  
The sample format is not supported by the device/drivers. If the channel is more than stereo or the BASS_SAMPLE_FLOAT flag is used, it could be that they are not supported.

BASS_ERROR_MEM  
There is insufficient memory.

BASS_ERROR_UNKNOWN  
Some other mystery problem!
Remarks
All of the channel's current settings are carried over to the new device, but if the channel is using the "with FX flag" DX8 effect implementation, the internal state (eg. buffers) of the DX8 effects will be reset. When using the "without FX flag" DX8 effect implementation, the state of the DX8 effects is preserved.

When changing a sample's device, all the sample's existing channels (HCHANNELs) are freed. It is not possible to change the device of an individual sample channel.
See also
BASS_ChannelGetDevice, BASS_Init
BASS_ChannelSetDSP

Sets up a user DSP function on a stream, MOD music, or recording channel.

```c
HDSP BASS_ChannelSetDSP(
    DWORD handle,
    DSPPROC *proc,
    void *user,
    int priority
);
```
**Parameters**

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

proc    The callback function.

user    User instance data to pass to the callback function.

priority The priority of the new DSP, which determines its position in the DSP chain. DSPs with higher priority are called before those with lower.
**Return value**
If successful, then the new DSP'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 valid channel.
Remarks
DSP functions can set and removed at any time, including mid-playback. Use `BASS_ChannelRemoveDSP` to remove a DSP function.

Multiple DSP functions may be used per channel, in which case the order that the functions are called is determined by their priorities. Any DSPs that have the same priority are called in the order that they were added.

DSP functions can be applied to MOD musics and streams, but not samples. If you want to apply a DSP function to a sample, then you should stream the sample.
See also
BASS_ChannelLock, BASS_ChannelRemoveDSP, BASS_ChannelSetFX, DSPPROC callback
BASS_ChannelSetFX

Sets an effect on a stream, MOD music, or recording channel.

HFX BASS_ChannelSetFX(
    DWORD handle,
    DWORD type,
    int priority
);
### Parameters

<table>
<thead>
<tr>
<th>handle</th>
<th>The channel handle... a HSTREAM, HMUSIC, or HRECORD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of the following types of effect.</td>
</tr>
<tr>
<td></td>
<td>DX8 Chorus. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_CHORUS</td>
<td>BASS_DX8_CHORUS structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Compression. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_COMPRESSOR</td>
<td>BASS_DX8_COMPRESSOR structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Distortion. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_DISTORTION</td>
<td>BASS_DX8_DISTORTION structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Echo. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_ECHO</td>
<td>BASS_DX8_ECHO structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Flanger. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_FLANGER</td>
<td>BASS_DX8_FLANGER structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Gargle. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_GARGLE</td>
<td>BASS_DX8_GARGLE structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 I3DL2 (Interactive 3D Audio Level 2) reverb.</td>
</tr>
<tr>
<td>BASS_FX_DX8_I3DL2REVERB</td>
<td>BASS_DX8_I3DL2REVERB structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Parametric equalizer.</td>
</tr>
<tr>
<td>BASS_FX_DX8_PARAMEQ</td>
<td>BASS_DX8_PARAMEQ structure to set/get parameters.</td>
</tr>
<tr>
<td></td>
<td>DX8 Reverb. Use</td>
</tr>
<tr>
<td>BASS_FX_DX8_REVERB</td>
<td>BASS_DX8_REVERB structure to set/get parameters.</td>
</tr>
</tbody>
</table>

*other effects may be supported by add-ons, eg. BASS_FX.*

<table>
<thead>
<tr>
<th>priority</th>
<th>The priority of the new FX, which determines its position in the DSP chain. DSP/FX with higher priority are applied before those with</th>
</tr>
</thead>
</table>
lower. This parameter has no effect with DX8 effects when the "with FX flag" DX8 effect implementation is used.
Return value
If successful, then the new effect'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 valid channel.
- **BASS_ERROR_ILLTYPE**: `type` is invalid.
- **BASS_ERROR_NOFX**: The specified DX8 effect is unavailable.
- **BASS_ERROR_FORMAT**: The channel's format is not supported by the effect.
- **BASS_ERROR_UNKNOWN**: Some other mystery problem!
Remarks
Multiple effects may be used per channel. Use BASS_ChannelRemoveFX to remove an effect. Use BASS_FXSetParameters to set an effect's parameters.

Effects can be applied to MOD musics and streams, but not samples. If you want to apply an effect to a sample, you could use a stream instead.

Depending on the DX8 effect implementation being used by the channel, the channel may have to be stopped before adding or removing DX8 effects on it. If necessary, that is done automatically and the channel is resumed afterwards.
Platform-specific
DX8 effects are a Windows feature requiring DirectX 8, or DirectX 9 for floating-point support. On other platforms, they are emulated by BASS, except for the following which are currently unsupported: COMPRESSOR, GARGLE, and I3DL2REVERB.
See also
BASS_ChannelLock, BASS_ChannelRemoveFX, BASS_FXGetParameters,
BASS_FXReset, BASS_FXSetParameters, BASS_ChannelSetDSP, DX8 effect
implementations
BASS_ChannelSetLink

Links two MOD music or stream channels together.

```c
BOOL BASS_ChannelSetLink(
    DWORD handle,
    DWORD chan
);
```
**Parameters**

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

chan    The handle of the channel to have linked with it... a HMUSIC or HSTREAM.
Return value
If successful, then TRUE is returned, else FALSE is returned. Use BASS_ErrorGetCode to get the error code.
### Error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_ERROR_HANDLE</td>
<td>At least one of <code>handle</code> and <code>chan</code> is not a valid channel.</td>
</tr>
<tr>
<td>BASS_ERROR_DECODE</td>
<td>At least one of <code>handle</code> and <code>chan</code> is a &quot;decoding channel&quot;, so cannot be linked.</td>
</tr>
<tr>
<td>BASS_ERROR_ALREADY</td>
<td><code>chan</code> is already linked to <code>handle</code>.</td>
</tr>
<tr>
<td>BASS_ERROR_UNKNOWN</td>
<td>Some other mystery problem!</td>
</tr>
</tbody>
</table>
Remarks
Linked channels are started/stopped/paused/resumed together. Links are one-way; for example, channel chan will be started by channel handle, but not vice versa unless another link has been set in that direction.

If a linked channel has reached the end, it will not be restarted when a channel it is linked to is started. If you want a linked channel to be restarted, you need to have reset its position using BASS_ChannelSetPosition beforehand.
**Platform-specific**
Except for on Windows, linked channels on the same device are guaranteed to start playing simultaneously. On Windows, it is possible for there to be a slight gap between them, but it will generally be shorter (and never longer) than starting them individually.
**Example**
Link 2 streams and play them together.

```
BASS_ChannelSetLink(stream1, stream2); // link stream2 to stream1
BASS_ChannelPlay(stream1, FALSE); // start both streams together
```
See also
BASS_ChannelRemoveLink
Sets the playback position of a sample, MOD music, or stream.

```c
BOOL BASS_ChannelSetPosition(
    DWORD handle,
    QWORD pos,
    DWORD mode
);
```
**Parameters**

**handle**  The channel handle... a HCHANNEL, HSTREAM or HMUSIC.

**pos**  The position, in units determined by the **mode**.

**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(order,row). (HMUSIC only)

- **BASS_POS_OGG**  The position is a bitstream number in an OGG file... 0 = first.

- **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_POS_INEXACT**  Flag: Allow inexact seeking. For speed, seeking may stop at the beginning of a block rather than partially processing the block to reach the requested position.

- **BASS_POS_SCAN**  Flag: Scan the file to build a seek table up to the position, if it has not already been scanned. Scanning will continue from where it left off previously rather than restarting from the beginning of the file each time. This flag only applies to MP3/MP2/MP1 files and will be ignored with other file formats.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASS_MUSIC_POSRESET</td>
<td>Flag: Stop all notes. This flag is applied automatically if it has been set on the channel, eg. via BASS_ChannelFlags.</td>
<td>(HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_POSRESETEX</td>
<td>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.</td>
<td>(HMUSIC)</td>
</tr>
</tbody>
</table>

*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       \textit{handle} is not a valid channel.
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 \textit{mode} is not available. Invalid flags are ignored and do not result in this error.
BASS_ERROR_UNKNOWN     Some other mystery problem!
Remarks
Setting the position of a MOD music in bytes (other than 0) requires that the BASS_MUSIC_PRESSCAN flag was used in the BASS_MusicLoad call, or the use of the BASS_POS_DECODETO flag. When setting the position in orders and rows, the channel's byte position (as reported by BASS_ChannelGetPosition) is reset to 0. That is because it is not possible to get the byte position of an order/row position; it is possible for an order/row position to never be played in the normal course of events, or it may be played multiple times.

When setting the position of a MOD music, and the BASS_MUSIC_POSRESET flag is active, all notes that were playing before the position changed will be stopped. Otherwise, the notes will continue playing until they are stopped in the MOD music. When setting the position in bytes, the BPM, speed and global volume are updated to what they would normally be at the new position. Otherwise they are left as they were prior to the position change, unless the seek position is 0 (the start), in which case they are also reset to the starting values (with the BASS_MUSIC_POSRESET flag). When the BASS_MUSIC_POSRESETEX flag is active, the BPM, speed and global volume are reset with every seek.

For MP3/MP2/MP1 streams, unless the file is scanned via the BASS_POS_SCAN flag or the BASS_STREAM_PRESCAN flag at stream creation, seeking will be approximate but generally still quite accurate. Besides scanning, exact seeking can also be achieved with the BASS_POS_DECODETO flag.

Seeking in internet file (and "buffered" user file) streams is possible once the download has reached the requested position, so long as the file is not being streamed in blocks (BASS_STREAM_BLOCK flag).

User streams (created with BASS_StreamCreate) are not seekable, but it is possible to reset a user stream (including its buffer contents) by setting its position to byte 0.

The BASS_POS_DECODETO flag can be used to seek forwards in streams that are not normally seekable, like custom streams or internet streams that are using the BASS_STREAM_BLOCK flag, but it will only go as far as what is currently
available; it will not wait for more data to be downloaded, for example.

In some cases, particularly when the BASS_POS_INEXACT flag is used, the new position may not be what was requested. BASS_ChannelGetPosition can be used to confirm what the new position actually is.

The BASS_POS_SCAN flag works the same way as the BASS_StreamCreateFile BASS_STREAM_PRESCAN flag, and can be used to delay the scanning until after the stream has been created. When a position beyond the end is requested, the call will fail (BASS_ERROR_POSITION error code) but the seek table and exact length will have been scanned. When a file has been scanned, all seeking (even without the BASS_POS_SCAN flag) within the scanned part of it will use the scanned information.
**Example**
Set the position of a channel to 1000000 bytes.

```c
BASS_ChannelSetPosition(channel, 1000000, BASS_POS_BYTE);
```

Set the position of a MOD music to row 20 of order 10, and stop all currently playing notes.

```c
BASS_ChannelSetPosition(music, MAKELONG(10, 20), BASS_POS_MUSIC_ORDER|BASS_MUSIC_POSRESET);
```
See also
BASS_ChannelFlags, BASS_ChannelGetLength, BASS_ChannelGetPosition,
BASS_ChannelIsActive, BASS_ChannelSeconds2Bytes, BASS_ChannelUpdate
BASS_ChannelSetSync

Sets up a synchronizer on a MOD music, stream or recording channel.

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

Parameters
handle  The channel handle... a HMUSIC, HSTREAM or HRECORD.
type   The type of sync (see the table below). The following flags may also be used.
       BASS_SYNC_MIXTIME  Call the sync function immediately when the sync is triggered, instead of delaying the call until the sync event is actually heard. This is automatic with some sync types (see table below), and always with decoding and recording channels, as they cannot be played/heard.
       BASS_SYNC_ONETIME  Call the sync only once, and then remove it from the channel.
param  The sync parameter. Depends on the sync type... see the table below.
proc   The callback function.
user   User instance data to pass to the callback function.

Sync types, with param and SYNCPROC data definitions.
BASS_SYNC_DOWNLOAD  Sync when downloading of an internet (or "buffered" user file) stream is done.
               mixtime only
       param: not used. data: not used.

BASS_SYNC_END  Sync when a channel reaches the end, including when looping. Note that some MOD musics never reach the end; they may jump to another position first. If the BASS_MUSIC_STOPBACK flag is used with a MOD music (through BASS_MusicLoad or BASS_ChannelFlags), then this sync will also be called when a backward jump effect is played.
       param: not used. data: 1 = the sync is triggered by a backward jump in a MOD music, otherwise not used.

BASS_SYNC_FREE  Sync when a channel is freed. This can be useful when you need to release some
resources associated with the channel. Note that you will not be able to use any BASS functions with the channel in the callback, as the channel will no longer exist.

**param** : not used. **data** : not used.

**BASS_SYNC_META**

Sync when metadata is received in a Shoutcast stream. The updated metadata is available from **BASS_ChannelGetTags**.

**param** : not used. **data** : not used.

**BASS_SYNC_MUSICFX**

Sync when the sync effect is used in a MOD music. The sync effect is $E8x$ or $Wxx$ for the XM/MTM/MOD formats, and $S2x$ for the IT/S3M formats (where $x = \text{any value}$).

**param** : $0 = \text{the position is passed to the callback} \ (\text{data} : \text{LOWORD} = \text{order}, \text{HIWORD} = \text{row}), \ 1 = \text{the value of } x \text{ is passed to the callback} \ (\text{data} : x \text{ value})$.

**BASS_SYNC_MUSICINST**

Sync when an instrument (sample for the MOD/S3M/MTM formats) is played in a MOD music (not including retrigs).

**param** : $\text{LOWORD} = \text{instrument} \ (1=\text{first}), \ \text{HIWORD} = \text{note} \ (0=c0...119=b9, -1=\text{all})$.

**data** : $\text{LOWORD} = \text{note}, \text{HIWORD} = \text{volume} \ (0-64)$.

**BASS_SYNC_MUSICPOS**

Sync when a MOD music reaches an order/row position.

**param** : $\text{LOWORD} = \text{order} \ (0=\text{first}, -1=\text{all}), \ \text{HIWORD} = \text{row} \ (0=\text{first}, -1=\text{all})$. **data** : $\text{LOWORD} = \text{order}, \text{HIWORD} = \text{row}$.

**BASS_SYNC_OGG_CHANGE**

Sync when a new logical bitstream begins in a chained OGG stream. Updated tags are available from **BASS_ChannelGetTags**.

**param** : not used. **data** : not used.

**BASS_SYNC_POS**

Sync when a channel reaches a position.

**param** : position in bytes (automatically rounded down to nearest sample). **data** : not
used.

**BASS_SYNC_SETPOS**

Sync when a channel's position is set, including when looping/restarting.

*param* : not used. *data* : 0 = playback buffer is not flushed, 1 = playback buffer is flushed.

**BASS_SYNC_SLIDE**

Sync when an attribute slide has ended.

*param* : not used. *data* : the attribute that has finished sliding.

**BASS_SYNCSTALL**

Sync when playback of the channel is stalled/resumed.

*param* : not used. *data* : 0 = stalled, 1 = resumed.

*other sync types may be supported by add-ons, see the documentation.*
**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  handle is not a valid channel.
BASS_ERROR_ILLTYPE  An illegal type was specified.
BASS_ERROR_ILLPARAM  An illegal param was specified.
Remarks
Multiple synchronizers may be used per channel, and they can be set before and while playing. Equally, synchronizers can also be removed at any time, using BASS_ChannelRemoveSync. If the BASS_SYNC_ONETIME flag is used, then the sync is automatically removed after its first occurrence.

The BASS_SYNC_MIXTIME flag can be used with BASS_SYNC_END or BASS_SYNC_POS/MUSICPOS syncs to implement custom looping, by using BASS_ChannelSetPosition in the callback. A mixtime sync can also be used to make DSP/FX changes at specific points, or change a HMUSIC channel's flags or attributes. The BASS_SYNC_MIXTIME flag can also be useful with a BASS_SYNC_SETPOS sync, to reset DSP states after seeking.

Several of the sync types are triggered in the process of rendering the channel's sample data; for example, BASS_SYNC_POS and BASS_SYNC_END syncs, when the rendering reaches the sync position or the end, respectively. Those sync types should be set before starting playback or pre-buffering (ie. before any rendering), to avoid missing any early sync events.

With recording channels, BASS_SYNC_POS syncs are triggered just before the RECORDPROC receives the block of data containing the sync position.
Example
Do some processing until a MOD music reaches the 10th order.

```c
BOOL order10=FALSE; // the order 10 flag

// the sync callback
void CALLBACK MySyncProc(HSYNC handle, DWORD channel, DWORD data, void *user)
{
    order10=TRUE; // set the order 10 flag
}

// the sync callback
BASS_ChannelSetSync(music, BASS_SYNC_MUSICPOS|BASS_SYNC_ONETIME, MAKELONG(10,0), MySyncProc, 0); // set the one-time order 10 sync
while (!order10) {
    // order 10 has not arrived, so do some processing
}
// order 10 has arrived!
```

Process metadata received from a Shoutcast stream.

```c
char title[100]=""; // the current stream title

// the sync callback
void CALLBACK MyMetaSyncProc(HSYNC handle, DWORD channel, DWORD data, void *user)
{
    char *meta=BASS_ChannelGetTags(channel, BASS_TAG_META); // get metadata
    meta=strstr(meta, "StreamTitle='"); // look for title
    if (meta) {
        // found it, copy it...
        strcpy(title, meta+13);
        strchr(title, ';')[-1]=0;
    }
}

// the sync callback
BASS_ChannelSetSync(stream, BASS_SYNC_META, 0, MyMetaSyncProc, 0); // set the meta sync
```
See also
BASS_ChannelRemoveSync, SYNCPROC callback
BASS_ChannelSlideAttribute

Slides a channel's attribute from its current value to a new value.

```c
BOOL BASS_ChannelSlideAttribute(
    DWORD handle,
    DWORD attrib,
    float value,
    DWORD time
);
```
**Parameters**

**handle**  The channel handle... a HCHANNEL, HSTREAM, HMUSIC, or HREC

**attrib**  The attribute to slide the value of... one of the following.

- **BASS_ATTRIB_EAXMIX**  EAX wet/dry mix.  (HCHANNEL/HMUSIC/HSTREAM only)
- **BASS_ATTRIB_FREQ**  Sample rate.
- **BASS_ATTRIB_MUSIC_AMPLIFY**  Amplification level. (HMUSIC)
- **BASS_ATTRIB_MUSIC_BPM**  BPM. (HMUSIC)
- **BASS_ATTRIB_MUSIC_PANSEP**  Pan separation level. (HMUSIC)
- **BASS_ATTRIB_MUSIC_PSCALER**  Position scaler. (HMUSIC)
- **BASS_ATTRIB_MUSIC_SPEED**  Speed. (HMUSIC)
- **BASS_ATTRIB_MUSIC_VOL_CHAN**  A channel volume level. (HMUSIC)
- **BASS_ATTRIB_MUSIC_VOL_GLOBAL**  Global volume level. (HMUSIC)
- **BASS_ATTRIB_MUSIC_VOL_INST**  An instrument/sample volume level. (HMUSIC)
- **BASS_ATTRIB_PAN**  Panning/balance position.
- **BASS_ATTRIB_VOL**  Volume level.

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

**value**  The new attribute value. See the attribute's documentation for details on possible values.

**time**  The length of time (in milliseconds) that it should take for the attribute to reach value.
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  handle is not a valid channel.
BASS_ERROR_ILLTYPE  attrib is not valid.
some attributes may have additional error codes, see the documentation.
Remarks
This function is similar to BASS_ChannelSetAttribute, except that the attribute is ramped to the value over the specified period of time. Another difference is that the value is not pre-checked. If it is invalid, the slide will simply end early.

If an attribute is already sliding, then the old slide is stopped and replaced by the new one.

BASS_ChannelIsSliding can be used to check if an attribute is currently sliding. A BASS_SYNC_SLIDE sync can also be set via BASS_ChannelSetSync, to be triggered at the end of a slide. The sync will not be triggered in the case of an existing slide being replaced by a new one.

Attribute slides are unaffected by whether the channel is playing, paused or stopped. They carry on regardless.
Example
Fadeout a channel's volume over a period of 1 second.

BASS_ChannelSlideAttribute(channel, BASS_ATTRIB_VOL, 0, 1000);
See also
BASS_ChannelGetAttribute, BASS_ChannelIsSliding, BASS_ChannelSetAttribute, BASS_ChannelSetSync
BASS_ChannelStop

Stops a sample, stream, MOD music, or recording.

```c
BOOL BASS_ChannelStop(
    DWORD handle
);
```
**Parameters**

handle   The channel handle... a HCHANNEL, HMUSIC, HSTREAM, or HRECORD.
**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 valid channel.
**Remarks**
Stopping a user stream (created with `BASS_StreamCreate`) will clear its buffer contents, and stopping a sample channel (HCHANNEL) will result in it being freed. Use `BASS_ChannelPause` instead if you wish to stop a user stream or sample and then resume it from the same point.

When used with a decoding channel, this function will end the channel at its current position, so that it is not possible to decode any more data from it. Any `BASS_SYNC_END` syncs that have been set on the channel will not be triggered by this; they are only triggered when reaching the natural end. `BASS_ChannelSetPosition` can be used to reset the channel and start decoding again.
See also
BASS_ChannelIsActive, BASS_ChannelPause, BASS_ChannelPlay,
BASS_RecordStart, BASS_SampleStop
BASS_ChannelUpdate

Updates the playback buffer of a stream or MOD music.

```c
BOOL BASS_ChannelUpdate(
    DWORD handle,
    DWORD length
);
```
**Parameters**

- **handle**: The channel handle... a HMUSIC or HSTREAM.
- **length**: The amount of data to render, in milliseconds... 0 = default (2 x update period). This is capped at the space available in the buffer.
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 valid channel.
- BASS_ERROR_NOTAVAIL: Decoding channels do not have playback buffers.
- BASS_ERROR_ENDED: The channel has ended.
- BASS_ERROR_UNKNOWN: Some other mystery problem!
Remarks
When starting playback of a stream or MOD music, after creating it or changing its position, there will be a slight delay while the initial data is generated for playback. Usually the delay is not noticeable or important, but if you need playback to start instantly when you call `BASS_ChannelPlay`, then use this function first. The `length` parameter should be at least equal to the `update period`.

It may not always be possible to render the requested amount of data, in which case this function will still succeed. `BASS_ChannelGetData` (BASS_DATA_AVAILABLE) can be used to check how much data a channel has buffered for playback.

When automatic updating is disabled (`BASS_CONFIG_UPDATEPERIOD = 0` or `BASS_CONFIG_UPDATETHREADS = 0`), this function could be used instead of `BASS_Update` to implement different update periods for different channels, instead of a single update period for all. Unlike `BASS_Update`, this function can also be used while automatic updating is enabled.

The CPU usage of this function is not included in the `BASS_GetCPU` reading, but is included in the channel's `BASS_ATTRIB_CPU` attribute value.
See also
BASS_ChannelPlay, BASS_Update
BASS_ATTRIB_EAXMIX attribute

The wet (reverb) / dry (no reverb) mix ratio of a channel.

BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_EAXMIX,
    float mix
);
**Parameters**

**handle**  The channel handle... a HCHANNEL, HMUSIC, HSTREAM.

**mix**  The wet / dry ratio... 0 (full dry) to 1 (full wet), -1 = automatically calculate the mix based on the distance (the default).
**Additional error codes**

**BASS_ERROR_NOEAX**  The channel does not have EAX support. EAX only applies to 3D channels that are mixed by the hardware/drivers. [BASS_ChannelGetInfo](#) can be used to check if a channel is being mixed by the hardware.
Platform-specific
EAX is only supported on Windows.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_SetEAXParameters
BASS_ATTRIB_CPU attribute

The CPU usage of a channel.

BASS_ChannelGetAttribute(
    DWORD handle,
    BASS_ATTRIB_CPU,
    float *cpu
);
Parameters
handle The channel handle... a HMUSIC or HSTREAM.
cpu The CPU usage.
**Remarks**

This attribute gives the percentage of CPU that the channel is using, including the time taken by decoding and DSP processing, and any FX that are not using the "with FX flag" DX8 effect implementation. It does not include the time taken to add the channel's data to the final output mix during playback. The processing of some add-on stream formats may also not be entirely included, if they use additional decoding threads; see the add-on documentation for details.

Like [BASS_GetCPU](#), this function does not strictly tell the CPU usage, but rather how timely the processing is. For example, if it takes 10ms to generate 100ms of data, that would be 10%. If the reported usage exceeds 100%, that means the channel's data is taking longer to generate than to play. The duration of the data is based on the channel's current sample rate ([BASS_ATTRIB_FREQ](#)).

A channel's CPU usage is updated whenever it generates data. That could be during a playback buffer update cycle, or a [BASS_Update](#) call, or a [BASS_ChannelUpdate](#) call. For a decoding channel, it would be in a [BASS_ChannelGetData](#) or [BASS_ChannelGetLevel](#) call.

This attribute is read-only, so cannot be modified via [BASS_ChannelSetAttribute](#).
See also
BASS_ChannelGetAttribute, BASS_GetCPU
BASS_ATTRIB_FREQ attribute

The sample rate of a channel.

```c
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_FREQ,
    float freq
);
```
**Parameters**

handle  The channel handle.

freq   The sample rate... 0 = original rate (when the channel was created).
Remarks
This attribute applies to playback of the channel, and does not affect the channel's sample data, so it has no real effect on decoding channels. It is still adjustable then though, so that it can be used by the BASSmix add-on and anything else that wants to use it.

It is not possible to change the sample rate of a channel if the "with FX flag" DX8 effect implementation enabled on it, unless DirectX 9 or above is installed.

Increasing the sample rate of a stream or MOD music increases its CPU usage, and reduces the length of its playback buffer in terms of time. If you intend to raise the sample rate above the original rate, then you may also need to increase the buffer length via the BASS_CONFIG_BUFFER config option to avoid break-ups in the sound.
**Platform-specific**
On Windows, the sample rate will get rounded down to a whole number during playback.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_GetInfo
**BASS_ATTRIB_MUSIC_ACTIVE attribute**

The number of active channels in a MOD music.

```c
BASS_ChannelGetAttribute( 
  HMUSIC handle,  
  BASS_ATTRIB_MUSIC_ACTIVE,  
  float *active  
);  
```
**Parameters**

- **handle**  The MOD music handle.
- **active**  The number of channels.
**Remarks**
This attribute gives the number of channels (including virtual) that are currently active in the decoder, which may not match what is being heard during playback due to buffering. To reduce the time difference, use the `BASS_CONFIG_BUFFER` config option to reduce the buffer length.

This attribute is read-only, so cannot be modified via `BASS_ChannelSetAttribute`. 
See also
BASS_ChannelGetAttribute, BASS_CONFIG_MUSIC_VIRTUAL
BASS_ATTRIB_MUSIC_AMPLIFY
attribute

The amplification level of a MOD music.

BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_AMPLIFY,
    float amp
);
Parameters

handle  The MOD music handle.
amp    Amplification level... 0 (min) to 100 (max). This will be rounded down to a whole number.
Remarks
As the amplification level get's higher, the sample data's range increases, and therefore, the resolution increases. But if the level is set too high, then clipping can occur, which can result in distortion of the sound.

You can check the current level of a MOD music at any time by using BASS_ChannelGetLevel. By doing so, you can decide if a MOD music's amplification level needs adjusting.

The default amplification level is 50.

During playback, the effect of changes to this attribute are not heard instantaneously, due to buffering. To reduce the delay, use the BASS_CONFIG_BUFFER config option to reduce the buffer length.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute
BASS_ATTRIB_MUSIC_BPM attribute

The BPM of a MOD music.

```c
BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_BPM,
    float bpm
);
```
**Parameters**

**handle**  The MOD music handle.

**bpm**  The BPM... 1 (min) to 255 (max). This will be rounded down to a whole number.
Remarks
This attribute is a direct mapping of the MOD's BPM, so the value can be changed via effects in the MOD itself.

Note that by changing this attribute, you are changing the playback length.

During playback, the effect of changes to this attribute are not heard instantaneously, due to buffering. To reduce the delay, use the `BASS_CONFIG_BUFFER` config option to reduce the buffer length.
See also

BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_ATTRIB_MUSIC_SPEED
**BASS_ATTRIB_MUSIC_PANSEP attribute**

The pan separation level of a MOD music.

```c
BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_PANSEP,
    float pansep
);
```
**Parameters**

*handle*  The MOD music handle.

*pansep*  Pan separation... 0 (min) to 100 (max), 50 = linear. This will be rounded down to a whole number.
Remarks
By default BASS uses a linear panning "curve". If you want to use the panning of FT2, use a pan separation setting of around 35. To use the Amiga panning (ie. full left and right) set it to 100.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute, BASS_ChannelSlideAttribute
The position scaler of a MOD music.

```c
BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_PSCALER,
    float scale
);
```
**Parameters**

handle  The MOD music handle.

scale  The scaler... 1 (min) to 256 (max). This will be rounded down to a whole number.
Remarks
When getting the order position via `BASS_ChannelGetPosition`, the row will be scaled by this value. By using a higher scaler, you can get a more precise position indication.

The default position scaler is 1.
Example
Get the position of a MOD music accurate to within a 10th of a row.

```c
DWORD pos, order, row, row10th;
BASS_ChannelSetAttribute(music, BASS_ATTRIB_MUSIC_PSCALER, 10); //
pos=ChannelGetPosition(music, BASS_POS_MUSIC_ORDER);
order=LOWORD(pos); // the order
row=HIWORD(pos)/10; // the row
row10th=HIWORD(pos)%10; // the 10th of a row
```
See also
BASS_ChannelGetAttribute, BASS_ChannelGetPosition,
BASS_ChannelSetAttribute, BASS_ChannelSlideAttribute
The speed of a MOD music.

BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_SPEED,
    float speed
);
Parameters

handle  The MOD music handle.
speed   The speed... 0 (min) to 255 (max). This will be rounded down to a whole number.
Remarks
This attribute is a direct mapping of the MOD's speed, so the value can be changed via effects in the MOD itself.

The "speed" is the number of ticks per row. Setting it to 0, stops and ends the music. Note that by changing this attribute, you are changing the playback length.

During playback, the effect of changes to this attribute are not heard instantaneously, due to buffering. To reduce the delay, use the BASS_CONFIG_BUFFER config option to reduce the buffer length.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_ATTRIB_MUSIC_BPM
**BASS_ATTRIB_MUSIC_VOL_CHAN attribute**

The volume level of a channel in a MOD music.

```c
BASS_ChannelSetAttribute(
    HMUSIC handle,  
    BASS_ATTRIB_MUSIC_VOL_CHAN + channel,  
    float volume  
);
```
**Parameters**

handle  The MOD music handle.

channel  The channel to set the volume of... 0 = 1st channel.

volume  The volume level... 0 (silent) to 1 (full).
Remarks
The volume curve used by this attribute is always linear, eg. 0.5 = 50%. The BASS_CONFIG_CURVE_VOL config option setting has no effect on this. The volume level of all channels is initially 1 (full).

During playback, the effect of changes to this attribute are not heard instantaneously, due to buffering. To reduce the delay, use the BASS_CONFIG_BUFFER config option to reduce the buffer length.

This attribute can also be used to count the number of channels in a MOD Music.
**Example**

Count the number of channels in a MOD music.

```c
int channels=0;
float dummy;
while (BASS_ChannelGetAttribute(music, BASS_ATTRIB_MUSIC_VOL_CHAN+channels++, &dummy))
```
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_ATTRIB_MUSIC_VOL_INST
The global volume level of a MOD music.

```c
BASS_ChannelSetAttribute(
    HMUSIC handle,
    BASS_ATTRIB_MUSIC_VOL_GLOBAL,
    float volume
);
```
**Parameters**

handle  The MOD music handle.

volume  The global volume level... 0 (min) to 64 (max, 128 for IT format). This will be rounded down to a whole number.
**Remarks**
This attribute is a direct mapping of the MOD’s global volume, so the value can be changed via effects in the MOD itself.

During playback, the effect of changes to this attribute are not heard instantaneously, due to buffering. To reduce the delay, use the `BASS_CONFIG_BUFFER` config option to reduce the buffer length.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_ATTRIB_MUSIC_AMPLIFY
**BASS_ATTRIB_MUSIC_VOL_INST**

attribute

The volume level of an instrument in a MOD music.

```c
BASS_ChannelSetAttribute(  
    HMUSIC handle,  
    BASS_ATTRIB_MUSIC_VOL_INST + inst,  
    float volume  
);
```
Parameters

handle  The MOD music handle.
inst    The instrument to set the volume of... 0 = 1st instrument.
volume  The volume level... 0 (silent) to 1 (full).
Remarks
The volume curve used by this attribute is always linear, eg. 0.5 = 50%. The
BASS_CONFIG_CURVE_VOL config option setting has no effect on this. The
volume level of all instruments is initially 1 (full). For MOD formats that do not
use instruments, read "sample" for "instrument".

During playback, the effect of changes to this attribute are not heard
instantaneously, due to buffering. To reduce the delay, use the
BASS_CONFIG_BUFFER config option to reduce the buffer length.

This attribute can also be used to count the number of instruments in a MOD
music.
**Example**

Count the number of instruments in a MOD music.

```c
int instruments=0;
float dummy;
while (BASS_ChannelGetAttribute(music, BASS_ATTRIB_MUSIC_VOL_INST+instruments, &dummy) != -1)
    instruments++;
```
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_ATTRIB_MUSIC_VOL_CHAN
BASS_ATTRIB_NET_RESUME attribute

The download buffer level required to resume stalled playback.

```c
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_NET_RESUME,
    float resume
);
```
**Parameters**

- **handle**  The channel handle.
- **resume**  The resumption level in percent... 0 - 100.
Remarks
This attribute determines what percentage of the download buffer
(BASS_CONFIG_NET_BUFFER) needs to be filled before playback of a
stalled internet stream will resume. It also applies to "buffered" user file streams
created with BASS_StreamCreateFileUser.

The default is 50%.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_CONFIG_NET_BUFFER
BASS_ATTRIB_NOBUFFER attribute

Disable playback buffering?

```c
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_NOBUFFER,
    float nobuffer
);
```
Parameters

handle The channel handle... a HMUSIC or HSTREAM.
nobuffer Disable playback buffering... 0 = no, else yes.
Remarks
A playing channel is normally asked to render data to its playback buffer in advance, via automatic buffer updates or the `BASS_Update` and `BASS_ChannelUpdate` functions, ready for mixing with other channels to produce the final mix that is given to the output device. When this attribute is switched on (the default is off), that buffering is skipped and the channel will only be asked to produce data as it is needed during the generation of the final mix. This allows the lowest latency to be achieved, but also imposes tighter timing requirements on the channel to produce its data and apply any DSP/FX (and run mixtime syncs) that are set on it; if too long is taken, there will be a break in the output, affecting all channels that are playing on the same device.

The channel's data is still placed in its playback buffer when this attribute is on, which allows `BASS_ChannelGetData` and `BASS_ChannelGetLevel` to be used, although there is likely to be less data available to them due to the buffer being less full.

This attribute can be changed mid-playback. If switched on, any already buffered data will still be played, so that there is no break in sound.
Platform-specific
This attribute is not available on Windows, as BASS does not generate the final mix.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute
BASS_ATTRIB_PAN attribute

The panning/balance position of a channel.

```c
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_PAN,
    float pan
);
```
**Parameters**

handle  The channel handle.

pan  The pan position... -1 (full left) to +1 (full right), 0 = centre.
Remarks
This attribute applies to playback of the channel, and does not affect the channel's sample data, so it has no real effect on decoding channels. It is still adjustable then though, so that it can be used by the BASSmix add-on and anything else that wants to use it.

It is not possible to set the pan position of a 3D channel.
Platform-specific
On Windows, this attribute has no effect when speaker assignment is used, except on Windows Vista and newer with the BASS_CONFIG_VISTA_SPEAKERS config option enabled. Balance control could be implemented via a DSP function instead.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_CONFIG_CURVE_PAN
BASS_ATTRIB_SCANINFO attribute

The scanned info of a channel.

```c
BASS_ChannelSetAttributeEx(
    DWORD handle,
    BASS_ATTRIB_SCANINFO,
    void *scaninfo,
    DWORD size
);```

**Parameters**

- handle  
  The channel handle.

- scaninfo  
  The scanned info.

- size  
  The size of the scanned info.
Remarks
This attribute is the information that is scanned from a file when the
BASS_STREAM_PRESCan flag is used in a BASS_StreamCreateFile call or
when the BASS_POS_SCAN flag is used with BASS_ChannelSetPosition. It is
supported on MP3/MP2/MP1 files and chained OGG files. It may be supported
by add-ons too; see the documentation.

The structure of the scanned info may change in future versions, so if the data is
stored, be prepared for BASS_ChannelSetAttributeEx to fail when trying to
apply it; the file can be scanned again if that happens.
Example
Transfer scanned info from one stream to another stream of the same file.

```c
DWORD size = BASS_ChannelGetAttributeEx(stream1, BASS_ATTRIB_SCANINFO, NULL, 0); // get the size
void *scaninfo = malloc(size); // allocate a buffer for the data
BASS_ChannelGetAttributeEx(stream1, BASS_ATTRIB_SCANINFO, scaninfo, size);
BASS_ChannelSetAttributeEx(stream2, BASS_ATTRIB_SCANINFO, scaninfo, size);
free(scaninfo);
```
See also
BASS_ChannelGetAttributeEx, BASS_ChannelSetAttributeEx
BASS_ATTRIB_SRC attribute

The sample rate conversion quality of a channel.

```c
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_SRC,
    float quality
);
```
**Parameters**

**handle**  The channel handle.

**quality**  The sample rate conversion quality... 0 = linear interpolation, 1 = 8 point sinc interpolation, 2 = 16 point sinc interpolation, 3 = 32 point sinc interpolation. Other values are also accepted but will be interpreted as 0 or 3, depending on whether they are lower or higher.
Remarks
When a channel has a different sample rate to what the output device is using, the channel's sample data will need to be converted to match the output device's rate during playback. This attribute determines how that is done. The linear interpolation option uses less CPU, but the sinc interpolation gives better sound quality (less aliasing), with the quality and CPU usage increasing with the number of points. A good compromise for lower spec systems could be to use sinc interpolation for music playback and linear interpolation for sound effects.

Whenever possible, a channel's sample rate should match the output device's rate to avoid the need for any sample rate conversion. The device's sample rate could be used in BASS_StreamCreate or BASS_MusicLoad or MIDI stream creation calls, for example.

The sample rate conversion occurs (when required) during playback, after the sample data has left the channel's playback buffer, so it does not affect the data delivered by BASS_ChannelGetData. Although this attribute has no direct effect on decoding channels, it is still available so that it can be used by the BASSmix add-on and anything else that wants to use it.

This attribute can be set at any time, and changes take immediate effect. A channel's initial setting is determined by the BASS_CONFIG_SRC config option, or BASS_CONFIG_SRC_SAMPLE in the case of a sample channel.
**Platform-specific**

On Windows, sample rate conversion is handled by Windows or the output device/driver rather than BASS, so this setting has no effect on playback there.
See also
BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_CONFIG_SRC, BASS_CONFIG_SRC_SAMPLE
BASS_ATTRIB_VOL attribute

The volume level of a channel.

```
BASS_ChannelSetAttribute(
    DWORD handle,
    BASS_ATTRIB_VOL,
    float volume
);
```
**Parameters**

**handle**  The channel handle.

**volume**  The volume level... 0 (silent) to 1.0 (full). This can go above 1.0 on decoding channels.
Remarks
This attribute applies to playback of the channel, and does not affect the channel's sample data, so has no real effect on decoding channels. It is still adjustable though, so that it can be used by the BASSmix add-on and anything else that wants to use it.

When using BASS_ChannelSlideAttribute to slide this attribute, a negative volume value can be used to fade-out and then stop the channel.
**See also**

BASS_ChannelGetAttribute, BASS_ChannelSetAttribute,
BASS_ChannelSlideAttribute, BASS_CONFIG_CURVE_VOL
DSPPROC callback

User defined DSP callback function.

```c
void CALLBACK DSPProc(
    HDSP handle,
    DWORD channel,
    void *buffer,
    DWORD length,
    void *user
);
```
**Parameters**

**handle**  The DSP handle.

**channel**  Channel that the DSP is being applied to.

**buffer**  Pointer to the sample data to apply the DSP to. The data is as follows: 8-bit samples are unsigned, 16-bit samples are signed, 32-bit floating-point samples range from -1 to +1 (not clipped, so can actually be outside this range).

**length**  The number of bytes to process.

**user**  The user instance data given when [BASS_ChannelSetDSP](https://www.fsource.com/bass/doc.php?f=BASS_ChannelSetDSP.htm) was called.
Remarks
A DSP function should be as quick as possible; playing streams and MOD
musics, and other DSP functions cannot be processed until it has finished.

Some functions can cause problems if called from within a DSP (or stream)
function. Do not call BASS_Stop or BASS_Free from within a DSP callback,
and do not call BASS_ChannelStop, BASS_MusicFree or BASS_StreamFree
with the same channel handle as received by the callback.

If the BASS_CONFIG_FLOATDSP config option is set, then DSP callback
functions will always be passed 32-bit floating-point sample data, regardless of
what the channels' actual sample format is.
Example
A simple DSP function to swap the left/right channels of a stereo 16-bit channel.

```c
void CALLBACK SwapDSP(HDSP handle, DWORD channel, void *buffer, DWORD length, void *user)
{
    short *s=buffer;
    for (; length; length-=4, s+=2) {
        short temp=s[0];
        s[0]=s[1];
        s[1]=temp;
    }
}
```

A panning/balance DSP function for a stereo 16-bit channel.

```c
float pan; // panning position, set as you would the BASS_ATTRIB_PAN attribute

void CALLBACK PanDSP(HDSP handle, DWORD channel, void *buffer, DWORD length, void *user)
{
    short *s=buffer;
    if (!pan) return; // no processing needed for centre panning
    for (; length; length-=4, s+=2) {
        if (pan<0) s[1]=s[1]*(1+pan); // pan left = reduce right
        else s[0]=s[0]*(1-pan); // vice versa
    }
}
See also
BASS_ChannelSetDSP, BASS_CONFIG_FLOATDSP, STREAMPROC callback
SYNCPROC callback

User defined synchronizer callback function.

```c
void CALLBACK SyncProc(
    HSYNC handle,
    DWORD channel,
    DWORD data,
    void *user
);
```
**Parameters**

handle  The sync that has occurred.
channel  The channel that the sync occurred on.
data  Additional data associated with the sync's occurrence.
user  The user instance data given when `BASS_ChannelSetSync` was called.
Remarks
BASS creates a single thread dedicated to executing sync callback functions, so a callback function should be quick as other syncs cannot be processed until it has finished. Attribute slides (BASS_ChannelSlideAttribute) are also performed by the sync thread, so are also affected if a sync callback takes a long time.

"Mixtime" syncs are not executed in the sync thread, but immediately in whichever thread triggers them. In most cases that will be an update thread, and so the same restrictions that apply to stream callbacks (STREAMPROC) also apply here, except that BASS_ChannelStop can be used in a BASS_SYNC_POS sync's callback to stop a channel at a particular position.

BASS_ChannelSetPosition can be used in a mixtime sync to implement custom looping, eg. set a BASS_SYNC_POS sync at the loop end position and seek to the loop start position in the callback.
See also
BASS_ChannelSetSync
BASS_CHANNELINFO structure

Used with BASS_ChannelGetInfo to retrieve information on a channel.

typedef struct {
    DWORD freq;
    DWORD chans;
    DWORD flags;
    DWORD ctype;
    DWORD origres;
    HPLUGIN plugin;
    HSAMPLE sample;
    char *filename;
} BASS_CHANNELINFO;
### Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>Default playback rate.</td>
</tr>
<tr>
<td>chans</td>
<td>Number of channels... 1=mono, 2=stereo, etc.</td>
</tr>
<tr>
<td>flags</td>
<td>A combination of these flags.</td>
</tr>
<tr>
<td>BASS_SAMPLE_8BITS</td>
<td>The channel's resolution is 8-bit. If this or the BASS_SAMPLE_FLOAT flags</td>
</tr>
<tr>
<td></td>
<td>are present, then the channel's resolution is 16-bit.</td>
</tr>
<tr>
<td>BASS_SAMPLE_FLOAT</td>
<td>The channel's resolution is 32-bit floating-point.</td>
</tr>
<tr>
<td>BASS_SAMPLE_LOOP</td>
<td>The channel is looped.</td>
</tr>
<tr>
<td>BASS_SAMPLE_3D</td>
<td>The channel has 3D functionality enabled.</td>
</tr>
<tr>
<td>BASS_SAMPLE_SOFTWARE</td>
<td>The channel is NOT using hardware mixing.</td>
</tr>
<tr>
<td>BASS_SAMPLE_VAM</td>
<td>The channel is using the DX7 voice allocation and management features</td>
</tr>
<tr>
<td></td>
<td>(HCHANNEL only)</td>
</tr>
<tr>
<td>BASS_SAMPLE_MUTEMAX</td>
<td>The channel is muted when at or beyond its max distance. (HCHANNEL)</td>
</tr>
<tr>
<td>BASS_SAMPLE_FX</td>
<td>The channel has the &quot;with FX flag&quot; effect implementation enabled. (HSTREAM/HMUSIC)</td>
</tr>
<tr>
<td>BASS_STREAM_RESTRATE</td>
<td>The internet file download rate is restricted. (HSTREAM)</td>
</tr>
<tr>
<td>BASS_STREAM_BLOCK</td>
<td>The internet file (or &quot;buffered&quot; user) file is streamed in small blocks. (HSTREAM/HMUSIC)</td>
</tr>
<tr>
<td>BASS_STREAM_AUTOFREE</td>
<td>The channel will automatically be freed when it ends. (HSTREAM/HMUSIC)</td>
</tr>
<tr>
<td>BASS_STREAM_DECODE</td>
<td>The channel is a &quot;decoding channel&quot; (HSTREAM/HMUSIC/HRECORD)</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMP</td>
<td>The MOD music is using &quot;normal&quot; ramping. (HMUSIC)</td>
</tr>
<tr>
<td>BASS_MUSIC_RAMPS</td>
<td>The MOD music is using &quot;sensitive&quot; ramping. (HMUSIC)</td>
</tr>
</tbody>
</table>
BASS_MUSIC_SURROUND  The MOD music is using surround sound. (HMUSIC)
BASS_MUSIC_SURROUND2  The MOD music is using surround sound mode 2. (HMUSIC)
BASS_MUSIC_NONINTER  The MOD music is using non-interpolated mixing. (HMUSIC)
BASS_MUSIC_FT2MOD  The MOD music is using FastTracker 2 .MOD playback. (HMUSIC)
BASS_MUSIC_PT1MOD  The MOD music is using ProTracker 1 .MOD playback. (HMUSIC)
BASS_MUSIC_STOPBACK  The MOD music will be stopped when a backward jump effect is played. (HMUSIC)
BASS_SPEAKER_xxx  Speaker assignment flags. (HSTREAM/HMUSIC)
BASS_UNICODE  filename is in UTF-16 form. Other flags may be supported by add-ons, see the documentation.

cctype  The type of channel it is, which can be one of the following.
BASS_CTYPE_SAMPLE  Sample channel. (HCHANNEL)
BASS_CTYPE_STREAM  User sample stream. This can also be used as a flag to test if the channel is any kind of HSTREAM.
BASS_CTYPE_STREAM_OGG  Ogg Vorbis format stream
BASS_CTYPE_STREAM_MP1  MPEG layer 1 format stream
BASS_CTYPE_STREAM_MP2  MPEG layer 2 format stream
BASS_CTYPE_STREAM_MP3  MPEG layer 3 format stream
BASS_CTYPE_STREAM_AIFF  Audio IFF format stream
BASS_CTYPE_STREAM_CA  CoreAudio codec stream. Additional format information available from BASS_ChannelGetTags (BASS_TAG_CA_CODEC).
BASS_CTYPE_STREAM_MF  Media Foundation codec stream
Additional format information is available from `BASS_ChannelGetTags` (BASS_TAG_WAVEFORMAT).

- **BASS_CTYPE_STREAM_WAV_PCM**: Integer PCM WAVE format stream.
- **BASS_CTYPE_STREAM_WAV_FLOAT**: Floating-point PCM WAVE format stream.
- **BASS_CTYPE_STREAM_WAV**: WAVE format flag. This can be used to test if the channel is any kind of WAVE format. The codec (the file's "wFormatTag") is specified in the LOWORD.
- **BASS_CTYPE_MUSIC_MOD**: Generic MOD format music. This can also be used as a flag to test if the channel is any kind of HMUSIC.
- **BASS_CTYPE_MUSIC_MTM**: MultiTracker format music.
- **BASS_CTYPE_MUSIC_S3M**: ScreamTracker 3 format music.
- **BASS_CTYPE_MUSIC_XM**: FastTracker 2 format music.
- **BASS_CTYPE_MUSIC_IT**: Impulse Tracker format music.
- **BASS_CTYPE_MUSIC_MO3**: MO3 format flag, used in combination with one of BASS_CTYPE_MUSIC_TYPES.

**BASS_CTYPE_RECORD**: Recording channel. (HRECORD)

Other channel types may be supported by add-ons, see the documentation.

- **origres**: The original resolution (bits per sample)... 0 = undefined.
- **plugin**: The plugin that is handling the channel... 0 = not using a plugin. Note only available with streams created using the plugin system via the standard BASS stream creation functions, not those created by add-on functions.
Information on the plugin can be retrieved via `BASS_PluginGetInfo`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample</td>
<td>The sample that is playing on the channel. (HCHANNEL only)</td>
<td></td>
</tr>
<tr>
<td>filename</td>
<td>The filename associated with the channel. (HSTREAM only)</td>
<td></td>
</tr>
</tbody>
</table>
Remarks
The BASS_SAMPLESOFTWARE flag indicates whether or not the channel's sample data is being mixed into the final output by the hardware. It does not indicate (in the case of a stream or MOD music) whether the processing required to generate the sample data is being done by the hardware; this processing is always done in software.

With a recording channel, the BASS_STREAM_DECODER flag indicates that it is not using a RECORDPROC callback function.

BASS supports 8/16/32-bit sample data, so if a WAV file, for example, uses another sample resolution, it will have to be converted by BASS. The origres member can be used to check what the resolution originally was.
Platform-specific
On Linux/OSX/iOS/Android, the BASS_UNICODE flag may not be present even if it was used in the stream's creation, as BASS will have translated the filename to the native UTF-8 form. On Windows CE, the opposite is true: the BASS_UNICODE flag may be present even if it was not used in the stream's creation, as BASS will have translated the filename to the native UTF-16 form.
Example
Check if a channel is an MP3 stream.

BASS_CHANNELINFO info;
BASS_ChannelGetInfo(channel, &info); // get info
if (info.cType==BASS_CTYPE_STREAM_MP3) {
    // it's an MP3!
}
See also

BASS_ChannelGetInfo
APEv2 binary tag structure.

typedef struct {
    char *key;
    void *data;
    DWORD length;
} TAG_APE_BINARY;
**Members**

key  The name of the tag.
data  The tag data.
length  The size of *data* in bytes.
See also
BASS_ChannelGetTags
TAG_BEXT structure

BWF "bext" tag structure.

typedef struct {
    char Description[256];
    char Originator[32];
    char OriginatorReference[32];
    char OriginationDate[10];
    char OriginationTime[8];
    QWORD TimeReference;
    WORD Version;
    BYTE UMID[64];
    BYTE Reserved[190];
    char CodingHistory[];
} TAG_BEXT;
### Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A free description of the sequence. To help applications which only display a short description, it is recommended that a summary of the description is contained in the first 64 characters, and the last 192 characters are used for details.</td>
</tr>
<tr>
<td>Originator</td>
<td>The name of the originator/producer of the audio file.</td>
</tr>
<tr>
<td>OriginatorReference</td>
<td>A non ambiguous reference allocated by the originating organization.</td>
</tr>
<tr>
<td>OriginationDate</td>
<td>The date of creation of the audio sequence, in the form of &quot;yyyy-mm-dd&quot; (year-month-day).</td>
</tr>
<tr>
<td>OriginationTime</td>
<td>The time of creation of the audio sequence, in the form of &quot;hh-mm-ss&quot; (hours-minutes-seconds).</td>
</tr>
<tr>
<td>TimeReference</td>
<td>The timecode of the sequence. The first sample count since midnight.</td>
</tr>
<tr>
<td>Version</td>
<td>The BWF version.</td>
</tr>
<tr>
<td>UMID</td>
<td>64 bytes containing a UMID (Unique Material Identifier) to the SPMTE 330M standard. If only a 32 byte &quot;basic UMID&quot; is used, the last 32 bytes should be set to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved for extensions.</td>
</tr>
<tr>
<td>CodingHistory</td>
<td>A series of CR/LF terminated strings, each containing a description of a coding process applied to the audio data.</td>
</tr>
</tbody>
</table>
Remarks
The structure is given by `BASS_ChannelGetTags` as it is in the RIFF file, which is little-endian, so the `TimeReference` and `Version` members will need to be reversed on big-endian platforms. The `UMID` member is only available with BWF version 1 (and above).

See the [BWF specification](#) for further details.
See also
BASS_ChannelGetTags, TAG_CART structure
BWF "cart" tag structure.

typedef struct {
    char Version[4];
    char Title[64];
    char Artist[64];
    char CutID[64];
    char ClientID[64];
    char Category[64];
    char Classification[64];
    char OutCue[64];
    char StartDate[10];
    char StartTime[8];
    char EndDate[10];
    char EndTime[8];
    char ProducerAppID[64];
    char ProducerAppVersion[64];
    char UserDef[64];
    DWORD dwLevelReference;
    TAG_CART_TIMER PostTimer[8];
    char Reserved[276];
    char URL[1024];
    char TagText[];
} TAG_CART;
<table>
<thead>
<tr>
<th><strong>Members</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Version of the data structure.</td>
</tr>
<tr>
<td>Title</td>
<td>Title of cart audio sequence.</td>
</tr>
<tr>
<td>Artist</td>
<td>Artist or creator name.</td>
</tr>
<tr>
<td>CutID</td>
<td>Cut number identification.</td>
</tr>
<tr>
<td>ClientID</td>
<td>Client identification.</td>
</tr>
<tr>
<td>Category</td>
<td>Category ID, PSA, NEWS, etc.</td>
</tr>
<tr>
<td>Classification</td>
<td>Classification or auxiliary key.</td>
</tr>
<tr>
<td>OutCue</td>
<td>Out cue text.</td>
</tr>
<tr>
<td>StartDate</td>
<td>Start date, in the form of &quot;yyyy-mm-dd&quot; (year-month-day).</td>
</tr>
<tr>
<td>StartTime</td>
<td>Start time, in the form of &quot;hh-mm-ss&quot; (hours-minutes-seconds).</td>
</tr>
<tr>
<td>EndDate</td>
<td>End date, in the form of &quot;yyyy-mm-dd&quot; (year-month-day).</td>
</tr>
<tr>
<td>EndTime</td>
<td>End time, in the form of &quot;hh-mm-ss&quot; (hours-minutes-seconds).</td>
</tr>
<tr>
<td>ProducerAppID</td>
<td>Name of vendor or application.</td>
</tr>
<tr>
<td>ProducerAppVersion</td>
<td>Version of producer application.</td>
</tr>
<tr>
<td>UserDef</td>
<td>User defined text.</td>
</tr>
<tr>
<td>dwLevelReference</td>
<td>Sample value for 0 dB reference.</td>
</tr>
<tr>
<td>PostTimer</td>
<td>8 time markers after head.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved for extensions.</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform resource locator.</td>
</tr>
<tr>
<td>TagText</td>
<td>Free form text for scripts or tags.</td>
</tr>
</tbody>
</table>
Remarks
The structure is given by BASS_ChannelGetTags as it is in the RIFF file, which is little-endian, so the dwLevelReference and PostTimer members will need to be reversed on big-endian platforms.

See the "CartChunk" specification for further details.
See also
BASS_ChannelGetTags, TAG_BEXT structure, TAG_CART_TIMER structure
TAG_CART_TIMER structure

BWF "cart" tag timer structure.

typedef struct {
    DWORD dwUsage;
    DWORD dwValue;
} TAG_CART_TIMER;
<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwUsage</td>
<td>FOURCC timer usage ID.</td>
</tr>
<tr>
<td>dwValue</td>
<td>Timer value in samples from head.</td>
</tr>
</tbody>
</table>
See also
TAG_CART structure
TAG_CA_CODEC structure

CoreAudio codec information structure.

typedef struct {
    DWORD ftype;
    DWORD atype;
    char *name;
} TAG_CA_CODEC;
Members
ftype  File format identifier.
atype  Audio data format identifier.
name  Description of the audio file format.
Remarks
A list of file and audio data format identifiers is available from Apple, here. Additional formats may be available via third-party codecs.
See also
BASS_ChannelGetTags
TAG_ID3 structure

ID3v1 tag structure.

typedef struct {
    char id[3];
    char title[30];
    char artist[30];
    char album[30];
    char year[4];
    char comment[30];
    BYTE genre;
} TAG_ID3;
**Members**

- **id**: ID3v1 tag identifier... "TAG".
- **title**: Song title.
- **artist**: Artist name.
- **album**: Album title.
- **year**: Year.
- **comment**: Comment. If the 30th character is non-null whilst the 29th character is null, then the 30th character is the track number and the comment is limited to the first 28 characters.
- **genre**: Genre number. The number can be translated to a genre, using the list at [www.id3.org](http://www.id3.org).
 Remarks
See www.id3.org/ID3v1 for further details.
**Example**

Display the title from a channel's ID3v1 tag.

```c
TAG_ID3 *id3=(TAG_ID3*)BASS_ChannelGetTags(channel, BASS_TAG_ID3);
if (id3) printf("title = %.30s\n", id3->title); // display the title
```
See also

BASS_ChannelGetTags
DX8 effect implementations

DX8 effects are otherwise known as DirectX Media Object (DMO) effects, and as the name suggests, requires DirectX 8 (or above) to be installed. BASS provides 2 different implementations of DX8 effects, each with its advantages. The method used by a channel depends on whether the BASS_SAMPLE_FX flag is used in its creation.
**With the BASS_SAMPLE_FX flag**

This is the standard way of using DX8 effects. The main advantage of this method is that effect parameter changes are audible instantaneously. The main disadvantages are that the channel's sample rate cannot be changed (can with DX9), and it cannot be used with decoding channels or speaker assignment.
Without the BASS_SAMPLE_FX flag
The advantages/disadvantages of this method are basically the opposite of the other method; the channel's sample rate can be changed, but there's a delay in effect parameter changes being audible. The reason being that, using this method, the effects are applied at the same stage as user DSP functions. There are also other advantages to this method, as shown in the table below.

<table>
<thead>
<tr>
<th>Adding &amp; removing</th>
<th>With FX flag</th>
<th>Without FX flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Channel needs to be stopped when adding or removing an effect.</td>
<td>Can add and remove effects without stopping playback.</td>
</tr>
<tr>
<td>Decoding channels</td>
<td>Not possible.</td>
<td>Automatically used for decoding channels.</td>
</tr>
<tr>
<td>Speaker assignment</td>
<td>Not possible.</td>
<td>Can be used with speaker assignment.</td>
</tr>
<tr>
<td>Recording</td>
<td>Not possible.</td>
<td>Automatically used for recording channels.</td>
</tr>
<tr>
<td>Parameter changes</td>
<td>Audible instantaneously.</td>
<td>Delayed by the length of the channel's buffer; using a smaller buffer means less delay.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel sample rate</th>
<th>With FX flag</th>
<th>Without FX flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Can only be changed when using DirectX 9.</td>
<td>Can be changed.</td>
</tr>
<tr>
<td>Effected sample data</td>
<td>Not available. DSP functions, BASS_ChannelGetData and BASS_ChannelGetLevel receive the original data (without the effects applied).</td>
<td>The effected data is available to BASS functions.</td>
</tr>
<tr>
<td>Effect chain ordering</td>
<td>Not possible.</td>
<td>The effects can be applied in any order you want, and can be intermingled with DSP functions.</td>
</tr>
<tr>
<td>Channel buffer length</td>
<td>Must be at least 150ms.</td>
<td>No restriction.</td>
</tr>
<tr>
<td>CPU usage</td>
<td>CPU use is not included in</td>
<td>CPU use is included in</td>
</tr>
</tbody>
</table>
BASS\_GetCPU. BASS\_GetCPU. Also slightly lower CPU usage.

In both cases, DX8 effects are not supported on channels that are more than stereo, and floating-point support requires DirectX 9.
**Platform-specific**
Away from Windows, the DX8 effects are emulated by BASS and the "With FX flag" system is unavailable. Floating-point is supported (8.24 fixed-point on Android and Windows CE), and the PARAMEQ effect also supports more than stereo.
See also
BASS_ChannelSetFX, BASS_MusicLoad, BASS_StreamCreate,
BASS_StreamCreateFile, BASS_StreamCreateURL
BASS_FXGetParameters

Retrieves the parameters of an effect.

```c
BOOL BASS_FXGetParameters(
    HFX handle,
    void *params
);
```
**Parameters**

**handle**  The effect handle.

**params**  Pointer to the parameters structure to fill. The structure used depends on the effect type.
**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 invalid.
See also
BASS_ChannelSetFX, BASS_FXSetParameters
BASS_FXReset

Resets the state of an effect or all effects on a channel.

```c
BOOL BASS_FXReset(
    DWORD handle
);
```
Parameters
handle The effect or channel handle... a HFX, HSTREAM, HMUSIC, or HRECORD.
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 invalid.
BASS_ERROR_UNKNOWN  Some other mystery problem!
Remarks
This function flushes the internal buffers of the effect(s). Effects are automatically reset by `BASS_ChannelSetPosition`, except when called from a "mixtime" `SYNCPROC`. 
See also

BASS_ChannelSetFX
**BASS_FXSetParameters**

Sets the parameters of an effect.

```c
BOOL BASS_FXSetParameters(
    HFX handle,
    void *params
);
```
Parameters

handle  The effect handle.
params  Pointer to the parameters structure. The structure used depends on the effect type.
**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 invalid.

BASS_ERROR_ILLPARAM  
One or more of the parameters are invalid, make sure all the values are within the valid ranges.

BASS_ERROR_UNKNOWN  
Some other mystery problem!
See also
BASS_ChannelSetFX, BASS_FXGetParameters
BASS_DX8_CHORUS structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 chorus effect.

typedef struct {
    float fWetDryMix;
    float fDepth;
    float fFeedback;
    float fFrequency;
    DWORD lWaveform;
    float fDelay;
    DWORD lPhase;
} BASS_DX8_CHORUS;
**Members**

- `fWetDryMix` Ratio of wet (processed) signal to dry (unprocessed) signal. Must be in the range from 0 through 100 (all wet). The default value is 50.

- `fDepth` Percentage by which the delay time is modulated by the low-frequency oscillator (LFO). Must be in the range from 0 through 100. The default value is 10.

- `fFeedback` Percentage of output signal to feed back into the effect's input, in the range from -99 to 99. The default value is 25.

- `fFrequency` Frequency of the LFO, in the range from 0 to 10. The default value is 1.1.

- `lWaveform` Waveform of the LFO... 0 = triangle, 1 = sine. By default, the waveform is sine.

- `fDelay` Number of milliseconds the input is delayed before it is played back, in the range from 0 to 20. The default value is 16 ms.

- `lPhase` Phase differential between left and right LFOs, one of BASS_DX8_PHASE_NEG_180, BASS_DX8_PHASE_NEG_90, BASS_DX8_PHASE_ZERO, BASS_DX8_PHASE_90 and BASS_DX8_PHASE_180. The default value is BASS_DX8_PHASE_90.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
BASS_DX8_COMPRESSOR structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 compression effect.

typedef struct {
    float fGain;
    float fAttack;
    float fRelease;
    float fThreshold;
    float fRatio;
    float fPredelay;
} BASS_DX8_COMPRESSOR;
**Members**

- **fGain**: Output gain of signal after compression, in the range from -60 to 60. The default value is 0 dB.

- **fAttack**: Time before compression reaches its full value, in the range from 0.01 to 500. The default value is 10 ms.

- **fRelease**: Speed at which compression is stopped after input drops below fThreshold, in the range from 50 to 3000. The default value is 200 ms.

- **fThreshold**: Point at which compression begins, in decibels, in the range from -60 to 0. The default value is -20 dB.

- **fRatio**: Compression ratio, in the range from 1 to 100. The default value is 3, which means 3:1 compression.

- **fPredelay**: Time after fThreshold is reached before attack phase is started, in milliseconds, in the range from 0 to 4. The default value is 4 ms.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
BASS_DX8_DISTORTION structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 distortion effect.

typedef struct {
    float fGain;
    float fEdge;
    float fPostEQCenterFrequency;
    float fPostEQBandwidth;
    float fPreLowpassCutoff;
} BASS_DX8_DISTORTION;
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fGain</td>
<td>Amount of signal change after distortion, in the range from -60 through 0. The default value is -18 dB.</td>
</tr>
<tr>
<td>fEdge</td>
<td>Percentage of distortion intensity, in the range from 0 through 100. The default value is 15 percent.</td>
</tr>
<tr>
<td>fPostEQCenterFrequency</td>
<td>Center frequency of harmonic content addition, in the range from 100 through 8000. The default value is 2400 Hz.</td>
</tr>
<tr>
<td>fPostEQBandwidth</td>
<td>Width of frequency band that determines range of harmonic content addition, in the range from 100 through 8000. The default value is 2400 Hz.</td>
</tr>
<tr>
<td>fPreLowpassCutoff</td>
<td>Filter cutoff for high-frequency harmonics attenuation, in the range from 100 through 8000. The default value is 8000 Hz.</td>
</tr>
</tbody>
</table>
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
BASS_DX8_ECHO structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 echo effect.

typedef struct {
    float fWetDryMix;
    float fFeedback;
    float fLeftDelay;
    float fRightDelay;
    BOOL lPanDelay;
} BASS_DX8_ECHO;
**Members**

- **fWetDryMix**: Ratio of wet (processed) signal to dry (unprocessed) signal. Must be in the range from 0 through 100 (all wet). The default value is 50.
- **fFeedback**: Percentage of output fed back into input, in the range from 0 through 100. The default value is 50.
- **fLeftDelay**: Delay for left channel, in milliseconds, in the range from 1 through 2000. The default value is 500 ms.
- **fRightDelay**: Delay for right channel, in milliseconds, in the range from 1 through 2000. The default value is 500 ms.
- **lPanDelay**: Value that specifies whether to swap left and right delays with each successive echo. The default value is FALSE, meaning no swap.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
**BASS_DX8_FLANGER structure**

Used with [BASS_FXGetParameters](#) and [BASS_FXSetParameters](#) to retrieve and set the parameters of a DX8 flanger effect.

```c
typedef struct {
    float fWetDryMix;
    float fDepth;
    float fFeedback;
    float fFrequency;
    DWORD lWaveform;
    float fDelay;
    DWORD lPhase;
} BASS_DX8_FLANGER;
```
Members

fWetDryMix  Ratio of wet (processed) signal to dry (unprocessed) signal. Must be in the range from 0 through 100 (all wet). The default value is 50.

fDepth  Percentage by which the delay time is modulated by the low-frequency oscillator (LFO). Must be in the range from 0 through 100. The default value is 100.

fFeedback  Percentage of output signal to feed back into the effect's input, in the range from -99 to 99. The default value is -50.

fFrequency  Frequency of the LFO, in the range from 0 to 10. The default value is 0.25.

lWaveform  Waveform of the LFO... 0 = triangle, 1 = sine. By default, the waveform is sine.

fDelay  Number of milliseconds the input is delayed before it is played back, in the range from 0 to 4. The default value is 2 ms.

lPhase  Phase differential between left and right LFOs, one of BASS_DX8_PHASE_NEG_180, BASS_DX8_PHASE_NEG_90, BASS_DX8_PHASE_ZERO, BASS_DX8_PHASE_90 and BASS_DX8_PHASE_180. The default value is BASS_DX8_PHASE_ZERO.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
BASS_DX8_GARGLE structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a gargle DX8 (amplitude modulation) effect.

typedef struct {
    DWORD dwRateHz;
    DWORD dwWaveShape;
} BASS_DX8_GARGLE;
**Members**

- **dwRateHz**  
  Rate of modulation, in Hertz. Must be in the range from 1 through 1000. The default value is 20.

- **dwWaveShape**  
  Shape of the modulation waveform... 0 = triangle, 1 = square.  
  By default, the waveform is triangle.
See also
BASS_ChannelSetFX, BASS_FXGetParameters,
BASS_DX8_I3DL2REVERB structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 I3DL2 (Interactive 3D Audio Level 2) reverberation effect.

```c
typedef struct {
    int lRoom;
    int lRoomHF;
    float flRoomRolloffFactor;
    float flDecayTime;
    float flDecayHFRatio;
    int lReflections;
    float flReflectionsDelay;
    int lReverb;
    float flReverbDelay;
    float flDiffusion;
    float flDensity;
    float flHFReference;
} BASS_DX8_I3DL2REVERB;
```
**Members**

**lRoom**  
Attenuation of the room effect, in millibels (mB), in the range from -10000 to 0. The default value is -1000 mB.

**lRoomHF**  
Attenuation of the room high-frequency effect, in mB, in the range from -10000 to 0. The default value is -100 mB.

**flRoomRolloffFactor**  
Rolloff factor for the reflected signals, in the range from 0 to 10. The default value is 0.0.

**flDecayTime**  
Decay time, in seconds, in the range from 0.1 to 20. The default value is 1.49 seconds.

**flDecayHFRatio**  
Ratio of the decay time at high frequencies to the decay time at low frequencies, in the range from 0.1 to 2. The default value is 0.83.

**lReflections**  
Attenuation of early reflections relative to lRoom, in mB, in the range from -10000 to 1000. The default value is -2602 mB.

**flReflectionsDelay**  
Delay time of the first reflection relative to the direct path, in seconds, in the range from 0 to 0.3. The default value is 0.007 seconds.

**lReverb**  
Attenuation of late reverberation relative to lRoom, in mB, in the range from -10000 to 2000. The default value is 200 mB.

**flReverbDelay**  
Time limit between the early reflections and the late reverberation relative to the time of the first reflection, in seconds, in the range from 0 to 0.1. The default value is 0.011 seconds.

**flDiffusion**  
Echo density in the late reverberation decay, in percent, in the range from 0 to 100. The default value is 100.0 percent.

**flDensity**  
Modal density in the late reverberation decay, in percent, in the range from 0 to 100. The default value is 100.0 percent.

**flHFReference**  
Reference high frequency, in hertz, in the range from 20 to 20000. The default value is 5000.0 Hz.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
BASS_DX8_PARAMEQ structure

Used with BASS_FXGetParameters and BASS_FXSetParameters to retrieve and set the parameters of a DX8 parametric equalizer effect.

typedef struct {
    float fCenter;
    float fBandwidth;
    float fGain;
} BASS_DX8_PARAMEQ;
Members

fCenter  Center frequency, in hertz.
fBandwidth  Bandwidth, in semitones, in the range from 1 to 36. The default value is 12.
fGain    Gain, in the range from -15 to 15. The default value is 0 dB.
**Platform-specific**
On Windows, $f_{Center}$ must be in the range of 80 to 16000, and not exceed one-third of the channel's sample rate. On other platforms, the range is above 0 and below half the channel's sample rate.
**See also**

BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
**BASS_DX8_REVERB structure**

Used with [BASS_FXGetParameters](#) and [BASS_FXSetParameters](#) to retrieve and set the parameters of a DX8 reverb effect.

```c
typedef struct {
    float fInGain;
    float fReverbMix;
    float fReverbTime;
    float fHighFreqRTRatio;
} BASS_DX8_REVERB;
```
**Members**

**fInGain**  
Input gain of signal, in decibels (dB), in the range from -96 through 0. The default value is 0 dB.

**fReverbMix**  
Reverb mix, in dB, in the range from -96 through 0. The default value is 0 dB.

**fReverbTime**  
Reverb time, in milliseconds, in the range from 0.001 through 3000. The default value is 1000.

**fHighFreqRTRatio**  
High-frequency reverb time ratio, in the range from 0.001 through 0.999. The default value is 0.001.
See also
BASS_ChannelSetFX, BASS_FXGetParameters, BASS_FXSetParameters
Floating-point channels

Channels can be made to use 32-bit floating-point sample data. When a channel uses floating-point sample data, BASS takes full advantage of the extra resolution when generating the decoded sample data; it does not simply convert 16-bit data to floating-point.

The main advantage of floating-point channels, aside from the increased resolution/quality, is that they are not clipped until output. This makes them particularly good for DSP/FX, because the quality is not degraded as the data passes through a chain of DSP/FX. So even if the output device is not capable of outputting the channel in its full quality, the quality is still improved.

Floating-point sample data ranges from -1 to +1, but as mentioned above, it is not clipped to this range until output, so it is possible that DSPPROC callback functions or BASS_ChannelGetData calls could receive data outside of this range.

When a floating-point channel is played, it is converted to whatever resolution the output device supports in the final mix.
**Platform-specific**

Floating-point channels are not supported when using VxD drivers (on Windows 98/95), except for "decoding channels". Windows Vista (and later) audio is natively floating-point, so for optimum performance (not to mention quality), BASS channels should also be floating-point. That particularly applies to formats that are decoded in floating-point anyway, ie. lossy formats like MP3 and OGG. It also applies to Linux and OSX, where BASS's mix format is floating-point.

Floating-point channels are not supported on Android or Windows CE. That includes "decoding channels".
**Example**
Check for 32-bit floating-point channel support.

```c
DWORD floatable; // floating-point channel support? 0 = no, else yes
...
floatable=BASS_StreamCreate(44100, 1, BASS_SAMPLE_FLOAT, NULL, NULL);
if (floatable) BASS_StreamFree(floatable); // floating-point channels supported!
```
See also
BASS_MusicLoad, BASS_StreamCreate, BASS_StreamCreateFile, BASS_StreamCreateURL
Most soundcards these days are capable of more than plain stereo output. To take advantage of this, HSTREAM and HMUSIC channels can be assigned to specific speakers. For example, channels can be played on the front or rear speakers to effectively have 2 separate stereo outputs from the one device. A 3rd stereo output is available on 5.1 cards, and a 4th on 7.1 cards. The `speakers` member of the `BASS_INFO` structure can be used to check how many speakers are available.
**Stereo speaker assignment flags**

- **BASS_SPEAKER_FRONT**: The front speakers.
- **BASS_SPEAKER_REAR**: The rear/side speakers.
- **BASS_SPEAKER_CENLFE**: The center and LFE (subwoofer) speakers in a 5.1 setup.
- **BASS_SPEAKER_REAR2**: The rear center speakers in a 7.1 setup.
Mono speaker assignment flags

- **BASS_SPEAKER_FRONTLEFT**  The left-front speaker.
- **BASS_SPEAKER_FRONTRIGHT** The right-front speaker.
- **BASS_SPEAKER_REARLEFT**  The left-rear/side speaker.
- **BASS_SPEAKER_REARRIGHT** The right-rear/side speaker.
- **BASS_SPEAKER_CENTER**  The center speaker in a 5.1 speaker setup.
- **BASS_SPEAKER_LFE**  The LFE (subwoofer) speaker in a 5.1 setup.
- **BASS_SPEAKER_REAR2LEFT**  The left-rear center speaker in a 7.1 setup.
- **BASS_SPEAKER_REAR2RIGHT** The right-rear center speaker in a 7.1 setup.

As well as these defined speaker location flags, there is the **BASS_SPEAKER_N(n)** macro that can be used to access the extra speakers of soundcards that have more than 8 speakers, where n is the n'th pair of speakers (up to a maximum of 15). For example, **BASS_SPEAKER_N(1)** is equivalent to **BASS_SPEAKER_FRONT**. To use a speaker in mono, add the **BASS_SPEAKER_LEFT** or **BASS_SPEAKER_RIGHT** flags.

The stereo speaker assignment flags can also be used with mono channels, so that, for example, a mono channel can be played on both the front speakers. But mono speaker assignment flags cannot be used with stereo channels, so, for example, it is not possible to play a stereo channel on just the center speaker.
See also
BASS_ChannelFlags, BASS_MusicLoad, BASS_StreamCreate, BASS_StreamCreateFile, BASS_StreamCreateURL, Multi-channel streams
Multi-channel streams

Most soundcards these days are capable of more than plain stereo output. To take advantage of this, as well as the speaker assignment flags, BASS has support for multi-channel user streams, and also has built-in support for multi-channel OGG, WAV and AIFF files. Add-ons provide support for other multi-channel formats.

When a stream having more channels than there are speakers is played, the extra channels will generally not be heard, but may be heard on other speakers instead in some cases on Windows. The chans member of the BASS_CHANNELINFO structure can be used to check how many channels a stream has, and the speakers member of the BASS_INFO structure can be used to check how many speakers there are.
**Platform-specific**
On Windows prior to Vista, multi-channel streams require the output device to have WDM drivers installed.
See also
BASS_StreamCreate, STREAMPROC callback
BASSCD

BASSCD enables digital streaming and ripping of audio CD tracks, and also includes analog playback support. It can be downloaded from the BASS website: www.un4seen.com

The BASSCD.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSCD documentation from within this BASS documentation.
BASSDSD enables the decoding and playing of DSD (Direct Stream Digital) files and streams. It can be downloaded from the BASS website: www.un4seen.com

The BASSDSD.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSDSD documentation from within this BASS documentation.
BASSFLAC enables the decoding and playing of FLAC (Free Lossless Audio Codec) encoded files and streams. It can be downloaded from the BASS website: www.un4seen.com

The BASSFLAC.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSFLAC documentation from within this BASS documentation.
BASSMIDI enables the playback of MIDI files, and also includes support for real-time events. It can be downloaded from the BASS website: 
www.un4seen.com

The BASSMIDI.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSMIDI documentation from within this BASS documentation.
BASSOPUS enables the decoding and playing of Opus encoded files. It can be downloaded from the BASS website: www.un4seen.com

The BASSOPUS.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSOPUS documentation from within this BASS documentation.
BASSWMA enables the playback of WMA files and streams, and also includes functions for WMA encoding and broadcasting. It can be downloaded from the BASS website: [www.un4seen.com](http://www.un4seen.com)

The BASSWMA.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSWMA documentation from within this BASS documentation.
BASSWV

BASSWV enables the decoding and playing of WavPack encoded files. It can be downloaded from the BASS website: www.un4seen.com

The BASSWV.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSWV documentation from within this BASS documentation.
BASSenc allows BASS channels to be encoded using command-line encoders with STDIN support (LAME/OGGENC/etc), or ACM codecs (on Windows) or CoreAudio codecs (on OSX/iOS), and can serve the encoded data to directly connecting clients, or send it to Shoutcast and Icecast servers. It can also write plain PCM/WAV files. It can be downloaded from the BASS website: www.un4seen.com

The BASSENCHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSenc documentation from within this BASS documentation.
BASSmix provides channel mixing capability, with resampling and matrix mixing features. It can be downloaded from the BASS website: www.un4seen.com

The BASSMIX.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASSmix documentation from within this BASS documentation.
BASSWASAPI provides the ability to use WASAPI output and input on
Windows Vista and beyond, including support for both exclusive and shared
WASAPI modes. It can be downloaded from the BASS website:
www.un4seen.com

The BASSWASAPI.CHM file should be copied into the same directory as this
file (BASS.CHM), to be able to conveniently access the BASSWASAPI
documentation from within this BASS documentation.
BASS_FX provides several effects, including tempo & pitch control. It can be downloaded from the BASS website: www.un4seen.com

The BASS_FX.CHM file should be copied into the same directory as this file (BASS.CHM), to be able to conveniently access the BASS_FX documentation from within this BASS documentation.
**BASS_CONFIG_DEV_NONSTOP config option**

Do not stop the output device when nothing is playing on it?

```c
BASS_SetConfig(
    BASS_CONFIG_DEV_NONSTOP,
    BOOL nonstop
);
```
**Parameters**

nonstop  If TRUE, sample data will continue to be sent to the output device while nothing is playing.
Remarks
By default, BASS will stop sending data to the output device when nothing is playing, to save a little CPU. When that happens, the device buffer will become empty, and the next playback will begin more quickly as a result. If more consistent playback latency (around the value given by \texttt{BASS\_GetInfo}) is wanted, this option can be enabled to keep the device buffer filled with silence when nothing is playing. The output will still be stopped by \texttt{BASS\_Stop}.
Platform-specific
The output stopping when nothing is playing only applies on Linux, Android, and Windows CE. So this config option is only available on those platforms.
See also
BASS_GetConfig, BASS_SetConfig, BASS_CONFIG_DEV_BUFFER