

# BASS\_CONFIG\_MIDI\_AUTOFONT config option

---

Automatically load matching soundfonts?

```
BASS_SetConfig(  
    BASS_CONFIG_MIDI_AUTOFONT,  
    DWORD autofont  
);
```

**Parameters**

autofont If set to 1, BASSMIDI will try to load a soundfont matching the MIDI file. If set to 2, the matching soundfont will also be used on all banks.

## Remarks

When enabled, BASSMIDI will check for a matching soundfont when loading a MIDI file. For example, if the MIDI file is "afile.mid", then it will look for a soundfont named "afile.sf2" or "afile.mid.sf2". If found, this soundfont will take precedence over any other soundfonts applied via [BASS MIDI StreamSetFonts](#).

When the soundfont is used on all banks (*autofont=2*), BASSMIDI will first check for the specific bank and preset in the soundfont, but if it does not find that, it will just look for a preset match on any bank.

This option only applies to local MIDI files, loaded using [BASS MIDI StreamCreateFile](#) (or [BASS StreamCreateFile](#) via the plugin system). BASSMIDI will not look for matching soundfonts for MIDI files loaded from the internet.

By default, this option is set to 1.

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#)

[BASS\\_GetConfig](#), [BASS\\_SetConfig](#)

# BASS\_CONFIG\_MIDI\_COMPACT config option

---

Automatically compact all soundfonts following a configuration change?

```
BASS_SetConfig(  
    BASS_CONFIG_MIDI_COMPACT,  
    BOOL compact  
);
```

**Parameters**

compact If TRUE, all soundfonts are compacted following a MIDI stream being freed, or a [BASS\\_MIDI\\_StreamSetFonts](#) call.

## Remarks

The compacting is not performed immediately upon a MIDI stream being freed or [BASS\\_MIDI\\_StreamSetFont](#)s being called, but rather 2 seconds later (in another thread), so that if another MIDI stream immediately starts using the soundfonts, they are not needlessly closed and reopened.

Samples that have been preloaded by [BASS\\_MIDI\\_FontLoad](#) are not affected by automatic compacting. Other samples that have been preloaded by [BASS\\_MIDI\\_StreamLoadSamples](#) are affected though, so it is probably wise to disable this option when using that function.

By default, this option is enabled.

**See also**

[BASS\\_MIDI\\_FontCompact](#)

[BASS\\_GetConfig](#), [BASS\\_SetConfig](#)

# BASS\_CONFIG\_MIDI\_DEFFONT config option

---

The default soundfont.

```
BASS_SetConfigPtr(  
    BASS_CONFIG_MIDI_DEFFONT,  
    char *filename  
);
```

**Parameters**

filename    Filename of the soundfont... NULL = no default soundfont.

**Remarks**

A copy is made of the provided filename, so it does not need to persist beyond the [BASS\\_SetConfigPtr](#) call. If the specified soundfont cannot be loaded, the default soundfont setting will remain as it is. [BASS\\_GetConfigPtr](#) can be used to confirm what that is.

**Platform-specific**

On Windows, the BASS\_UNICODE flag can be used to set/get the soundfont filename in UTF-16 form, otherwise it is ANSI. The filename is in UTF-8 form on other platforms. On Windows, the default is to use one of the Creative soundfonts (28MBGM.SF2 or CT8MGM.SF2 or CT4MGM.SF2 or CT2MGM.SF2) if present in the Windows system directory.

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#)

[BASS\\_GetConfigPtr](#), [BASS\\_SetConfigPtr](#)

# BASS\_CONFIG\_MIDI\_IN\_PORTS config option

---

The number of MIDI input ports to make available.

```
BASS_SetConfig(  
    BASS_CONFIG_MIDI_IN_PORTS,  
    DWORD ports  
);
```

## **Parameters**

ports Number of input ports... 0 (min) - 10 (max).

**Remarks**

MIDI input ports allow MIDI data to be received from other software, not only MIDI devices. Once a port has been initialized via [BASS\\_MIDI\\_InInit](#), the ALSA client and port IDs can be retrieved from [BASS\\_MIDI\\_InGetDeviceInfo](#), which other software can use to connect to the port and send data to it. Prior to initialization, an input port will have a client ID of 0.

The default is for 1 input port to be available.

**Platform-specific**

This option is only available on Linux.

**See also**

[BASS\\_MIDI\\_InGetDeviceInfo](#), [BASS\\_MIDI\\_InInit](#)

[BASS\\_GetConfig](#), [BASS\\_SetConfig](#)

# BASS\_CONFIG\_MIDI\_VOICES config option

---

The maximum number of samples to play at a time.

```
BASS_SetConfig(  
    BASS_CONFIG_MIDI_VOICES,  
    DWORD voices  
);
```

## **Parameters**

voices Maximum number of samples to play at a time... 1 (min) - 100000 (max).

## Remarks

This setting determines the maximum number of samples that can play together in a single MIDI stream. This is not necessarily the same thing as the maximum number of notes, due to presets possibly containing multiple layered samples, ie. multiple samples may be played for a single note. When the voice limit is hit, the voice with the lowest volume level will be killed, which will usually be one that is already fading out following a note release.

The more voices that are used, the more CPU that is required. So this option can be used to restrict that, for example on a lower spec system. The [BASS\\_ATTRIB\\_MIDI\\_CPU](#) attribute can also be used to limit CPU usage.

Changing this setting only affects subsequently created MIDI streams, not any that already exist. The voice limit of an existing MIDI stream can be changed via the [BASS\\_ATTRIB\\_MIDI\\_VOICES](#) attribute.

**Platform-specific**

The default setting is 100, except on Android and iOS where it is 40, and Windows CE where it is 30. The maximum setting is 1000 on those 3 platforms.

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_ATTRIB\\_MIDI\\_CPU](#),  
[BASS\\_ATTRIB\\_MIDI\\_VOICES](#)

[BASS\\_GetConfig](#), [BASS\\_SetConfig](#)

# BASS\_MIDI\_StreamCreate

---

Creates a sample stream to render real-time MIDI events.

```
HSTREAM BASS_MIDI_StreamCreate(  
    DWORD channels,  
    DWORD flags,  
    DWORD freq  
);
```

## Parameters

channels	The number of MIDI channels... 1 (min) - 128 (max). The number of channels can subsequently be changed via the <a href="#">BASS_ATTRIB_MIDI_CHANS</a> attribute.
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 <a href="#">Floating-point channels</a> for more info.
BASS_SAMPLE_MONO	Decode/play the MIDI in mono (uses less CPU than stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling <a href="#">BASS_Init</a> .
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 <a href="#">BASS_Init</a> . 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 <a href="#">DX8 effect implementations</a> section for details. Use <a href="#">BASS_ChannelSetFX</a> to add effects to the stream.
BASS_STREAM_AUTOFREE	Automatically free the stream

BASS_STREAM_DECODE	when playback ends. Decode/render the sample data, without playing it. Use <a href="#">BASS_ChannelGetData</a> 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_xxx	<a href="#">Speaker assignment flags</a> . The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.
BASS_MIDI_NOFX	Disable reverb and chorus processing, saving some CPU time. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOSYSRESET	Ignore system reset events (MIDI_EVENT_SYSTEM) when the system mode is unchanged. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOTEOFF1	Only release the oldest instance upon a note off event (MIDI_EVENT_NOTE with velocity=0) when there are overlapping instances of the note. Otherwise all instances are released. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .

**BASS\_MIDI\_SINCINTER** Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. Otherwise linear interpolation is used. The sinc interpolation uses 8 points by default, but 16 point sinc interpolation is also available via the [BASS\\_ATTRIB\\_MIDI\\_SRC](#) attribute.

**freq** Sample rate to render/play the MIDI stream 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 new stream's handle is returned, else 0 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

## Error codes

BASS_ERROR_INIT	<a href="#">BASS_Init</a> 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	<i>channels</i> is not valid.
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 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

This function creates a stream solely for real-time MIDI events. As it is not based on any file, the stream has no predetermined length and is never-ending. Seeking is not possible, but it is possible to reset everything, including playback buffer, by calling [BASS\\_ChannelPlay](#) (*restart = TRUE*) or [BASS\\_ChannelSetPosition](#) (*pos = 0*).

MIDI events are applied using the [BASS\\_MIDI\\_StreamEvent](#) and [BASS\\_MIDI\\_StreamEvents](#) functions. If the stream is being played (it is not a decoding channel), then there will be some delay in the effect of the events being heard. This latency can be reduced by making use of the [BASS\\_CONFIG\\_BUFFER](#) and [BASS\\_CONFIG\\_UPDATEPERIOD](#) config options.

If a stream has 16 MIDI channels, then channel 10 defaults to percussion/drums and the rest melodic, otherwise they are all melodic. That can be changed using [BASS\\_MIDI\\_StreamEvent](#) and the MIDI\_EVENT\_DRUMS event.

Soundfonts provide the sounds that are used to render a MIDI stream. A default soundfont configuration is applied initially to the new MIDI stream, which can subsequently be overridden using [BASS\\_MIDI\\_StreamSetFont](#).

To play a MIDI file, use [BASS\\_MIDI\\_StreamFromFile](#).

**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 Android and iOS, sinc interpolation requires a NEON-supporting CPU; the BASS\_MIDI\_SINCINTER flag will otherwise be ignored. Sinc interpolation is not available on Windows CE.

**See also**

[BASS\\_MIDI\\_StreamCreateEvents](#), [BASS\\_MIDI\\_StreamCreateFile](#),  
[BASS\\_MIDI\\_StreamEvent](#), [BASS\\_MIDI\\_StreamEvents](#),  
[BASS\\_MIDI\\_StreamGetChannel](#), [BASS\\_MIDI\\_StreamSetFonts](#),  
[BASS\\_ATTRIB\\_MIDI\\_CPU](#), [BASS\\_ATTRIB\\_MIDI\\_SRC](#),  
[BASS\\_CONFIG\\_MIDI\\_DEFFONT](#), [BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetInfo](#), [BASS\\_ChannelPlay](#), [BASS\\_ChannelSetAttribute](#),  
[BASS\\_ChannelSetDSP](#), [BASS\\_ChannelSetFX](#), [BASS\\_ChannelSetLink](#),  
[BASS\\_StreamFree](#)

# BASS\_MIDI\_StreamCreateEvents

---

Creates a sample stream from a sequence of MIDI events.

```
HSTREAM BASS_MIDI_StreamCreateEvents(  
    BASS\_MIDI\_EVENT *events,  
    DWORD ppqn,  
    DWORD flags,  
    DWORD freq  
);
```

## Parameters

events	Pointer to an array containing the event sequence to play.
ppqn	The Pulses Per Quarter Note, or ticks per beat.
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 <a href="#">Floating-point channels</a> for more info.
BASS_SAMPLE_MONO	Decode/play the MIDI in mono (uses less CPU than stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling <a href="#">BASS_Init</a> .
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 <a href="#">BASS_Init</a> . 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_LOOP	Loop the events. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_SAMPLE_FX	Enable the old implementation of DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details. Use <a href="#">BASS_ChannelSetFX</a> to add effects to the stream.

BASS_STREAM_AUTOFREE	Automatically free the stream when playback ends.
BASS_STREAM_DECODE	Decode/render the sample data, without playing it. Use <a href="#">BASS_ChannelGetData</a> 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_xxx	<a href="#">Speaker assignment flags</a> . The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.
BASS_MIDI_DECAYEND	Let the sound decay naturally (including reverb) instead of stopping abruptly at the end of the events, including when looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_DECAYSEEK	Let the old sound decay naturally (including reverb) when changing the position, including looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> , but it should generally only be used in <a href="#">BASS_ChannelSetPosition</a> calls to have it applied to particular position changes, eg. custom loops.
BASS_MIDI_NOFX	Disable reverb and chorus processing, saving some CPU time. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .

BASS_MIDI_NOSYSRESET	Ignore system reset events (MIDI_EVENT_SYSTEM) when the system mode is unchanged. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOTEOFF1	Only release the oldest instance upon a note off event (MIDI_EVENT_NOTE with velocity=0) when there are overlapping instances of the note. Otherwise all instances are released. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_SINCINTER	Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. Otherwise linear interpolation is used. The sinc interpolation uses 8 points by default, but 16 point sinc interpolation is also available via the <a href="#">BASS_ATTRIB_MIDI_SRC</a> attribute.
freq	Sample rate to render/play the MIDI stream at... 0 = the rate specified in the <a href="#">BASS_Init</a> call, 1 = the device's current output rate (or the <a href="#">BASS_Init</a> rate if that is not available).

**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	<a href="#">BASS_Init</a> 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	<i>ppqn</i> cannot be 0.
BASS_ERROR_ILLTYPE	The event sequence contains invalid event type(s).
BASS_ERROR_POSITION	The events must be in chronological order (within each track).
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 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

This function creates a 16 channel MIDI stream to play a predefined sequence of MIDI events. Any of the standard MIDI events listed in the [BASS\\_MIDI\\_StreamEvent](#) section can be used, but the MIDI\_EVENT\_SYSTEMEX and the "non-MIDI" events (eg. MIDI\_EVENT\_MIXLEVEL) events are not available and will be rejected. The sequence should end with a MIDI\_EVENT\_END event. Multiple tracks are possible via the MIDI\_EVENT\_END\_TRACK event, which signals the end of a track; the next event will be in a new track.

The event sequence is copied, so the *events* array does not need to persist beyond the function call.

Soundfonts provide the sounds that are used to render a MIDI stream. A default soundfont configuration is applied initially to the new MIDI stream, which can subsequently be overridden using [BASS\\_MIDI\\_StreamSetFont](#).

**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 Android and iOS, sinc interpolation requires a NEON-supporting CPU; the BASS\_MIDI\_SINCINTER flag will otherwise be ignored. Sinc interpolation is not available on Windows CE.

## Example

Play a middle C note (key 60) on a violin every 2 seconds.

```
BASS_MIDI_EVENT events[]={
    {MIDI_EVENT_TEMPO, 500000, 0, 0}, // set the tempo to 0.5 seconds
    {MIDI_EVENT_PROGRAM, 40, 0, 0}, // select the violin preset
    {MIDI_EVENT_NOTE, MAKEWORD(60, 100), 0, 0}, // press the key
    {MIDI_EVENT_NOTE, 60, 0, 200}, // release the key after 200 ticks
    {MIDI_EVENT_END, 0, 0, 400} // end after 400 ticks
};
HSTREAM stream=BASS_MIDI_StreamCreateEvents(events, 100, BASS_SAMPLERATE);
BASS_ChannelPlay(stream, 0); // start playing it
```

**See also**

[BASS MIDI StreamCreate](#), [BASS MIDI StreamCreateFile](#),  
[BASS MIDI StreamGetChannel](#), [BASS MIDI StreamLoadSamples](#),  
[BASS MIDI StreamSetFonts](#), [BASS\\_ATTRIB MIDI CPU](#),  
[BASS\\_ATTRIB MIDI SRC](#), [BASS\\_CONFIG MIDI DEFFONT](#),  
[BASS\\_CONFIG MIDI VOICES](#)

[BASS\\_ChannelGetInfo](#), [BASS\\_ChannelPlay](#), [BASS\\_ChannelSetAttribute](#),  
[BASS\\_ChannelSetDSP](#), [BASS\\_ChannelSetFX](#), [BASS\\_ChannelSetLink](#),  
[BASS\\_StreamFree](#)

# BASS\_MIDI\_StreamCreateFile

---

Creates a sample stream from a MIDI file.

```
HSTREAM BASS_MIDI_StreamCreateFile(  
    BOOL mem,  
    void *file,  
    QWORD offset,  
    QWORD length,  
    DWORD flags,  
    DWORD freq  
);
```

## Parameters

mem	TRUE = stream the file from memory.
file	Filename (mem = FALSE) or a memory location (mem = TRUE).
offset	File offset to begin streaming from (only used if mem = FALSE).
length	Data length... 0 = use all data up to the end of the 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 <a href="#">Floating-point channels</a> for more info.
BASS_SAMPLE_MONO	Decode/play the MIDI in mono (uses less CPU than stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling <a href="#">BASS_Init</a> .
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 <a href="#">BASS_Init</a> . 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_LOOP	Loop the file. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_SAMPLE_FX	Enable the old implementation of

	DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details. Use <a href="#">BASS_ChannelSetFX</a> to add effects to the stream.
BASS_STREAM_AUTOFREE	Automatically free the stream when playback ends.
BASS_STREAM_DECODE	Decode/render the sample data, without playing it. Use <a href="#">BASS_ChannelGetData</a> 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_xxx	<a href="#">Speaker assignment flags</a> . The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.
BASS_MIDI_DECAYEND	Let the sound decay naturally (including reverb) instead of stopping abruptly at the end of the file, including when looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_DECAYSEEK	Let the old sound decay naturally (including reverb) when changing the position, including looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> , but it should generally only be used in <a href="#">BASS_ChannelSetPosition</a> calls to have it applied to particular position changes, eg. custom loops.

BASS_MIDI_NOCROP	Do not remove empty space (containing no events) from the end of the file.
BASS_MIDI_NOFX	Disable reverb and chorus processing, saving some CPU time. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOSYSRESET	Ignore system reset events (MIDI_EVENT_SYSTEM) when the system mode is unchanged. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOTEOFF1	Only release the oldest instance upon a note off event (MIDI_EVENT_NOTE with velocity=0) when there are overlapping instances of the note. Otherwise all instances are released. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_SINCINTER	Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. Otherwise linear interpolation is used. The sinc interpolation uses 8 points by default, but 16 point sinc interpolation is also available via the <a href="#">BASS_ATTRIB_MIDI_SRC</a> attribute.
BASS_UNICODE	<i>file</i> 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 MIDI stream at... 0 = the rate specified in the <a href="#">BASS_Init</a> call, 1 = the device's current output rate (or the <a href="#">BASS_Init</a> rate if that is not available).

**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	<a href="#">BASS_Init</a> 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 <i>length</i> 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_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 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

BASSMIDI supports standard MIDI format 0/1/2 files. In the case of format 2, the tracks are rendered/played one after another. RIFF MIDI (RMID) files are also supported. The General MIDI standard events are supported, as are several Roland GS and Yamaha XG NRPN and SysEx events. A full list of supported MIDI events can be found in the [BASS\\_MIDI\\_StreamEvent](#) documentation.

Soundfonts provide the sounds that are used to render a MIDI stream. A default soundfont configuration is applied initially to the new MIDI stream, which can subsequently be overridden using [BASS\\_MIDI\\_StreamSetFont](#). By default, with the [BASS\\_CONFIG\\_MIDI\\_AUTOFONT](#) config option enabled, BASSMIDI will also check for a soundfont of the same name as the MIDI file. Note that a MIDI stream can have multiple soundfonts stacked, each providing different presets, for example.

As well as the standard byte/time-based positioning, MIDI tick-based positioning is also supported. The `BASS_POS_MIDI_TICK` "mode" can be used with [BASS\\_ChannelGetLength](#), [BASS\\_ChannelGetPosition](#) and [BASS\\_ChannelSetPosition](#) to deal in ticks. The [BASS\\_ATTRIB\\_MIDI\\_PPQN](#) attribute can be used to translate the position to beats.

Marker, instrument name, cue, text, and lyric events can be retrieved via the [BASS\\_MIDI\\_StreamGetMark](#) function. [Syncs](#) can also be used to be notified of their occurrence.

Other texts of each track (eg. track name) are available via the `BASS_TAG_MIDI_TRACK+<track>` tag, where *track=0* is the first track. A pointer to a series of null-terminated strings is given, the final string ending with a double null. The first text in the first track is generally the title of the MIDI file. RIFF MIDI tags are also available via the standard [BASS\\_TAG\\_RIFF\\_INFO](#) tag.

Alongside the events played from the MIDI file, custom MIDI events can be applied via the [BASS\\_MIDI\\_StreamEvent](#) and [BASS\\_MIDI\\_StreamEvents](#) functions. The events can be played on the same MIDI channels that are used by the MIDI file, or they can be played on additional channels allocated via the [BASS\\_ATTRIB\\_MIDI\\_CHANS](#) attribute.

Unlike with most stream formats, the entire MIDI file is loaded to memory. This means the file can be deleted or moved after calling this function, or the memory can be discarded (*mem = TRUE*).

To play a MIDI file from the internet, use [BASS\\_MIDI\\_StreamCreateURL](#). To play a custom sequence of MIDI events, [BASS\\_MIDI\\_StreamCreateEvents](#) can be used.

**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 Android and iOS, sinc interpolation requires a NEON-supporting CPU; the BASS\_MIDI\_SINCINTER flag will otherwise be ignored. Sinc interpolation is not available on Windows CE.

## Example

Create a stream of a MIDI file, with a sample rate of 44100hz.

```
HSTREAM stream=BASS_MIDI_StreamCreateFile(FALSE, "afile.mid", 0, 0,
```

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamCreateEvents](#),  
[BASS\\_MIDI\\_StreamCreateFileUser](#), [BASS\\_MIDI\\_StreamCreateURL](#),  
[BASS\\_MIDI\\_StreamGetChannel](#), [BASS\\_MIDI\\_StreamGetEvents](#),  
[BASS\\_MIDI\\_StreamGetMark](#), [BASS\\_MIDI\\_StreamLoadSamples](#),  
[BASS\\_MIDI\\_StreamSetFont](#), [BASS\\_ATTRIB\\_MIDI\\_CPU](#),  
[BASS\\_ATTRIB\\_MIDI\\_SRC](#), [BASS\\_CONFIG\\_MIDI\\_AUTOFONT](#),  
[BASS\\_CONFIG\\_MIDI\\_DEFFONT](#), [BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetInfo](#), [BASS\\_ChannelGetLength](#), [BASS\\_ChannelGetTags](#),  
[BASS\\_ChannelPlay](#), [BASS\\_ChannelSetAttribute](#), [BASS\\_ChannelSetDSP](#),  
[BASS\\_ChannelSetFX](#), [BASS\\_ChannelSetLink](#), [BASS\\_StreamFree](#)

# BASS\_MIDI\_StreamCreateFileUser

---

Creates a sample stream from a MIDI file via user callback functions.

```
HSTREAM BASS_MIDI_StreamCreateFileUser(  
    DWORD system,  
    DWORD flags,  
    BASS\_FILEPROCS *procs,  
    void *user,  
    DWORD freq  
);
```

## Parameters

system	File system to use, which must be <code>STREAMFILE_NOBUFFER</code> , as the entire MIDI file is preloaded.
flags	Any combination of these flags.
<code>BASS_SAMPLE_8BITS</code>	Use 8-bit resolution. If neither this or the <code>BASS_SAMPLE_FLOAT</code> flags are specified, then the sample data will be 16-bit.
<code>BASS_SAMPLE_FLOAT</code>	Use 32-bit floating-point sample data. See <a href="#">Floating-point channels</a> for more info.
<code>BASS_SAMPLE_MONO</code>	Decode/play the MIDI in mono (uses less CPU than stereo). This flag is automatically applied if <code>BASS_DEVICE_MONO</code> was specified when calling <a href="#">BASS_Init</a> .
<code>BASS_SAMPLE_SOFTWARE</code>	Force the stream to not use hardware mixing.
<code>BASS_SAMPLE_3D</code>	Enable 3D functionality. This requires that the <code>BASS_DEVICE_3D</code> flag was specified when calling <a href="#">BASS_Init</a> . 3D channels must also be mono, so <code>BASS_SAMPLE_MONO</code> is automatically applied. The <code>SPEAKER</code> flags cannot be used together with this flag.
<code>BASS_SAMPLE_LOOP</code>	Loop the file. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
<code>BASS_SAMPLE_FX</code>	Enable the old implementation of DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details. Use <a href="#">BASS_ChannelSetFX</a> to add effects to the stream.

BASS_STREAM_AUTOFREE	Automatically free the stream when playback ends.
BASS_STREAM_DECODE	Decode/render the sample data, without playing it. Use <a href="#">BASS_ChannelGetData</a> 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_xxx	<a href="#">Speaker assignment flags</a> . The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.
BASS_MIDI_DECAYEND	Let the sound decay naturally (including reverb) instead of stopping abruptly at the end of the file, including when looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_DECAYSEEK	Let the old sound decay naturally (including reverb) when changing the position, including looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> , but it should generally only be used in <a href="#">BASS_ChannelSetPosition</a> calls to have it applied to particular position changes, eg. custom loops.
BASS_MIDI_NOCROP	Do not remove empty space (containing no events) from the end of the file.

BASS_MIDI_NOFX	Disable reverb and chorus processing, saving some CPU time. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOSYSRESET	Ignore system reset events (MIDI_EVENT_SYSTEM) when the system mode is unchanged. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOTEOFF1	Only release the oldest instance upon a note off event (MIDI_EVENT_NOTE with velocity=0) when there are overlapping instances of the note. Otherwise all instances are released. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_SINCINTER	Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. Otherwise linear interpolation is used. The sinc interpolation uses 8 points by default, but 16 point sinc interpolation is also available via the <a href="#">BASS_ATTRIB_MIDI_SRC</a> attribute.

procs	The user defined file functions.
user	User instance data to pass to the callback functions.
freq	Sample rate to render/play the MIDI stream at... 0 = the rate specified in the <a href="#">BASS_Init</a> call, 1 = the device's current output rate (or the <a href="#">BASS_Init</a> rate if that is not available).

**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	<a href="#">BASS_Init</a> 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	<i>system</i> is not valid.
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 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 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**

As there is no file associated with a user file stream, it is not possible for BASSMIDI to look for a soundfont with the same name as the MIDI file. If there is a matching soundfont, it can be applied using [BASS\\_MIDI\\_StreamSetFont](#).

**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 Android and iOS, sinc interpolation requires a NEON-supporting CPU; the BASS\_MIDI\_SINCINTER flag will otherwise be ignored. Sinc interpolation is not available on Windows CE.

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamCreateEvents](#),  
[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamCreateURL](#),  
[BASS\\_MIDI\\_StreamGetChannel](#), [BASS\\_MIDI\\_StreamGetMark](#),  
[BASS\\_MIDI\\_StreamLoadSamples](#), [BASS\\_MIDI\\_StreamSetFont](#),  
[BASS\\_ATTRIB\\_MIDI\\_CPU](#), [BASS\\_ATTRIB\\_MIDI\\_SRC](#),  
[BASS\\_CONFIG\\_MIDI\\_DEFFONT](#), [BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetInfo](#), [BASS\\_ChannelGetLength](#), [BASS\\_ChannelGetTags](#),  
[BASS\\_ChannelPlay](#), [BASS\\_ChannelSetAttribute](#), [BASS\\_ChannelSetDSP](#),  
[BASS\\_ChannelSetFX](#), [BASS\\_ChannelSetLink](#), [BASS\\_StreamFree](#),  
[BASS\\_FILEPROCS](#) structure

# BASS\_MIDI\_StreamCreateURL

---

Creates a sample stream from an MIDI file on the internet, optionally receiving the downloaded data in a callback.

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

## Parameters

url	URL of the file to stream. Should begin with "http://" or "ftp://".
offset	File position to start streaming from. This is ignored by some servers, specifically when the file length is unknown.
flags	Any 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 <a href="#">Floating-point channels</a> for more info.
BASS_SAMPLE_MONO	Decode/play the MIDI in mono (uses less CPU than stereo). This flag is automatically applied if BASS_DEVICE_MONO was specified when calling <a href="#">BASS_Init</a> .
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 <a href="#">BASS_Init</a> . 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_LOOP	Loop the file. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_SAMPLE_FX	Enable the old implementation of DirectX 8 effects. See the <a href="#">DX8 effect implementations</a> section for details. Use <a href="#">BASS_ChannelSetFX</a> to add

BASS_STREAM_STATUS	effects to the stream. Pass status info (HTTP/ICY tags) from the server to the <a href="#">DOWNLOADPROC</a> 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/render the sample data, without playing it. Use <a href="#">BASS_ChannelGetData</a> 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_xxx	<a href="#">Speaker assignment flags</a> . The BASS_SAMPLE_MONO flag is automatically applied when using a mono speaker assignment flag.
BASS_MIDI_DECAYEND	Let the sound decay naturally (including reverb) instead of stopping abruptly at the end of the file, including when looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_DECAYSEEK	Let the old sound decay naturally (including reverb) when changing the position, including looping. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> , but it should generally only be used in <a href="#">BASS_ChannelSetPosition</a> calls to

	have it applied to particular position changes, eg. custom loops.
BASS_MIDI_NOCROP	Do not remove empty space (containing no events) from the end of the file.
BASS_MIDI_NOFX	Disable reverb and chorus processing, saving some CPU time. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOSYSRESET	Ignore system reset events (MIDI_EVENT_SYSTEM) when the system mode is unchanged. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_NOTEOFF1	Only release the oldest instance upon a note off event (MIDI_EVENT_NOTE with velocity=0) when there are overlapping instances of the note. Otherwise all instances are released. This flag can be toggled at any time using <a href="#">BASS_ChannelFlags</a> .
BASS_MIDI_SINCINTER	Use sinc interpolated sample mixing. This increases the sound quality, but also requires more CPU. Otherwise linear interpolation is used. The sinc interpolation uses 8 points by default, but 16 point sinc interpolation is also available via the <a href="#">BASS_ATTRIB_MIDI_SRC</a> attribute.
BASS_UNICODE	<i>url</i> is in UTF-16 form. Otherwise it is ANSI on Windows or Windows CE, and UTF-8 on other platforms.
proc	Callback function to receive the downloaded file... NULL = no callback.

user User instance data to pass to the callback function.

freq Sample rate to render/play the MIDI stream 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 new stream's handle is returned, else 0 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

## Error codes

BASS_ERROR_INIT	<a href="#">BASS_Init</a> 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_NONET	No internet connection could be opened.
BASS_ERROR_ILLPARAM	<i>url</i> is not a valid URL.
BASS_ERROR_TIMEOUT	The server did not respond to the request within the timeout period, as set with the <a href="#">BASS_CONFIG_NET_TIMEOUT config option</a> .
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 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 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**

The entire MIDI file is preloaded, so the standard `BASS_STREAM_BLOCK` and `BASS_STREAM_RESTRATE` flags have no effect here.

Regardless of the [BASS\\_CONFIG\\_MIDI\\_AUTOFONT](#) setting, a matching soundfont is not looked for when opening a MIDI file from a URL.

**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 Android and iOS, sinc interpolation requires a NEON-supporting CPU; the BASS\_MIDI\_SINCINTER flag will otherwise be ignored. Sinc interpolation is not available on Windows CE.

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamCreateFileUser](#),  
[BASS\\_MIDI\\_StreamGetChannel](#), [BASS\\_MIDI\\_StreamGetMark](#),  
[BASS\\_MIDI\\_StreamLoadSamples](#), [BASS\\_MIDI\\_StreamSetFonts](#),  
[BASS\\_ATTRIB\\_MIDI\\_CPU](#), [BASS\\_ATTRIB\\_MIDI\\_SRC](#),  
[BASS\\_CONFIG\\_MIDI\\_DEFFONT](#), [BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetInfo](#), [BASS\\_ChannelGetLength](#), [BASS\\_ChannelGetTags](#),  
[BASS\\_ChannelPlay](#), [BASS\\_ChannelSetAttribute](#), [BASS\\_ChannelSetDSP](#),  
[BASS\\_ChannelSetFX](#), [BASS\\_ChannelSetLink](#), [BASS\\_StreamFree](#),  
[DOWNLOADPROC](#) callback, [BASS\\_CONFIG\\_NET\\_AGENT](#),  
[BASS\\_CONFIG\\_NET\\_PROXY](#), [BASS\\_CONFIG\\_NET\\_TIMEOUT](#)

# BASS\_MIDI\_StreamEvent

---

Applies an event to a channel in a MIDI stream.

```
BOOL BASS_MIDI_StreamEvent(  
    HSTREAM handle,  
    DWORD chan,  
    DWORD event,  
    DWORD param  
);
```

## Parameters

- handle The MIDI stream to apply the event to.  
chan The MIDI channel to apply the event to... 0 = channel 1.  
event The event to apply, see the table below.  
param The event parameter.

## Event types, with *param* definitions.

MIDI_EVENT_NOTE	Press or release a key, or stop without sustain/decay. <b>param</b> : LOBYTE = key number (0-127, 60=middle C), HIBYTE = velocity (0=release, 1-127=press, 255=stop).
MIDI_EVENT_PROGRAM	Select the preset/instrument to use. Standard soundfont presets follow the <a href="#">General MIDI</a> standard, and generally also include Roland GS variations in other banks (accessible via the MIDI_EVENT_BANK event). <b>param</b> : preset number (0-65535).
MIDI_EVENT_CHANPRES	Set the channel pressure. <b>param</b> : pressure level (0-127).
MIDI_EVENT_KEYPRES	Set a key's pressure/aftertouch. <b>param</b> : LOBYTE = key number (0-127), HIBYTE = pressure level (0-127).
MIDI_EVENT_PITCH	Set the pitch wheel position. <b>param</b> : pitch wheel position (0-16383, 8192=normal/middle).
MIDI_EVENT_BANK	Select the bank to use (MIDI controller 0). <b>param</b> : bank number MSB (0-127).
MIDI_EVENT_MODULATION	Set the modulation (MIDI controller 1)

MIDI_EVENT_PORTATIME	<i>param</i> : modulation level (0-127). Set the portamento time (MIDI controller 5).
MIDI_EVENT_VOLUME	<i>param</i> : portamento time (0-127). Set the volume (MIDI controller 7).
MIDI_EVENT_PAN	<i>param</i> : volume level (0-127). Set the pan position (MIDI controller 10).
MIDI_EVENT_EXPRESSION	<i>param</i> : pan position (0-128, 0=left, 64=middle, 127=right, 128=random). Set the expression (MIDI controller 11).
MIDI_EVENT_BANK_LSB	<i>param</i> : expression level (0-127). Select the bank LSB to use (MIDI controller 32).
MIDI_EVENT_SUSTAIN	<i>param</i> : bank number LSB (0-127). Set the sustain pedal/switch (MIDI controller 64).
MIDI_EVENT_PORTAMENTO	<i>param</i> : sustain is on? (0-63=no, 64-127=yes). Set the portamento switch (MIDI controller 65).
MIDI_EVENT_SOSTENUTO	<i>param</i> : portamento is on? (0-63=no, 64-127=yes). Set the sostenuto pedal/switch (MIDI controller 66).
MIDI_EVENT_SOFT	<i>param</i> : sostenuto is on? (0-63=no, 64-127=yes). Set the soft pedal/switch (MIDI controller 67).
MIDI_EVENT_RESONANCE	<i>param</i> : soft is on? (0-63=no, 64-127=yes). Set the low-pass filter resonance (MIDI controller 71, NRPN 121h)

MIDI_EVENT_RELEASE	<p><b>param</b> : resonance level (0-127, 0=-64, 64=normal, 127=+63).</p> <p>Set the release time (MIDI controller 72, NRPN 166h)</p> <p><b>param</b> : release time (0-127, 0=-64, 64=normal, 127=+63).</p>
MIDI_EVENT_ATTACK	<p>Set the attack time (MIDI controller 73, NRPN 163h)</p> <p><b>param</b> : attack time (0-127, 0=-64, 64=normal, 127=+63).</p>
MIDI_EVENT_CUTOFF	<p>Set the low-pass filter cutoff (MIDI controller 74, NRPN 120h)</p> <p><b>param</b> : cutoff level (0-127, 0=-64, 64=normal, 127=+63).</p>
MIDI_EVENT_DECAY	<p>Set the decay time (MIDI controller 75, NRPN 164h)</p> <p><b>param</b> : decay time (0-127, 0=-64, 64=normal, 127=+63).</p>
MIDI_EVENT_PORTANOTE	<p>Set the portamento start key; the next note starts at this key (MIDI controller 84).</p> <p><b>param</b> : key number (1-127, 60=middle C).</p>
MIDI_EVENT_REVERB	<p>Set the reverb send level (MIDI controller 91)</p> <p><b>param</b> : reverb level (0-127).</p>
MIDI_EVENT_CHORUS	<p>Set the chorus send level (MIDI controller 93)</p> <p><b>param</b> : chorus level (0-127).</p>
MIDI_EVENT_USERFX	<p>Set the user effect send level (MIDI controller 94). This will have no audible effect unless custom processing is applied to the user effect mix via <a href="#">BASS_MIDI_StreamGetChannel</a>.</p> <p><b>param</b> : user effect level (0-127).</p>

MIDI_EVENT_SOUNDOFF	Stop all sounds (MIDI controller 120). <b>param</b> : not used.
MIDI_EVENT_RESET	Reset controllers (MIDI controller 121), that is modulation=0, expression=127, sustain=0, portamento=0, release time=64, attack time=64, pitch wheel=8192, channel pressure=0. <b>param</b> : not used.
MIDI_EVENT_NOTESOFF	Release all keys (MIDI controller 123). <b>param</b> : not used.
MIDI_EVENT_MODE	Set poly/mono mode (MIDI controllers 126 & 127). <b>param</b> : mode (0=poly, 1=mono, 2=legato).
MIDI_EVENT_CONTROL	Unhandled controller. This has no effect on the MIDI stream, but can be useful for custom processing purposes with a <a href="#"><u>BASS_SYNC_MIDI_EVENT</u></a> sync. <b>param</b> : LOBYTE = controller number, HIBYTE = controller value.
MIDI_EVENT_PITCHRANGE	Set pitch wheel range (MIDI RPN 0). <b>param</b> : range in semitones.
MIDI_EVENT_FINETUNE	Set the fine tuning (MIDI RPN 1). <b>param</b> : finetune in cents (0-16383, 0=-100, 8192=normal, 16383=+100).
MIDI_EVENT_COARSETUNE	Set the coarse tuning (MIDI RPN 2). <b>param</b> : finetune in semitones (0-

	127, 0=-64, 64=normal, 127=+63).
MIDI_EVENT_DRUMS	Set the percussion/drums channel switch. The bank and program are reset to 0 when this changes. <b>param</b> : use drums? (0=no, 1=yes).
MIDI_EVENT_DRUM_CUTOFF	Set the low-pass filter cutoff of a drum key (MIDI NRPN 14knh) <b>param</b> : LOBYTE = key number (0-127), HIBYTE = cutoff level (0-127, 0=-64, 64=normal, 127=+63).
MIDI_EVENT_DRUM_RESONANCE	Set the low-pass filter resonance of a drum key (MIDI NRPN 15knh) <b>param</b> : LOBYTE = key number (0-127), HIBYTE = resonance level (0-127, 0=-64, 64=normal, 127=+63).
MIDI_EVENT_DRUM_COARSETUNE	Set the coarse tuning of a drum key (MIDI NRPN 18knh). <b>param</b> : LOBYTE = key number (0-127), HIBYTE = finetune in semitones (0-127, 0=-64, 64=normal, 127=+63).
MIDI_EVENT_DRUM_FINETUNE	Set the fine tuning of a drum key (MIDI NRPN 19knh). <b>param</b> : LOBYTE = key number (0-127), HIBYTE = finetune in cents (0-127, 0=-100, 64=normal, 127=+100).
MIDI_EVENT_DRUM_LEVEL	Set the level of a drum key (MIDI NRPN 1Aknh) <b>param</b> : LOBYTE = key number (0-127), HIBYTE = level (0-127, 127=normal/full).
MIDI_EVENT_DRUM_PAN	Set the pan position of a drum key (MIDI NRPN 1Cknh).

	<p><b>param</b> : LOBYTE = key number (0-127), HIBYTE = pan position (0-128, 0=random, 1=left, 64=middle, 127=right, 128=normal).</p>
MIDI_EVENT_DRUM_REVERB	<p>Set the reverb send level of a drum key (MIDI NRPN 1Dknh)</p> <p><b>param</b> : LOBYTE = key number (0-127), HIBYTE = reverb level (0-127, 127=normal/full).</p>
MIDI_EVENT_DRUM_CHORUS	<p>Set the chorus send level of a drum key (MIDI NRPN 1Eknh)</p> <p><b>param</b> : LOBYTE = key number (0-127), HIBYTE = chorus level (0-127, 127=normal/full).</p>
MIDI_EVENT_DRUM_USERFX	<p>Set the user effect send level of a drum key (MIDI NRPN 1Fknh)</p> <p><b>param</b> : LOBYTE = key number (0-127), HIBYTE = user effect level (0-127, 127=normal/full).</p>
MIDI_EVENT_SCALETUNING	<p>Set the tuning of a note in every octave.</p> <p><b>param</b> : LOWORD = tuning change in cents (0-16383, 0=-100, 8192=normal, 16383=+100), HIWORD = note (0-11, 0=C).</p>
MIDI_EVENT_MOD_FILTER	<p>Set the maximum effect of modulation (MIDI controller 1) on filter cutoff.</p> <p><b>param</b> : filter cutoff effect in cents (0=-9600, 9600=none, 19200=+9600).</p>
MIDI_EVENT_MOD_PITCH	<p>Set the maximum effect of modulation (MIDI controller 1) on pitch.</p> <p><b>param</b> : pitch effect in semitones (0=-24, 24=none, 48=+24).</p>

MIDI_EVENT_MOD_VIBRATO	Set the maximum effect of modulation (MIDI controller 1) on vibrato depth (MIDI RPN 5). <b>param</b> : vibrato depth effect in cents (0=none, 128=100, 256=200, etc).
MIDI_EVENT_MOD_VOLUME	Set the maximum effect of modulation (MIDI controller 1) on volume. <b>param</b> : volume effect percentage (0=-100, 100=none, 200=+100).
MIDI_EVENT_CHANPRES_FILTER	Set the maximum effect of channel pressure on filter cutoff. <b>param</b> : filter cutoff effect in cents (0=-9600, 9600=none, 19200=+9600).
MIDI_EVENT_CHANPRES_PITCH	Set the maximum effect of channel pressure on pitch. <b>param</b> : pitch effect in semitones (0=-24, 24=none, 48=+24).
MIDI_EVENT_CHANPRES_VIBRATO	Set the maximum effect of channel pressure on vibrato depth. <b>param</b> : vibrato depth effect in cents (0=none, 128=100, 256=200, etc).
MIDI_EVENT_CHANPRES_VOLUME	Set the maximum effect of channel pressure on volume. <b>param</b> : volume effect percentage (0=-100, 100=none, 200=+100).
MIDI_EVENT_KEYPRES_FILTER	Set the maximum effect of key pressure/aftertouch on filter cutoff. <b>param</b> : filter cutoff effect in cents (0=-9600, 9600=none, 19200=+9600).
MIDI_EVENT_KEYPRES_PITCH	Set the maximum effect of key pressure/aftertouch on pitch. <b>param</b> : pitch effect in semitones

MIDI\_EVENT\_KEYPRES\_VIBRATO

(0=-24, 24=none, 48=+24).

Set the maximum effect of key pressure/aftertouch on vibrato depth.

**param** : vibrato depth effect in cents (0=none, 128=100, 256=200, etc).

MIDI\_EVENT\_KEYPRES\_VOLUME

Set the maximum effect of key pressure/aftertouch on volume.

**param** : volume effect percentage (0=-100, 100=none, 200=+100).

## Global events.

MIDI\_EVENT\_SYSTEM,  
MIDI\_EVENT\_SYSTEMEX

Set the system mode, resetting everything to the system's defaults. MIDI\_SYSTEM\_DEFAULT is basically identical to MIDI\_SYSTEM\_GS, except that channel 10 is melodic if there are not 16 channels.

MIDI\_EVENT\_SYSTEM does not reset things in any additional channels allocated to a MIDI file stream via the

[BASS\\_ATTRIB\\_MIDI\\_CHANS](#) attribute, while

MIDI\_EVENT\_SYSTEMEX does. If the system mode is unchanged and the BASS\_MIDI\_NOSYSRESET flag is set on the MIDI stream, then MIDI\_EVENT\_SYSTEM has no effect (MIDI\_EVENT\_SYSTEMEX still does).

***param*** : system mode  
(MIDI\_SYSTEM\_DEFAULT,  
MIDI\_SYSTEM\_GM1,  
MIDI\_SYSTEM\_GM2,  
MIDI\_SYSTEM\_GS,  
MIDI\_SYSTEM\_XG).

MIDI\_EVENT\_TEMPO

Set the tempo (MIDI meta event 81). Changing the tempo affects the stream length, and the

[BASS\\_ChannelGetLength](#) byte value will no longer be valid.

***param*** : tempo in microseconds per quarter note.

MIDI_EVENT_MASTERVOL	Set the master volume. <b>param</b> : volume level (0-16383, 0=silent, 16383=normal/full).
MIDI_EVENT_REVERB_TIME	Set the reverb time. <b>param</b> : reverb time in milliseconds.
MIDI_EVENT_REVERB_DELAY	Set the reverb delay. <b>param</b> : reverb delay in 10ths of a millisecond.
MIDI_EVENT_REVERB_LOCUTOFF	Set the reverb low-pass cutoff <b>param</b> : reverb low-pass cutoff in hertz (0=off).
MIDI_EVENT_REVERB_HICUTOFF	Set the reverb high-pass cutoff <b>param</b> : reverb high-pass cutoff in hertz (0=off).
MIDI_EVENT_REVERB_LEVEL	Set the reverb level <b>param</b> : reverb level (0=off, 100=0dB, 200=+6dB).
MIDI_EVENT_CHORUS_DELAY	Set the chorus delay. <b>param</b> : chorus delay in 10ths of a millisecond.
MIDI_EVENT_CHORUS_DEPTH	Set the chorus depth. <b>param</b> : chorus depth in 10ths of a millisecond.
MIDI_EVENT_CHORUS_RATE	Set the chorus rate. <b>param</b> : chorus rate in 100ths of a hertz.
MIDI_EVENT_CHORUS_FEEDBACK	Set the chorus feedback level. <b>param</b> : chorus feedback level (0=-100%, 100=off, 200=+100%).
MIDI_EVENT_CHORUS_LEVEL	Set the chorus level. <b>param</b> : chorus level (0=off, 100=0dB, 200=+6dB).
MIDI_EVENT_CHORUS_REVERB	Set the chorus send to reverb level. <b>param</b> : chorus send to reverb level (0=off, 100=0dB, 200=+6dB).

MIDI\_EVENT\_USERFX\_LEVEL

Set the user effect level.

***param*** : user effect level (0=off, 100=0dB, 200=+6dB).

MIDI\_EVENT\_USERFX\_CHORUS

Set the user effect send to chorus level.

***param*** : user effect send to chorus level (0=off, 100=0dB, 200=+6dB).

MIDI\_EVENT\_USERFX\_REVERB

Set the user effect send to reverb level.

***param*** : user effect send to reverb level (0=off, 100=0dB, 200=+6dB).

## Other (non-MIDI) events.

- MIDI\_EVENT\_MIXLEVEL Set the level.  
*param* : the level (0=silent, 100=0dB, 200=+6dB).
- MIDI\_EVENT\_TRANSPOSE Transpose all notes. Changes take effect from the next note played, and affect melodic channels only (not drum channels).  
*param* : transposition amount in semitones (0=-100, 100=normal, 200=+100).

**Other (non-MIDI)  
global events.**

MIDI\_EVENT\_SPEED Set a tempo modification. Changing the tempo affects the stream length, and the [BASS\\_ChannelGetLength](#) byte value will no longer be valid. The modification does not affect seeking.  
***param*** : speed in 100ths of a percent (100=1%/min, 10000=100%/normal, 20000=200%).

**Return value**

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

## **Error codes**

BASS_ERROR_HANDLE	<i>handle</i> is not valid.
BASS_ERROR_ILLPARAM	One of the other parameters is invalid.
BASS_ERROR_NOTAVAIL	Tempo does not apply to streams created with <a href="#"><u>BASS_MIDI_StreamCreate</u></a> .

## Remarks

Apart from the "global" events, all events apply only to the specified MIDI channel.

Except for the "non-MIDI" events, events applied to a MIDI file stream can subsequently be overridden by events in the file itself, and will also be overridden when seeking or looping. That can be avoided by using additional channels, allocated via the [BASS\\_ATTRIB\\_MIDI\\_CHANS](#) attribute.

[BASS\\_SYNC\\_MIDI\\_EVENT](#) syncs are not triggered by this function. If sync triggering is wanted, [BASS\\_MIDI\\_StreamEvents](#) can be used instead.

If the MIDI stream is being played (it is not a decoding channel), then there will be some delay in the effect of the event being heard. This latency can be reduced by making use of the [BASS\\_CONFIG\\_BUFFER](#) and [BASS\\_CONFIG\\_UPDATEPERIOD](#) config options when creating the stream.

If multiple events need to be applied at the same time, [BASS\\_MIDI\\_StreamEvents](#) can be used instead of this function.

## Example

Play the middle C note (key 60) with a velocity of 100, on channel 1 for 2 seconds.

```
BASS_MIDI_StreamEvent(handle, 0, MIDI_EVENT_NOTE, MAKEWORD(60, 100))  
Sleep(2000); // wait 2 seconds  
BASS_MIDI_StreamEvent(handle, 0, MIDI_EVENT_NOTE, 60); // release t
```

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamEvents](#),  
[BASS\\_MIDI\\_StreamGetEvent](#)

# BASS\_MIDI\_StreamEvents

---

Applies any number of events to a MIDI stream.

```
DWORD BASS_MIDI_StreamEvents(  
    HSTREAM handle,  
    DWORD mode,  
    void *events,  
    DWORD length  
);
```

## Parameters

handle	The MIDI stream to apply the events to.
mode	The type of event data to apply. One of the following, with optional flag
BASS_MIDI_EVENTS_RAW	Raw MIDI event data, as would be sent to a MIDI device. To overcome the 16 channel limit the event data's channel information can optionally be overridden by adding the new channel number to this parameter, where +1 = the 1st channel.
BASS_MIDI_EVENTS_STRUCT	An array of <a href="#">BASS_MIDI_EVENT</a> structures.
BASS_MIDI_EVENTS_CANCEL	Flag: Cancel pending events from a previous call of this function.
BASS_MIDI_EVENTS_NORSTATUS	Flag: Disable running status, meaning each event must include a status byte. Only applicable with BASS_MIDI_EVENTS_RAW
BASS_MIDI_EVENTS_SYNC	Flag: Trigger <a href="#">BASS_SYNC_MIDI_EVENT</a> syncs for the processed events
BASS_MIDI_EVENTS_TIME	Flag: The raw MIDI data includes delta-time info (when used with BASS_MIDI_EVENTS_RAW or the <a href="#">BASS_MIDI_EVENT</a> <i>tick</i> and <i>pos</i> members should be processed (without BASS_MIDI_EVENTS_RAW). The <a href="#">BASS_MIDI_EVENT</a> <i>tic</i>

and *pos* members will otherwise be ignored.

events The event data.

length The length of the event data. The number of bytes or BASS\_MIDI\_EVENT structures, depending on the type of event data.

**Return value**

If successful, the number of events processed 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\_ILLPARAM   *mode* is not valid.

## Remarks

Events applied to a MIDI file stream can subsequently be overridden by events in the file itself, and will also be overridden when seeking or looping. That can be avoided by using additional channels, allocated via the [BASS\\_ATTRIB\\_MIDI\\_CHANS](#) attribute.

The `BASS_MIDI_EVENTS_TIME` flag allows events to be delayed. With raw MIDI data, it enables delta-time info to be included in the data. With an [BASS\\_MIDI\\_EVENT](#) array, it enables processing of the *tick* and *pos* members (only one should be used). In both cases, the values are relative to the current decoding position, ie. 0 = no delay. The events do not necessarily need to be provided in chronological order; they will be sorted automatically. Tick-based delays are ignored on streams created with [BASS\\_MIDI\\_StreamCreate](#) because the Pulses Per Quarter Note ([BASS\\_ATTRIB\\_MIDI\\_PPQN](#)) is undefined; [BASS\\_MIDI\\_StreamCreateEvents](#) could be used instead when tick-based delays are wanted.

If the MIDI stream is being played (it is not a decoding channel), then there will be some delay in the effect of the event being heard even if no delay is requested in the event data. This latency can be reduced by making use of the [BASS\\_CONFIG\\_BUFFER](#) and [BASS\\_CONFIG\\_UPDATEPERIOD](#) config options when creating the stream.

## Example

Play a C major chord with a velocity of 100, on channel 1 for 2 seconds.

```
BASS_MIDI_EVENT events[3];
memset(events, 0, sizeof(events));
events[0].event=MIDI_EVENT_NOTE;
events[0].param=MAKWORD(60, 100); // C
events[1].event=MIDI_EVENT_NOTE;
events[1].param=MAKWORD(64, 100); // E
events[2].event=MIDI_EVENT_NOTE;
events[2].param=MAKWORD(67, 100); // G
BASS_MIDI_StreamEvents(handle, BASS_MIDI_EVENTS_STRUCT, events, 3);
Sleep(2000); // wait 2 seconds
// modify the event data to release the keys
events[0].param=MAKWORD(60, 0); // release C
events[1].param=MAKWORD(64, 0); // release E
events[2].param=MAKWORD(67, 0); // release G
BASS_MIDI_StreamEvents(handle, BASS_MIDI_EVENTS_STRUCT, events, 3);
```

The same thing using raw MIDI event data.

```
BYTE events[7]={0x90, 60, 100, 64, 100, 67, 100}; // the event data
BASS_MIDI_StreamEvents(handle, BASS_MIDI_EVENTS_RAW, events, 7); //
Sleep(2000); // wait 2 seconds
// modify the event data to release the keys
events[2]=0; // release C
events[4]=0; // release E
events[6]=0; // release G
BASS_MIDI_StreamEvents(handle, BASS_MIDI_EVENTS_RAW, events, 7); //
```

The same thing in a single event sequence, making use of *pos* to delay the release.

```
BASS_MIDI_EVENT events[6];
memset(events, 0, sizeof(events));
events[0].event=MIDI_EVENT_NOTE;
events[0].param=MAKWORD(60, 100); // C
events[1].event=MIDI_EVENT_NOTE;
events[1].param=MAKWORD(64, 100); // E
events[2].event=MIDI_EVENT_NOTE;
events[2].param=MAKWORD(67, 100); // G
events[3].event=MIDI_EVENT_NOTE;
events[3].param=MAKWORD(60, 0); // release C
events[3].pos=BASS_ChannelSeconds2Bytes(handle, 2); // 2 seconds
events[4].event=MIDI_EVENT_NOTE;
```

```
events[4].param=MAKEWORD(64, 0); // release E
events[4].pos=events[3].pos;
events[5].event=MIDI_EVENT_NOTE;
events[5].param=MAKEWORD(67, 0); // release G
events[5].pos=events[3].pos;
BASS_MIDI_StreamEvents(handle, BASS_MIDI_EVENTS_STRUCT|BASS_MIDI_EV
```

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamEvent](#),  
[BASS\\_MIDI\\_StreamGetEvent](#)

# BASS\_MIDI\_StreamGetChannel

---

Retrieves a MIDI channel's stream handle, which can be used to set DSP/FX on it. Can also replace the default reverb and chorus processing.

```
HSTREAM BASS_MIDI_StreamGetChannel(  
    HSTREAM handle,  
    DWORD chan  
);
```

## Parameters

handle The MIDI stream.

chan The MIDI channel... 0 = channel 1. Or one of the following special channels.

BASS\_MIDI\_CHAN\_CHORUS Chorus mix channel. The default chorus processing is replaced by the stream's processing.

BASS\_MIDI\_CHAN\_REVERB Reverb mix channel. The default reverb processing is replaced by the stream's processing.

BASS\_MIDI\_CHAN\_USERFX User effect mix channel.

**Return value**

If successful, the MIDI channel's stream handle is returned, else 0 is returned.

Use [BASS\\_ErrorGetCode](#) to get the error code.

**Error codes**

BASS\_ERROR\_HANDLE     *handle* is not valid.

BASS\_ERROR\_ILLPARAM   *chan* is not valid.

## Remarks

By default, MIDI channels do not have streams assigned to them; a MIDI channel only gets a stream when this function is called, which it then keeps until the MIDI stream is freed. MIDI channel streams can also be freed before then via [BASS\\_StreamFree](#). Each MIDI channel stream increases the CPU usage slightly, even if there are no DSP/FX set on them, so for optimal performance they should not be activated unnecessarily.

A MIDI channel stream receives the channel's dry mix, and its output is then sent through the normal reverb/chorus/userfx processing, but drum key reverb/chorus/userfx settings get ignored as it is impossible to apply different reverb/chorus/userfx levels to individual keys in this case (they have already been mixed).

MIDI channel streams can only be used to set DSP/FX on the channels. They cannot be used with [BASS\\_ChannelGetData](#) or [BASS\\_ChannelGetLevel](#) to visualise the channels, for example, but that could be achieved with a DSP function instead. A MIDI channel stream's sample format is always floating-point, regardless of the MIDI stream's sample format.

**Platform-specific**

This function is not available on the Android armeabi architecture.

## Example

Apply some DX8 distortion to channel 1 of a MIDI stream.

```
HSTREAM chan1=BASS_MIDI_StreamGetChannel(midi, 0); // get a stream  
HFX fx=BASS_ChannelSetFX(chan1, BASS_FX_DX8_DISTORTION, 0); // set
```

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamCreateFile](#)

[BASS\\_ChannelSetDSP](#), [BASS\\_ChannelSetFX](#)

# BASS\_MIDI\_StreamGetEvent

---

Retrieves the current value of an event in a channel of a MIDI stream.

```
DWORD BASS_MIDI_StreamGetEvent(  
    HSTREAM handle,  
    DWORD chan,  
    DWORD event  
);
```

## Parameters

- handle The MIDI stream to get the event value from.
- chan The MIDI channel to get the event value from... 0 = channel 1.
- event The event value to retrieve. See [BASS MIDI StreamEvent](#) for details on the available event types and their values. With the MIDI\_EVENT\_NOTE, MIDI\_EVENT\_KEYPRES, MIDI\_EVENT\_SCALETUNING and drum key (MIDI\_EVENT\_DRUM\_CUTOFF/etc) events, the HIWORD - use MAKELONG(event,key) - can be used to specify which key/note to get the value from. Special MIDI\_EVENT\_NOTES and MIDI\_EVENT\_VOICES events are also available to check how many keys are pressed and how many voices are active, respectively.

**Return value**

If successful, the event value 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\_ILLPARAM One of the other parameters is invalid.

**Remarks**

The MIDI\_EVENT\_NOTE event value will be 1 if the specified key is pressed and 0 if not. The MIDI\_EVENT\_NOTES event can be used to check how many keys in total are pressed in the specified channel. If a key is simultaneously pressed multiple times, it will still only be counted once.

Syncs can be used to be informed of when event values change.

**See also**

[BASS\\_MIDI\\_StreamEvent](#), [BASS\\_MIDI\\_StreamGetEvents](#)

# BASS\_MIDI\_StreamGetEvents

---

Retrieves the events in a MIDI stream.

```
DWORD BASS_MIDI_StreamGetEvents(  
    HSTREAM handle,  
    int track,  
    DWORD filter,  
    BASS_MIDI_EVENT *events  
);
```

## Parameters

- handle The MIDI stream to get the events from.
- track The track to get the events from... 0 = 1st track, -1 = all tracks.
- filter The type of events to retrieve... 0 = all events. See [BASS MIDI StreamEvent](#) for a list of possible event types.
- events Pointer to an array to receive the events... NULL = get the number of events without getting the events themselves.

**Return value**

If successful, the number of events 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 does not have an event sequence.  
BASS\_ERROR\_ILLPARAM   *track* is not valid.

**Remarks**

This function should first be called with *events* = *NULL* to get the number of events, before allocating an array of the required size and retrieving the events. The events are ordered chronologically, and by track number (lowest first) if multiple events have the same position. Global events (eg. tempo) are always placed in the 1st track, even if they were originally in a different track in the MIDI file.

[BASS\\_MIDI\\_StreamGetEventsEx](#) can be used to get a portion of the events instead of all of them.

## Example

Retrieve all events in the 1st track.

```
DWORD eventc=BASS_MIDI_StreamGetEvents(handle, 0, 0, NULL); // get 1
BASS_MIDI_EVENT *events=(BASS_MIDI_EVENT*)malloc(eventc*sizeof(BASS_
BASS_MIDI_StreamGetEvents(handle, 0, 0, events); // get the events
```

Retrieve all note events in the 2nd track.

```
DWORD eventc=BASS_MIDI_StreamGetEvents(handle, 1, MIDI_EVENT_NOTE, 1
BASS_MIDI_EVENT *events=(BASS_MIDI_EVENT*)malloc(eventc*sizeof(BASS_
BASS_MIDI_StreamGetEvents(handle, 1, MIDI_EVENT_NOTE, events); // g
```

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamGetEvent](#),  
[BASS\\_MIDI\\_StreamGetEventsEx](#), [BASS\\_MIDI\\_EVENT](#) structure

# BASS\_MIDI\_StreamGetEventsEx

---

Retrieves a portion of the events in a MIDI stream.

```
DWORD BASS_MIDI_StreamGetEventsEx(  
    HSTREAM handle,  
    int track,  
    DWORD filter,  
    BASS\_MIDI\_EVENT *events,  
    DWORD start,  
    DWORD count  
);
```

## Parameters

- handle The MIDI stream to get the events from.
- track The track to get the events from... 0 = 1st track, -1 = all tracks.
- filter The type of events to retrieve... 0 = all events. See [BASS MIDI StreamEvent](#) for a list of possible event types.
- events Pointer to an array to receive the events... NULL = get the number of events available in the specified range without getting the events themselves.
- start The first event to retrieve.
- count The maximum number of events to retrieve.

**Return value**

If successful, the number of events 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 does not have an event sequence.  
BASS\_ERROR\_ILLPARAM   *track* is not valid.

**Remarks**

This function is identical to [BASS\\_MIDI\\_StreamGetEvents](#) except that it can retrieve a portion of the events instead of all of them.

## Example

Retrieve the first 10 events in the 1st track.

```
BASS_MIDI_EVENT events[10]; // event array  
DWORD eventc=BASS_MIDI_StreamGetEvents(handle, 0, 0, events, 0, 10)
```

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamGetEvent](#),  
[BASS\\_MIDI\\_StreamGetEvents](#), [BASS\\_MIDI\\_EVENT](#) structure

# BASS\_MIDI\_StreamGetFonts

---

Retrieves the soundfont configuration of a MIDI stream, or the default soundfont configuration.

```
DWORD BASS_MIDI_StreamGetFonts(  
    HSTREAM handle,  
    void *fonts,  
    DWORD count  
);
```

## Parameters

- handle** The MIDI stream to retrieve the soundfont configuration of... 0 = get default soundfont configuration.
- fonts** An array of [BASS\\_MIDI\\_FONT](#) or [BASS\\_MIDI\\_FONTEX](#) to retrieve the soundfont configuration.
- count** The maximum number of elements to retrieve in the *fonts* array. The `BASS_MIDI_FONT_EX` flag may also be used to specify that *fonts* is an array of [BASS\\_MIDI\\_FONTEX](#) rather than [BASS\\_MIDI\\_FONT](#). This and *fonts* can be 0, to get the number of elements in the soundfont configuration.

**Return value**

If successful, the number of soundfonts in the configuration (which can be higher than *count*) is returned, else -1 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

**Error codes**

BASS\_ERROR\_HANDLE *handle* is not valid.

**Remarks**

When a soundfont matching the MIDI file is loaded, it will be the first element in the returned configuration.

### **Platform-specific**

Depending on the programming language used, the `BASS_MIDI_FONT_EX` flag may be automatically applied when the [BASS\\_MIDI\\_FONTEX](#) structure is used, through overloading in the BASSMIDI header. That is true for C++ and Delphi.

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamSetFonts](#),  
[BASS\\_MIDI\\_FONT](#) structure, [BASS\\_MIDI\\_FONTEX](#) structure

# BASS\_MIDI\_StreamGetMark

---

Retrieves a marker from a MIDI file stream.

```
BOOL BASS_MIDI_StreamGetMark(  
    HSTREAM handle,  
    DWORD type,  
    DWORD index,  
    BASS_MIDI_MARK *mark  
);
```

## Parameters

handle	The MIDI stream to retrieve the marker from.
type	The type of marker to retrieve, one of the following.
BASS_MIDI_MARK_COPY	Copyright notice events (MIDI meta event 2).
BASS_MIDI_MARK_CUE	Cue events (MIDI meta event 7).
BASS_MIDI_MARK_INST	Instrument name events (MIDI meta event 4).
BASS_MIDI_MARK_KEYSIG	Key signature events (MIDI meta event 89). The marker text is in the form of "a b", where <i>a</i> is the number of sharps (if positive) or flats (if negative), and <i>b</i> signifies major (if 0) or minor (if 1).
BASS_MIDI_MARK_LYRIC	Lyric events (MIDI meta event 5).
BASS_MIDI_MARK_MARKER	Marker events (MIDI meta event 6).
BASS_MIDI_MARK_TEXT	Text events (MIDI meta event 1).
BASS_MIDI_MARK_TIMESIG	Time signature events (MIDI meta event 88). The marker text is in the form of "a/b c d", where <i>a</i> is the numerator, <i>b</i> is the denominator, <i>c</i> is the metronome pulse, and <i>d</i> is the number of 32nd notes per MIDI quarter-note.
BASS_MIDI_MARK_TRACK	Track name events (MIDI meta event 3).
BASS_MIDI_MARK_TRACKSTART	Start of a track in a standard MIDI format 2 file. The marker text is the track

BASS\_MIDI\_MARK\_TICK                      number (0 = the first).  
Flag: Get the marker's  
position in ticks rather than  
bytes.

index    The marker to retrieve... 0 = the first.  
mark    Pointer to a structure to receive the marker details.

**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\_ILLTYPE    *type* in not valid.  
BASS\_ERROR\_ILLPARAM   *index* in not valid.

## **Remarks**

The markers are ordered chronologically, and by track number (lowest first) if multiple markers have the same position.

Syncs can be used to be informed of when markers are encountered during playback.

If a lyric marker text begins with a / (slash) character, that means a new line should be started. If the text begins with a \ (backslash) character, the display should be cleared. Lyrics can sometimes be found in BASS\_MIDI\_MARK\_TEXT instead of BASS\_MIDI\_MARK\_LYRIC markers.

## Example

List a MIDI stream's markers.

```
BASS_MIDI_MARK mark;  
int a;  
for (a=0; BASS_MIDI_StreamGetMark(handle, BASS_MIDI_MARK_MARKER, a,  
    printf("marker @ %d = %s\n", mark.pos, mark.text); // display m
```

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamGetMarks](#),  
[BASS\\_MIDI\\_MARK](#) structure

# BASS\_MIDI\_StreamGetMarks

---

Retrieves the markers in a MIDI file stream.

```
DWORD BASS_MIDI_StreamGetMarks(  
    HSTREAM handle,  
    int track,  
    DWORD type,  
    BASS_MIDI_MARK *marks  
);
```

## Parameters

handle	The MIDI stream to retrieve the markers from.
track	The track to get the markers from... 0 = 1st track, -1 = all tracks.
type	The type of marker to retrieve, one of the following.
BASS_MIDI_MARK_COPY	Copyright notice events (MIDI meta event 2).
BASS_MIDI_MARK_CUE	Cue events (MIDI meta event 7).
BASS_MIDI_MARK_INST	Instrument name events (MIDI meta event 4).
BASS_MIDI_MARK_KEYSIG	Key signature events (MIDI meta event 89). The marker text is in the form of "a b", where <i>a</i> is the number of sharps (if positive) or flats (if negative), and <i>b</i> signifies major (if 0) or minor (if 1).
BASS_MIDI_MARK_LYRIC	Lyric events (MIDI meta event 5).
BASS_MIDI_MARK_MARKER	Marker events (MIDI meta event 6).
BASS_MIDI_MARK_TEXT	Text events (MIDI meta event 1).
BASS_MIDI_MARK_TIMESIG	Time signature events (MIDI meta event 88). The marker text is in the form of "a/b c d", where <i>a</i> is the numerator, <i>b</i> is the denominator, <i>c</i> is the metronome pulse, and <i>d</i> is the number of 32nd notes per MIDI quarter-note.
BASS_MIDI_MARK_TRACK	Track name events (MIDI meta event 3).
BASS_MIDI_MARK_TICK	Flag: Get the marker's position in ticks rather than bytes.
marks	Pointer to an array to receive the marker details... NULL = get the number of markers without getting the markers themselves.

**Return value**

If successful, the number of markers 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\_ILLTYPPE    *type* is not valid.  
BASS\_ERROR\_ILLPARAM    *track* is not valid.

## Remarks

This function should first be called with *marks* = *NULL* to get the number of markers, before allocating an array of the required size and retrieving the markers. The markers are ordered chronologically, and by track number (lowest first) if multiple markers have the same position.

[Syncs](#) can be used to be informed of when markers are encountered during playback.

If a lyric marker text begins with a / (slash) character, that means a new line should be started. If the text begins with a \ (backslash) character, the display should be cleared. Lyrics can sometimes be found in BASS\_MIDI\_MARK\_TEXT instead of BASS\_MIDI\_MARK\_LYRIC markers.

## Example

Retrieve the markers from all tracks in a MIDI stream.

```
DWORD markc=BASS_MIDI_StreamGetMarks(handle, -1, BASS_MIDI_MARK_MARKER);
BASS_MIDI_MARK *marks=(BASS_MIDI_MARK*)malloc(markc*sizeof(BASS_MIDI_MARK));
BASS_MIDI_StreamGetMarks(handle, -1, BASS_MIDI_MARK_MARKER, marks);
```

**See also**

[BASS\\_MIDI\\_StreamCreateFile](#), [BASS\\_MIDI\\_StreamGetMark](#),  
[BASS\\_MIDI\\_MARK](#) structure

# BASS\_MIDI\_StreamGetPreset

---

Retrieves the preset currently in use on a channel of a MIDI stream.

```
BOOL BASS_MIDI_StreamGetPreset(  
    HSTREAM handle,  
    DWORD chan,  
    BASS\_MIDI\_FONT *font  
);
```

## Parameters

- handle The MIDI stream to retrieve the soundfont configuration of... 0 = get default soundfont configuration.
- chan The MIDI channel... 0 = channel 1.
- fonts Pointer to a structure to receive the preset 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	<i>handle</i> is not valid.
BASS_ERROR_ILLPARAM	<i>chan</i> is not valid.
BASS_ERROR_NOTAVAIL	No preset is currently in use on the specified MIDI channel.

**Remarks**

This function tells what preset from what soundfont is currently being used on a particular MIDI channel. That information can be used to get the preset's name from [BASS\\_MIDI\\_FontGetPreset](#).

No preset information will be available for a MIDI channel until a note is played in that channel. The present and bank numbers will not necessarily match the channel's current MIDI\_EVENT\_PROGRAM and MIDI\_EVENT\_BANK event values, but rather what the MIDI stream's soundfont configuration maps those event values to.

## Example

List the presets in use on all 16 MIDI channels.

```
BASS_MIDI_FONT font;
int ch;
for (ch=0; ch<16; ch++) {
    if (BASS_MIDI_StreamGetPreset(handle, ch, &font;)) // get prese
        printf("%d: %s\n", ch+1, BASS_MIDI_FontGetPreset(font.font,
}
}
```

**See also**

[BASS\\_MIDI\\_FontGetPreset](#), [BASS\\_MIDI\\_FONT structure](#)

# BASS\_MIDI\_StreamLoadSamples

---

Preloads the samples required by a MIDI file stream.

```
BOOL BASS_MIDI_StreamLoadSamples(  
    HSTREAM handle  
);
```

**Parameters**

handle The MIDI 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	<i>handle</i> is not valid.
BASS_ERROR_NOTAVAIL	The stream is for real-time events only, so it is not possible to know what presets are going to be used. Use <a href="#">BASS_MIDI_FontLoad</a> instead.

## Remarks

Samples are normally loaded as they are needed while rendering a MIDI stream, which can result in CPU spikes, particularly with packed soundfonts. That generally will not cause any problems, but when smooth/constant performance is critical this function can be used to preload the samples before rendering, so avoiding the need to load them while rendering.

The samples loaded by this function are subject to automatic compacting via the [BASS\\_CONFIG\\_MIDI\\_COMPACT](#) option, so it is probably wise to disable that option when using this function, to avoid any chance of the loaded samples subsequently being automatically unloaded.

This function is not affected by any filtering that may have been enabled via [BASS\\_MIDI\\_StreamSetFilter](#); the samples loaded will be those needed by the original event sequence.

This function should not be used while the MIDI stream is being rendered, as it could delay the rendering.

**See also**

[BASS MIDI FontLoad](#), [BASS MIDI StreamCreateFile](#),  
[BASS MIDI StreamSetFont](#), [BASS CONFIG MIDI COMPACT](#)

# BASS\_MIDI\_StreamSetFilter

---

Sets an event filtering function on a MIDI stream.

```
BOOL BASS_MIDI_StreamSetFilter(  
    HSTREAM handle,  
    BOOL seeking,  
    MIDIFILTERPROC *proc,  
    void *user  
);
```

## **Parameters**

- handle The MIDI stream handle.
- seeking Also filter events when seeking?
- proc The callback function... NULL = no filtering.
- user User instance data to pass to the callback function.

**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\_NOTAVAIL   The stream does not have an event sequence.

**Remarks**

This function allows a MIDI stream to have its events modified during playback via a callback function. The callback function will be called before an event is processed, and it can choose to keep the event as is, or it can modify or drop the event. The filtering can also be applied to events while seeking, so that playback begins in a filtered state after seeking.

Filtering only applies to a MIDI stream's defined event sequence, not any events that are applied via [BASS\\_MIDI\\_StreamEvent](#) or [BASS\\_MIDI\\_StreamEvents](#).

**See also**

[MIDIFILTERPROC callback](#)

# BASS\_MIDI\_StreamSetFont

---

Applies a soundfont configuration to a MIDI stream, or sets the default soundfont configuration.

```
BOOL BASS_MIDI_StreamSetFont(  
    HSTREAM handle,  
    void *fonts,  
    DWORD count  
);
```

## Parameters

- handle** The MIDI stream to apply the soundfonts to... 0 = set default soundfont configuration.
- fonts** An array of [BASS\\_MIDI\\_FONT](#) or [BASS\\_MIDI\\_FONTEX](#) containing the soundfonts to apply.
- count** The number of elements in the *fonts* array. The `BASS_MIDI_FONT_EX` flag may also be used to specify that *fonts* is an array of [BASS\\_MIDI\\_FONTEX](#) rather than [BASS\\_MIDI\\_FONT](#).

**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\_ILLPARAM Something in the *fonts* array is invalid, check the soundfont handles.

**Remarks**

Multiple soundfonts can be stacked, each providing different presets, for example. When a preset is present in multiple soundfonts, the earlier soundfont in the array has priority. When a soundfont matching the MIDI file is loaded, that remains loaded when calling this function, and has priority over all other soundfonts. When a preset is not available on a non-0 bank in any soundfont, BASSMIDI will try to fall back to bank 0; first the LSB and then the MSB if still unsuccessful.

Changing the default configuration only affects subsequently created MIDI streams. Existing streams that are using the previous default configuration will continue to use that previous configuration.

### **Platform-specific**

Depending on the programming language used, the `BASS_MIDI_FONT_EX` flag may be automatically applied when the [BASS\\_MIDI\\_FONTEX](#) structure is used, through overloading in the BASSMIDI header. That is true for C++ and Delphi.

## Example

Set a MIDI stream to use one soundfont for program 10 on bank 0, and all available presets from another soundfont.

```
BASS_MIDI_FONT fonts[2];
fonts[0].font=font1;
fonts[0].preset=10; // preset 10
fonts[0].bank=0; // bank 0
fonts[1].font=font2;
fonts[1].preset=-1; // all presets
fonts[1].bank=0; // default banks
BASS_MIDI_StreamSetFont(handle, fonts, 2); // apply it to the stream
```

Set a MIDI stream to use preset 20 from one soundfont for program 10 on bank 0, and all available presets from another soundfont.

```
BASS_MIDI_FONT_EX fonts[2];
fonts[0].font=font1;
fonts[0].spreset=20; // soundfont preset 20
fonts[0].sbank=0; // soundfont bank 0
fonts[0].dpreset=10; // destination preset 10
fonts[0].dbank=0; // destination bank 0
fonts[0].dbanklsb=0; // destination bank LSB 0
fonts[1].font=font2;
fonts[1].spreset=-1; // all presets
fonts[1].sbank=-1; // all banks
fonts[1].dpreset=-1; // all presets
fonts[1].dbank=0; // default banks
fonts[1].dbanklsb=0; // destination bank LSB 0
BASS_MIDI_StreamSetFont(handle, fonts, 2|BASS_MIDI_FONT_EX); // apply it to the stream
```

**See also**

[BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_StreamCreateFile](#),  
[BASS\\_MIDI\\_StreamGetFonts](#), [BASS\\_MIDI\\_StreamLoadSamples](#),  
[BASS\\_MIDI\\_FONT](#) structure, [BASS\\_MIDI\\_FONTEX](#) structure,  
[BASS\\_CONFIG\\_MIDI\\_AUTOFONT](#), [BASS\\_CONFIG\\_MIDI\\_DEFFONT](#)

# MIDI syncs - BASS\_ChannelSetSync

---

Syncs are set on MIDI streams in exactly the same way as on any other stream, using [BASS\\_ChannelSetSync](#). The following is a list of the types of sync supported on MIDI streams.

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

- BASS\_SYNC\_MIDI\_EVENT** Sync when a type of event is processed.  
*param* : event type (0 = all types). *data* :  
LOWORD = event parameter, HIWORD =  
channel (high 8 bits contain the event type  
when syncing on all types). If the event type is  
MIDI\_EVENT\_TEMPO, then the event  
parameter will use 24 bits (*channel* is  
replaced). See [BASS\\_MIDI\\_StreamEvent](#) for a  
list of event types and their parameters.
- BASS\_SYNC\_MIDI\_MARK** Sync when a marker is encountered.  
*param* : marker type. *data* : the marker index,  
which can be used in a  
[BASS\\_MIDI\\_StreamGetMark](#) call.
- BASS\_SYNC\_MIDI\_TICK** Sync when reaching a tick position.  
*param* : tick position. *data* : not used.

The BASS\_SYNC\_POS, BASS\_SYNC\_END, BASS\_SYNC\_SLIDE,  
BASS\_SYNC\_STALL and BASS\_SYNC\_FREE sync types are also supported  
on MIDI streams, as described in the [BASS\\_ChannelSetSync](#) documentation.

**See also**

[BASS\\_MIDI\\_StreamSetFilter](#)

# Plugin system

---

As well as providing dedicated stream creation functions, BASSMIDI supports the BASS plugin system, adding MIDI file support to the standard BASS stream creation functions: [BASS\\_StreamCreateFile](#), [BASS\\_StreamCreateURL](#), and [BASS\\_StreamCreateFileUser](#). This is enabled using the [BASS\\_PluginLoad](#) function.

MIDI streams created via the plugin system use the device's sample rate, equivalent to using *freq=1* in a [BASS\\_MIDI\\_StreamCreateFile](#) call.

# MIDIFILTERPROC callback

---

User defined callback function to filter events.

```
BOOL CALLBACK MidiFilterProc(  
    HSTREAM handle,  
    DWORD track,  
    BASS_MIDI_EVENT *event,  
    BOOL seeking,  
    void *user  
);
```

## Parameters

- handle The MIDI stream handle.
- track The track that the event is from... 0 = 1st track.
- event Pointer to the event structure.
- seeking TRUE = the event is being processed while seeking, FALSE = the event is being played.
- user The user instance data given when [BASS\\_MIDI\\_StreamSetFilter](#) was called.

**Return value**

Return TRUE to process the event, and FALSE to drop the event.

## Remarks

The event's type (*event*), parameter (*param*), and channel (*chan*) can be modified, but not its position (*tick* or *pos*). It is also possible to apply additional events at the same time via [BASS\\_MIDI\\_StreamEvent](#), but not [BASS\\_MIDI\\_StreamEvents](#).

MIDI\_EVENT\_NOTE, MIDI\_EVENT\_NOTESOFF, and MIDI\_EVENT\_SOUNDOFF events are ignored while seeking so they will not be received by a filtering function then. MIDI\_EVENT\_TEMPO events can be changed while seeking but doing so when seeking in bytes (BASS\_POS\_BYTE) will result in reaching a different position. Seeking in ticks (BASS\_POS\_MIDI\_TICK) is unaffected by tempo changes. The MIDI\_EVENT\_SPEED event can be used to modify the tempo without affecting seeking.

## Example

A filtering function that drops all notes with a velocity lower than 10.

```
BOOL CALLBACK MidiFilterProc(HSTREAM handle, DWORD track, BASS_MIDI_
{
    if (event->event==MIDI_EVENT_NOTE) { // got a note
        int vel=HIBYTE(event->param); // extract the velocity
        if (vel<10 && vel>0) return FALSE; // drop the note if velo
    }
    return TRUE; // process the event
}
```

A filtering function that changes bank 2 to bank 1.

```
BOOL CALLBACK MidiFilterProc(HSTREAM handle, DWORD track, BASS_MIDI_
{
    if (event->event==MIDI_EVENT_BANK && event->param==2) // got a l
        event->param==1; // change it to bank 1
    return TRUE; // process the event
}
```

**See also**

[BASS\\_MIDI\\_StreamSetFilter](#), [BASS\\_MIDI\\_EVENT](#) structure

# BASS\_MIDI\_EVENT structure

---

Used with [BASS\\_MIDI\\_StreamEvents](#) to apply events and [BASS\\_MIDI\\_StreamGetEvents](#) to retrieve events, and [BASS\\_MIDI\\_StreamCreateEvents](#) to play event sequences.

```
typedef struct {  
    DWORD event;  
    DWORD param;  
    DWORD chan;  
    DWORD tick;  
    DWORD pos;  
} BASS_MIDI_EVENT;
```

## **Members**

- event The event type.
- param The event parameter.
- chan The MIDI channel of the event... 0 = channel 1.
- tick The position of the event, in ticks.
- pos The position of the event, in bytes.

**Remarks**

When used with [BASS\\_MIDI\\_StreamEvents](#), the *tick* and *pos* member values are relative to the current decoding position, and only one of them should be used at a time (*pos* has precedence if both are). The *pos* member is ignored by [BASS\\_MIDI\\_StreamCreateEvents](#).

**See also**

[BASS\\_MIDI\\_StreamEvents](#), [BASS\\_MIDI\\_StreamGetEvents](#)

# BASS\_MIDI\_FONT structure

---

Used with [BASS\\_MIDI\\_StreamSetFonts](#) and [BASS\\_MIDI\\_StreamGetFonts](#) to set and retrieve soundfont configurations.

```
typedef struct {  
    HSOUNDFONT font;  
    int preset;  
    int bank;  
} BASS_MIDI_FONT;
```

## Members

- font     Soundfont handle, previously initialized with [BASS\\_MIDI\\_FontInit](#).
- preset   Preset number... 0-65535, -1 = use all presets in the soundfont. This determines what MIDI\_EVENT\_PROGRAM event value(s) the soundfont is used for.
- bank     Base bank number, or the bank number of the individual preset. This determines what MIDI\_EVENT\_BANK event value(s) the soundfont is used for.

## Remarks

When using an individual preset from a soundfont, BASSMIDI will first look for the exact *preset* and *bank* match, but if that is not present, the first preset from the soundfont will be used. This is useful for single preset soundfonts. Individual presets can be assigned to program numbers beyond the standard 127 limit, up to 65535, which can then be accessed via [BASS\\_MIDI\\_StreamEvent](#).

When using all presets in a soundfont, the *bank* member is a base number that is added to the soundfont's banks. For example, if *bank*=1 then the soundfont's bank 0 becomes bank 1, etc. Negative base numbers are allowed.

For more flexible mapping of soundfont presets to MIDI programs, see the [BASS\\_MIDI\\_FONTEX](#) structure.

**See also**

[BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_StreamGetFonts](#),  
[BASS\\_MIDI\\_StreamSetFonts](#)

# BASS\_MIDI\_FONTEX structure

---

Used with [BASS\\_MIDI\\_StreamSetFonts](#) and [BASS\\_MIDI\\_StreamGetFonts](#) to set and retrieve soundfont configurations.

```
typedef struct {
    HSOUNDFONT font;
    int spreset;
    int sbank;
    int dpreset;
    int dbank;
    int dbanklsb;
} BASS_MIDI_FONTEX;
```

## Members

font	Soundfont handle, previously initialized with <a href="#">BASS_MIDI_FontInit</a> .
spreset	Soundfont preset number... 0-127, -1 = use all presets.
sbank	Soundfont bank number... 0-128, -1 = use all banks.
dpsreset	Destination preset/program number... 0-65535, -1 = all presets. This determines what MIDI_EVENT_PROGRAM event value(s) the soundfont is used for.
dbank	Destination bank number, or a base bank number when using all presets from all banks. This determines what MIDI_EVENT_BANK event value(s) the soundfont is used for.
dbanklsb	Destination bank number LSB. This is the MIDI_EVENT_BANK_LSB event value that the soundfont is used for.

## Remarks

This is an extended version of the [BASS MIDI FONT](#) structure that allows more flexible mapping of soundfont presets to MIDI programs, including access to the bank LSB (eg. MIDI controller 32).

When using an individual preset from a soundfont, BASSMIDI will first look for the exact *spreset* and *sbank* match, but if that is not present, the first preset from the soundfont will be used. This is useful for single preset soundfonts. Individual presets can be assigned to program numbers beyond the standard 127 limit, up to 65535, which can then be accessed via [BASS MIDI StreamEvent](#).

When using all presets from all banks in a soundfont, the *dbank* member is a base number that is added to the soundfont's banks. For example, if *dbank* = 1 then the soundfont's bank 0 becomes bank 1, etc. Negative base numbers are allowed, to lower a soundfont's bank numbers.

The bank LSB raises the maximum number of melodic banks from 128 to 16384 (128 x 128), but the SF2 soundfont format only supports 128 banks, so a soundfont that is set to be used on all banks (*dpreset* and *dbank* are -1) will still only apply to the single bank LSB specified by *dbanklsb*.

**See also**

[BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_StreamGetFonts](#),  
[BASS\\_MIDI\\_StreamSetFonts](#)

# BASS\_MIDI\_MARK structure

---

Used with [BASS\\_MIDI\\_StreamGetMark](#) and [BASS\\_MIDI\\_StreamGetMarks](#) to retrieve markers.

```
typedef struct {  
    DWORD track;  
    DWORD pos;  
    char *text;  
} BASS_MIDI_MARK;
```

## Members

- track The MIDI track containing the marker... 0 = first.
- pos The position of the marker. This may be in bytes or ticks, depending on whether the BASS\_MIDI\_MARK\_TICK flag was used in the [BASS\\_MIDI\\_StreamGetMark](#) or [BASS\\_MIDI\\_StreamGetMarks](#) call.
- text The marker text.

**Remarks**

If a lyric marker's text begins with a / (slash) character, a new line should be started. If it begins with a \ (backslash) character, the display should be cleared.

**See also**

[BASS\\_MIDI\\_StreamGetMark](#)

# BASS\_ATTRIB\_MIDI\_CHANS attribute

---

The number of MIDI channels in a MIDI stream.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_CHANS,  
    float channels  
);
```

## **Parameters**

handle The MIDI stream handle.

channels The number of MIDI channels... 1 (min) - 128 (max). For a MIDI file stream, the minimum is 16.

**Remarks**

New channels are melodic by default. Any notes playing on a removed channel are immediately stopped.

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamCreateFile](#)

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#)

# BASS\_ATTRIB\_MIDI\_CPU attribute

---

The maximum percentage of CPU time that a MIDI stream can use.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_CPU,  
    float limit  
);
```

## **Parameters**

handle The MIDI stream handle.

limit The CPU usage limit... 0 to 100, 0 = no limit.

## Remarks

It is not strictly the CPU usage that is measured, but rather how timely the stream is able to render data. For example, a limit of 50% would mean that the rendering would need to be at least 2x real-time speed. When the limit is exceeded, BASSMIDI will begin killing voices, starting with the most quiet.

When the CPU usage is limited, the stream's samples are loaded asynchronously so that any loading delays (eg. due to slow disk) do not hold up the stream for too long. If a sample cannot be loaded in time, then it will be silenced until it is available and the stream will continue playing other samples as normal in the meantime. This does not affect sample loading via [BASS\\_MIDI\\_StreamLoadSamples](#), which always operates synchronously.

By default, a MIDI stream will have no CPU limit.

**See also**

[BASS\\_ATTRIB\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#)

# BASS\_ATTRIB\_MIDI\_PPQN attribute

---

The Pulses Per Quarter Note (or ticks per beat) value of a MIDI stream.

```
BASS_ChannelGetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_PPQN,  
    float *ppqn  
);
```

**Parameters**

handle The MIDI stream handle.

ppqn The PPQN value.

**Remarks**

This attribute is the number of ticks per beat as defined by the MIDI file; it will be 0 for MIDI streams created via [BASS\\_MIDI\\_StreamCreate](#). It is also read-only, so cannot be modified via [BASS\\_ChannelSetAttribute](#).

## Example

Get the current position of a MIDI stream in beats.

```
float ppqn;  
BASS_ChannelGetAttribute(handle, BASS_ATTRIB_MIDI_PPQN, &ppqn); //  
QWORD tick=BASS_ChannelGetPosition(handle, BASS_POS_MIDI_TICK); //  
DWORD beat=tick/ppqn; // translate it to beats
```

**See also**

[BASS\\_ChannelGetAttribute](#)

# BASS\_ATTRIB\_MIDI\_STATE attribute

---

The current state of a MIDI stream.

```
BASS_ChannelSetAttributeEx(  
    DWORD handle,  
    BASS_ATTRIB_MIDI_STATE,  
    void *state,  
    DWORD size  
);
```

**Parameters**

handle The MIDI stream handle.  
state The state data.  
size The size of the state data.

## Remarks

This attribute includes the state of all events in all MIDI channels, except for MIDI\_EVENT\_NOTE (playing notes are not preserved) and MIDI\_EVENT\_TEMPO. [BASS\\_MIDI\\_StreamGetEvent](#) can be used to get the MIDI\_EVENT\_TEMPO event state.

The structure of the MIDI state data may change in future versions, so if the data is stored, be prepared for [BASS\\_ChannelSetAttributeEx](#) to fail when trying to apply it.

## Example

Transfer the state from one MIDI stream to another.

```
DWORD size=BASS_ChannelGetAttributeEx(stream1, BASS_ATTRIB_MIDI_STATE, &size);
void *state=malloc(size); // allocate a buffer for the data
BASS_ChannelGetAttributeEx(stream1, BASS_ATTRIB_MIDI_STATE, state, &size);
BASS_ChannelSetAttributeEx(stream2, BASS_ATTRIB_MIDI_STATE, state, &size);
free(state);
```

**See also**

[BASS\\_ChannelGetAttributeEx](#), [BASS\\_ChannelSetAttributeEx](#)

# BASS\_ATTRIB\_MIDI\_SRC attribute

---

The sample rate conversion quality of a MIDI stream's samples.

```
BASS_ChannelSetAttribute(  
    DWORD handle,  
    BASS_ATTRIB_MIDI_SRC,  
    float quality  
);
```

## **Parameters**

handle The MIDI stream handle.

quality The sample rate conversion quality... 0 = linear interpolation, 1 = 8 point sinc interpolation, 2 = 16 point sinc interpolation.

**Remarks**

The samples in a soundfont will usually need to be played at rates that are different to their original rates. 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.

When this attribute setting is changed, the `BASS_MIDI_SINCINTER` flag is automatically set or unset on the MIDI stream accordingly, and vice versa. Changes can be made at any time, but the effect of changes during playback will not be heard instantaneously due to buffering.

**Platform-specific**

On Android and iOS, sinc interpolation requires a NEON supporting CPU. Sinc interpolation is not available on Windows CE. 16 point sinc interpolation is only available on Windows/OSX/Linux and requires an SSE2 supporting CPU.

**See also**

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#), [BASS\\_ATTRIB\\_SRC](#)

# BASS\_ATTRIB\_MIDI\_TRACK\_VOL attribute

---

The volume level of a track in a MIDI stream.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_TRACK_VOL + track,  
    float volume  
);
```

## **Parameters**

handle The MIDI stream handle.

track The track to set the volume of... 0 = first track.

volume The volume level... 0 = silent, 1.0 = normal/default.

## 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.

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 tracks in a MIDI file stream. MIDI streams created via [BASS\\_MIDI\\_StreamCreate](#) do not have any tracks.

## Example

Count the number of tracks in a MIDI stream.

```
int tracks=0;
float dummy;
while (BASS_ChannelGetAttribute(handle, BASS_ATTRIB_MIDI_TRACK_VOL+
    tracks++;
```

**See also**

[BASS\\_MIDI\\_StreamEvent](#)

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#),  
[BASS\\_ChannelSlideAttribute](#)

# BASS\_ATTRIB\_MIDI\_VOICES attribute

---

The maximum number of samples to play at a time in a MIDI stream.

```
BASS_ChannelSetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_VOICES,  
    float voices  
);
```

## **Parameters**

handle The MIDI stream handle.

voices The number of voices... 1 (min) - 100000 (max).

**Remarks**

If there are currently more voices active than the new limit, then some voices will be killed to meet the limit. The number of voices currently active is available via the [BASS\\_ATTRIB\\_MIDI\\_VOICES\\_ACTIVE](#) attribute.

A MIDI stream will initially have a default number of voices as determined by the [BASS\\_CONFIG\\_MIDI\\_VOICES](#) config option.

**Platform-specific**

The maximum setting is 1000 on Android, iOS, and Windows CE.

**See also**

[BASS\\_ATTRIB\\_MIDI\\_CHANS](#), [BASS\\_ATTRIB\\_MIDI\\_CPU](#),  
[BASS\\_ATTRIB\\_MIDI\\_VOICES\\_ACTIVE](#), [BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#)

# BASS\_ATTRIB\_MIDI\_VOICES\_ACTIVE attribute

---

The number of samples currently playing in a MIDI stream.

```
BASS_ChannelGetAttribute(  
    HSTREAM handle,  
    BASS_ATTRIB_MIDI_VOICES_ACTIVE,  
    float *voices  
);
```

**Parameters**

handle The MIDI stream handle.

voices The number of voices.

**Remarks**

This attribute is read-only, so cannot be modified via [BASS\\_ChannelSetAttribute](#).

[BASS\\_MIDI\\_StreamGetEvent](#) can be used with the MIDI\_EVENT\_VOICES event to check how many voices are active in individual MIDI channels.

**See also**

[BASS\\_ATTRIB\\_MIDI\\_CPU](#), [BASS\\_ATTRIB\\_MIDI\\_VOICES](#),  
[BASS\\_CONFIG\\_MIDI\\_VOICES](#)

[BASS\\_ChannelGetAttribute](#), [BASS\\_ChannelSetAttribute](#)

# BASS\_MIDI\_FontCompact

---

Compacts a soundfont's memory usage.

```
BOOL BASS_MIDI_FontCompact(  
    HSOUNDFONT handle  
);
```

**Parameters**

handle The soundfont handle... 0 = all soundfonts.

**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**

Compacting involves freeing any samples that are currently loaded but unused by any MIDI streams. The amount of sample data currently loaded can be retrieved using [BASS\\_MIDI\\_FontGetInfo](#).

**See also**

[BASS\\_MIDI\\_FontFree](#), [BASS\\_MIDI\\_FontGetInfo](#), [BASS\\_MIDI\\_FontUnload](#),  
[BASS\\_CONFIG\\_MIDI\\_COMPACT](#)

# BASS\_MIDI\_FontFree

---

Frees a soundfont.

```
BOOL BASS_MIDI_FontFree(  
    HSOUNDFONT handle  
);
```

**Parameters**

handle The soundfont 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.

**Remarks**

When a soundfont is freed, it is automatically removed from any MIDI streams that are using it.

**See also**

[BASS\\_MIDI\\_FontCompact](#), [BASS\\_MIDI\\_FontInit](#)

# BASS\_MIDI\_FontGetInfo

---

Retrieves information on a soundfont.

```
BOOL BASS_MIDI_FontGetInfo(  
    HFONT handle,  
    BASS_MIDI_FONTINFO *info  
);
```

**Parameters**

handle The soundfont to get info on.

info Pointer to a structure to receive the info.

**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\\_MIDI\\_FontGetPreset](#), [BASS\\_MIDI\\_FontGetPresets](#),  
[BASS\\_MIDI\\_FONTINFO](#) structure

# BASS\_MIDI\_FontGetPreset

---

Retrieves the name of a preset in a soundfont.

```
char *BASS_MIDI_FontGetPreset(  
    HOUNDFONT handle,  
    int preset,  
    int bank  
);
```

## **Parameters**

handle The soundfont to get the preset name from.

preset Preset number... -1 = any preset (the first encountered).

bank Bank number... -1 = any bank (the first encountered).

**Return value**

If successful, the requested preset name is 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 soundfont does not contain the requested  
                          preset.

**Remarks**

SFZ files do not contain preset names, so their filenames (minus the ".sfz" extension) are used instead when available.

A list of all presets in a soundfont is available from [BASS\\_MIDI\\_FontGetPresets](#).

Drum kits are located in bank 128, and possibly bank 127 in the case of XG drum kits.

**See also**

[BASS\\_MIDI\\_FontGetInfo](#), [BASS\\_MIDI\\_FontGetPresets](#),  
[BASS\\_MIDI\\_FontInit](#)

# BASS\_MIDI\_FontGetPresets

---

Retrieves the presets in a soundfont.

```
BOOL BASS_MIDI_FontGetPresets(  
    HSOUNDFONT handle,  
    DWORD *presets  
);
```

**Parameters**

handle The soundfont to get the presets from.

presets A pointer to an array to receive the presets.

**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**

Before using this function to retrieve the presets, [BASS\\_MIDI\\_FontGetInfo](#) should be used to get the number of presets in the soundfont. The presets are delivered with the preset number in the LOWORD and the bank number in the HIWORD, and in numerically ascending order.

## Example

List all presets in a soundfont, with their names.

```
int a;
BASS_MIDI_FONTINFO info;
BASS_MIDI_FontGetInfo(handle, &info); // get soundfont info for pr
DWORD *presets=(DWORD*)malloc(info.presets*sizeof(DWORD)); // alloc
BASS_MIDI_FontGetPresets(handle, presets); // get the presets
for (a=0; a<info.presets; a++) {
    DWORD preset=LOWORD(presets[a]), bank=HIWORD(presets[a]); // ex
    const char *name=BASS_MIDI_FontGetPreset(handle, preset, bank);
    printf("%d.%d: %s\n", bank, preset, name);
}
free(presets);
```

**See also**

[BASS\\_MIDI\\_FontGetInfo](#), [BASS\\_MIDI\\_FontGetPreset](#)

# BASS\_MIDI\_FontGetVolume

---

Retrieves a soundfont's volume level.

```
float BASS_MIDI_FontGetVolume(  
    HSONDFONT handle  
);
```

**Parameters**

handle The soundfont to get the volume of.

**Return value**

If successful, the soundfont's volume level is returned, else -1 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

**Error codes**

BASS\_ERROR\_HANDLE *handle* is not valid.

**See also**

[BASS\\_MIDI\\_FontSetVolume](#)

# BASS\_MIDI\_FontInit

---

Initializes a soundfont.

```
HSOUNDFONT BASS_MIDI_FontInit(  
    void *file,  
    DWORD flags  
);
```

## Parameters

**file** The soundfont filename.

**flags** Any combination of these flags.

**BASS\_MIDI\_FONT\_MMAP**

Map the file into memory. This flag is ignored if the soundfont is packed as the sample data cannot be played directly from a mapping; it needs to be decoded. This flag is also ignored if the file is too large to be mapped into memory.

**BASS\_MIDI\_FONT\_NOFX**

Ignore the reverb/chorus levels of the presets in the soundfont (only use the levels in the MIDI events).

**BASS\_MIDI\_FONT\_XGDRUMS**

Use bank 127 in the soundfont for XG drum kits. When an XG drum kit is needed, bank 127 in soundfonts that have this flag set will be checked first, before falling back to bank 128 (the standard SF2 drum kit bank) if it is not available there.

**BASS\_UNICODE**

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

**Return value**

If successful, the soundfont'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's format is not recognised/supported.

## Remarks

BASSMIDI uses SF2 and/or SFZ soundfonts to provide the sounds to use in the rendering of MIDI files. Several soundfonts can be found on the internet, including a couple on the [BASS website](#).

A soundfont needs to be initialized before it can be used to render MIDI streams. Once initialized, a soundfont can be assigned to MIDI streams via the [BASS\\_MIDI\\_StreamSetFont](#) function. A single soundfont can be shared by multiple MIDI streams. If a soundfont is initialized multiple times, each instance will have its own handle but share the same sample data. Information on the initialized soundfont can be retrieved using [BASS\\_MIDI\\_FontGetInfo](#).

Soundfonts use PCM sample data as standard, but BASSMIDI can accept any format that is supported by BASS or its add-ons (the add-ons need to be loaded via [BASS\\_PluginLoad](#)). The [BASS\\_MIDI\\_FontPack](#) function can be used to compress the sample data in SF2 files. SFZ samples are in separate files and can be compressed using standard encoding tools.

Using soundfonts that are located somewhere other than the file system is possible via [BASS\\_MIDI\\_FontInitUser](#).

## **SF2 support**

The SF2 synthesis model is fully supported, as are all SF2 generators. Basic support for the note velocity to filter cutoff (`initialFilterFc`) and note velocity to volume envelope attack (`attackVolEnv`) modulators is also included. In each case, multiple modulators in the global/instrument/preset zones is supported but only if they are identical (eg. all linear negative mono). If different types are present on a preset, then the last one (with a non-zero amount) will be used. The SF2 spec's slightly strange default note velocity to filter cutoff modulator is not used.

## SFZ support

The following SFZ opcodes are supported: ampeg\_attack, ampeg\_decay, ampeg\_delay, ampeg\_hold, ampeg\_release, ampeg\_sustain, ampeg\_vel2attack, ampeg\_vel2decay, amplfo\_delay/fillfo\_delay/pitchlfo\_delay, amplfo\_depth, amplfo\_freq/fillfo\_freq/pitchlfo\_freq, amp\_veltrack, cutoff, default\_path, effect1, effect2, end, fileg\_attack/pitcheq\_attack, fileg\_decay/pitcheq\_decay, fileg\_delay/pitcheq\_delay, fileg\_depth, fileg\_hold/pitcheq\_hold, fileg\_release/pitcheq\_release, fileg\_sustain/pitcheq\_sustain, fileg\_vel2depth, fillfo\_depth, fil\_veltrack, group, hicc1, hicc64, hikey, hirand, hivel, key, locc1, locc64, lokey, loop\_end, loop\_mode, loop\_start, lorand, lovel, offset, off\_by, off\_mode, pan, pitcheq\_depth, pitchlfo\_depth, pitch\_keycenter, pitch\_keytrack, pitch\_veltrack, resonance, sample, seq\_length, seq\_position, transpose, tune, volume. The fil\_type opcode is also supported, but only to confirm that a low pass filter is wanted (the filter will be disabled otherwise). The combined EG and LFO entries in the opcode list reflect that there is a shared EG for pitch/filter and a shared LFO for amplitude/pitch/filter, as is the case in SF2. Information on these (and other) SFZ opcodes can be found at [www.sfzformat.com](http://www.sfzformat.com).

Samples can also be loaded from memory by setting the "sample" opcode to "mem:<address>:<length>", where *address* and *length* are both in hexadecimal. The memory should remain valid until the font is freed via [BASS\\_MIDI\\_FontFree](#).

SFZ files do not have a defined preset or bank number, so they are nominally assigned to preset 0 in bank 0 when loaded, but can be assigned to other presets/banks via [BASS\\_MIDI\\_StreamSetFonts](#).

**Platform-specific**

The `BASS_MIDI_FONT_MMAP` option is not available on big-endian systems (eg. PowerPC) as a soundfont's little-endian sample data cannot be played directly from a mapping; its byte order needs to be reversed.

## Example

Initialize a soundfont.

```
HSOUNDFONT sfont=BASS_MIDI_FontInit("afile.sf2", 0);
```

**See also**

[BASS\\_MIDI\\_FontFree](#), [BASS\\_MIDI\\_FontGetInfo](#),  
[BASS\\_MIDI\\_FontGetPresets](#), [BASS\\_MIDI\\_FontInitUser](#),  
[BASS\\_MIDI\\_FontPack](#), [BASS\\_MIDI\\_FontLoad](#), [BASS\\_MIDI\\_FontSetVolume](#),  
[BASS\\_MIDI\\_StreamSetFonts](#), [BASS\\_CONFIG\\_MIDI\\_COMPACT](#)

# BASS\_MIDI\_FontInitUser

---

Initializes a soundfont via user callback functions.

```
HSOUNDFONT BASS_MIDI_FontInitUser(  
    BASS\_FILEPROCS *procs,  
    void *user,  
    DWORD flags  
);
```

## Parameters

procs The user defined file functions.

user User instance data to pass to the callback functions.

flags Any combination of these flags.

**BASS\_MIDI\_FONT\_XGDRUMS** Use bank 127 in the soundfont for XG drum kits. When an XG drum kit is needed, bank 127 in soundfonts that have this flag set will be checked first, before falling back to bank 128 (the standard SF2 drum kit bank) if it is not available there.

**Return value**

If successful, the soundfont's handle is returned, else 0 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

**Error codes**

BASS\_ERROR\_FILEFORM The file's format is not recognised/supported.

**Remarks**

The unbuffered file system (STREAMFILE\_NOBUFFER) is always used by this function.

**See also**

[BASS\\_MIDI\\_FontFree](#), [BASS\\_MIDI\\_FontGetInfo](#),  
[BASS\\_MIDI\\_FontGetPresets](#), [BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_FontPack](#),  
[BASS\\_MIDI\\_FontLoad](#), [BASS\\_MIDI\\_FontSetVolume](#),  
[BASS\\_MIDI\\_StreamSetFonts](#), [BASS\\_CONFIG\\_MIDI\\_COMPACT](#)

# BASS\_MIDI\_FontLoad

---

Preloads presets from a soundfont.

```
BOOL BASS_MIDI_FontLoad(  
    HSOUNDFONT handle,  
    int preset,  
    int bank  
);
```

## **Parameters**

- handle The soundfont handle.
- preset Preset number to load... -1 = all presets.
- bank Bank number to load... -1 = all banks.

**Return value**

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

**Error codes**

BASS_ERROR_HANDLE	<i>handle</i> is not valid.
BASS_ERROR_CODEC	The appropriate add-on to decode the samples is not loaded.
BASS_ERROR_NOTAVAIL	The soundfont does not contain the requested preset.

## Remarks

Samples are normally loaded as they are needed while rendering a MIDI stream, which can result in CPU spikes, particularly with packed soundfonts. That generally will not cause any problems, but when smooth/constant performance is critical this function can be used to preload the samples before rendering, so avoiding the need to load them while rendering.

When preloading samples to render a particular MIDI stream, it is more efficient to use [BASS\\_MIDI\\_StreamLoadSamples](#) to preload the specific samples that the MIDI stream will use, rather than preloading the entire soundfont.

Samples that are preloaded by this function are not affected by automatic compacting via the [BASS\\_CONFIG\\_MIDI\\_COMPACT](#) option, but can be compacted/unloaded manually with [BASS\\_MIDI\\_FontCompact](#) and [BASS\\_MIDI\\_FontUnload](#).

A soundfont should not be preloaded while it is being used to render any MIDI streams, as that could delay the rendering.

**See also**

[BASS\\_MIDI\\_FontCompact](#), [BASS\\_MIDI\\_FontGetInfo](#),  
[BASS\\_MIDI\\_FontGetPreset](#), [BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_FontUnload](#),  
[BASS\\_MIDI\\_StreamLoadSamples](#)

# BASS\_MIDI\_FontPack

---

Produces a compressed version of a soundfont.

```
BOOL BASS_MIDI_FontPack(  
    HSONDFONT handle,  
    void *outfile,  
    void *encoder,  
    DWORD flags  
);
```

## Parameters

handle	The soundfont to pack.
outfile	Filename for the packed soundfont.
encoder	Encoder command-line.
flags	Any combination of these flags.
	<b>BASS_MIDI_PACK_NOHEAD</b> Don't send a WAVE header to the encoder. If this flag is used then the sample format (mono 16-bit) must be passed to the encoder some other way, eg. via the command-line.
	<b>BASS_MIDI_PACK_16BIT</b> Reduce 24-bit sample data to 16-bit before encoding.
	<b>BASS_UNICODE</b> <i>outfile</i> and <i>encoder</i> are in UTF-16 form. Otherwise they are ANSI on Windows, and UTF-8 on other platforms.

**Return value**

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

## **Error codes**

BASS_ERROR_HANDLE	<i>handle</i> is not valid.
BASS_ERROR_FILEOPEN	Could not start the encoder. Check that the executable exists.
BASS_ERROR_CREATE	Could not create the output file, <i>outfile</i> .
BASS_ERROR_UNKNOWN	Some other mystery problem!

## Remarks

Standard soundfonts use PCM samples, so they can be quite large, which can be a problem if they are to be distributed. To reduce the size, BASSMIDI can compress the samples using any command-line encoder with STDIN and STDOUT support. Packed soundfonts can be used for rendering by BASSMIDI just like normal soundfonts. They can also be unpacked using [BASS\\_MIDI\\_FontUnpack](#).

Although any command-line encoder can be used, it is best to use a lossless format like [FLAC](#) or [WavPack](#), rather than a lossy one like OGG or MP3. Using a lossless encoder, the packed soundfont will produce exactly the same results as the original soundfont, and will be identical to the original when unpacked. As a compromise between quality and size, the WavPack hybrid/lossy mode also produces good sounding results.

The encoder must be told (via the command-line) to expect input from STDIN and to send its output to STDOUT.

Before using a packed soundfont, the appropriate BASS add-on needs to be loaded via [BASS\\_PluginLoad](#). For example, if the samples are FLAC encoded, BASSFLAC would need to be loaded. During rendering, the samples are unpacked as they are needed, which could result in CPU spikes. Where smooth performance is critical, it may be wise to preload the samples using [BASS\\_MIDI\\_FontLoad](#) or [BASS\\_MIDI\\_StreamLoadSamples](#).

A soundfont should not be packed while it is being used to render any MIDI streams, as that could delay the rendering.

This function only applies to SF2 soundfonts. SFZ samples can be compressed using standard encoding tools.

**Platform-specific**

This function is not available on iOS or Android.

## Example

Create a FLAC encoded version of a soundfont.

```
HSOUNDFONT handle=BASS_MIDI_FontInit("afile.sf2", 0); // open original  
BASS_MIDI_FontPack(handle, "afile.sf2pack", "flac --best -", 0); //
```

**See also**

[BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_FontLoad](#), [BASS\\_MIDI\\_FontUnpack](#)

# BASS\_MIDI\_FontSetVolume

---

Sets a soundfont's volume level.

```
BOOL BASS_MIDI_FontSetVolume(  
    HOUNDFONT handle,  
    float volume  
);
```

**Parameters**

handle The soundfont to set the volume of.

volume The volume level... 0 = silent, 1.0 = normal/default.

**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**

By default, some soundfonts may be louder than others, which could be a problem when mixing multiple soundfonts. This function can be used to compensate for any differences, by raising the level of the quieter soundfonts or lowering the louder ones.

Changes take immediate effect in any MIDI streams that are using the soundfont.

**See also**

[BASS\\_MIDI\\_FontGetVolume](#)

# BASS\_MIDI\_FontUnload

---

Unloads presets from a soundfont.

```
BOOL BASS_MIDI_FontUnload(  
    HSOUNDFONT handle,  
    int preset,  
    int bank  
);
```

## **Parameters**

- handle The soundfont handle.
- preset Preset number to unload... -1 = all presets.
- bank Bank number to unload... -1 = all banks.

**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\_NOTAVAIL   The soundfont does not contain the specified preset, or the soundfont is memory mapped.

**Remarks**

An unloaded preset will be loaded again when needed by a MIDI stream. Any samples that are currently being used by a MIDI stream will not be unloaded.

**See also**

[BASS\\_MIDI\\_FontCompact](#), [BASS\\_MIDI\\_FontLoad](#)

# BASS\_MIDI\_FontUnpack

---

Produces a decompressed version of a packed soundfont.

```
BOOL BASS_MIDI_FontUnpack(  
    HFONT handle,  
    void *outfile,  
    DWORD flags  
);
```

## Parameters

handle The soundfont to unpack.

outfile Filename for the unpacked soundfont.

flags Any combination of these flags.

BASS\_UNICODE *outfile* is in UTF-16 form. Otherwise it is ANSI on Windows, and UTF-8 on other platforms.

**Return value**

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

## **Error codes**

BASS_ERROR_HANDLE	<i>handle</i> is not valid.
BASS_ERROR_NOTAVAIL	The soundfont is not packed.
BASS_ERROR_INIT	<a href="#"><u>BASS_Init</u></a> has not been successfully called - it needs to be to decode the samples.
BASS_ERROR_CODEC	The appropriate add-on to decode the samples is not loaded.
BASS_ERROR_CREATE	Could not create the output file, <i>outfile</i> .
BASS_ERROR_UNKNOWN	Some other mystery problem!

## **Remarks**

To unpack a soundfont, the appropriate BASS add-on needs to be loaded (via [BASS PluginLoad](#)) to decode the samples. For example, if the samples are FLAC encoded, BASSFLAC would need to be loaded. BASS also needs to have been initialized, using [BASS Init](#). For just unpacking a soundfont, the "no sound" device could be used.

A soundfont should not be unpacked while it is being used to render any MIDI streams, as that could delay the rendering.

[BASS MIDI FontGetInfo](#) can be used to check if a soundfont is packed.

## Example

Unpack a FLAC encoded soundfont.

```
BASS_PluginLoad("bassflac.dll", 0); // load FLAC plugin
HSOUNDFONT handle=BASS_MIDI_FontInit("afile.sf2pack", 0); // open s
BASS_MIDI_FontUnpack(handle, "afile.sf2", 0); // produce unpacked v
```

**See also**

[BASS\\_MIDI\\_FontInit](#), [BASS\\_MIDI\\_FontPack](#)

# BASS\_MIDI\_FONTINFO structure

---

Used with [BASS\\_MIDI\\_FontGetInfo](#) to retrieve information on a soundfont.

```
typedef struct {  
    char *name;  
    char *copyright;  
    char *comment;  
    DWORD presets;  
    DWORD samsize;  
    DWORD samload;  
    DWORD samtype;  
} BASS_MIDI_FONTINFO;
```

## Members

name	Name of the soundfont.
copyright	Copyright notice.
comment	Any comments.
presets	The number of presets/instruments in the soundfont.
samsize	The total size (in bytes) of the sample data in the soundfont.
samload	The amount of sample data currently loaded... -1 = the soundfont is memory mapped.
samtype	The BASS_CTYPE_STREAM_XXX format of the sample data if it is packed... -1 = unknown format (appropriate BASS add-on not loaded), 0 = not packed.

**Remarks**

The *name*, *copyright* and *comment* information might not be included in some SF2 files. Only the *presets*, *samload* and *samtype* members are available with SFZ files, with the *samtype* value reflecting the most recently loaded encoded sample (it is possible for different samples to use different encoding).

**See also**

[BASS\\_MIDI\\_FontGetInfo](#)

# BASS\_MIDI\_InFree

---

Frees a MIDI input device.

```
BOOL BASS_MIDI_InFree(  
    DWORD device  
);
```

**Parameters**

device The device to 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\_DEVICE *device* is invalid.

BASS\_ERROR\_INIT The device has not been initialized.

**See also**

[BASS\\_MIDI\\_InInit](#)

# BASS\_MIDI\_InGetDeviceInfo

---

Retrieves information on a MIDI input device.

```
BOOL BASS_MIDI_InGetDeviceInfo(  
    DWORD device,  
    BASS_MIDI_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.

**Platform-specific**

MIDI input is not available on Android.

**See also**

[BASS\\_MIDI\\_InInit](#)

# BASS\_MIDI\_InInit

---

Initializes a MIDI input device.

```
BOOL BASS_MIDI_InInit(  
    DWORD device,  
    MIDIINPROC *proc,  
    void *user  
);
```

## Parameters

- device The device to use... 0 = first. [BASS\\_MIDI\\_InGetDeviceInfo](#) can be used to enumerate the available devices.
- proc Callback function to receive MIDI data from the device.
- user User instance data to pass to the callback function.

**Return value**

If the device was successfully initialized, TRUE is returned, else FALSE is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

## **Error codes**

BASS_ERROR_DEVICE	<i>device</i> is invalid.
BASS_ERROR_ALREADY	The device has already been initialized. <a href="#">BASS_MIDI_InFree</a> must be called before it can be initialized again.
BASS_ERROR_NOTAVAIL	The device not available.
BASS_ERROR_UNKNOWN	Some other mystery problem!

**Platform-specific**

MIDI input is not available on Android.

**See also**

[BASS\\_MIDI\\_InFree](#), [BASS\\_MIDI\\_InGetDeviceInfo](#), [BASS\\_MIDI\\_InStart](#),  
[BASS\\_CONFIG\\_MIDI\\_IN\\_PORTS](#)

# BASS\_MIDI\_InStart

---

Starts a MIDI input device.

```
BOOL BASS_MIDI_InStart(  
    DWORD device  
);
```

## **Parameters**

device The device to 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_DEVICE	<i>device</i> is invalid.
BASS_ERROR_INIT	The device has not been initialized.
BASS_ERROR_UNKNOWN	Some other mystery problem!

**See also**

[BASS MIDI InStop](#)

# BASS\_MIDI\_InStop

---

Stops a MIDI input device.

```
BOOL BASS_MIDI_InStop(  
    DWORD device  
);
```

**Parameters**

device The device to stop.

**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	<i>device</i> is invalid.
BASS_ERROR_INIT	The device has not been initialized.
BASS_ERROR_UNKNOWN	Some other mystery problem!

**See also**

[BASS\\_MIDI\\_InFree](#), [BASS\\_MIDI\\_InStart](#)

# MIDIINPROC callback

---

User defined callback function to receive MIDI data.

```
void CALLBACK MidiInProc(  
    DWORD device,  
    double time,  
    const BYTE *buffer,  
    DWORD length,  
    void *user  
);
```

## Parameters

- device The MIDI input device that the data is from.
- time Timestamp, in seconds since [BASS\\_MIDI\\_InStart](#) was called.
- buffer Pointer to the MIDI data.
- length The amount of data in bytes.
- user The user instance data given when [BASS\\_MIDI\\_InInit](#) was called.

## Example

A callback function that forwards the received data to a MIDI stream.

```
stream=BASS_MIDI_StreamCreate(16, 0, 0); // create a MIDI stream to
...
void CALLBACK MidiInProc(DWORD device, double time, const BYTE *buf
{
    BASS_MIDI_StreamEvents(stream, BASS_MIDI_EVENTS_RAW, buffer, len
}
```

**See also**

[BASS\\_MIDI\\_InInit](#)

# BASS\_MIDI\_DEVICEINFO structure

---

Used with [BASS\\_MIDI\\_InGetDeviceInfo](#) to retrieve information on a MIDI input device.

```
typedef struct {  
    char *name;  
    DWORD id;  
    DWORD flags;  
} BASS_MIDI_DEVICEINFO;
```

## Members

name	Description of the device.
id	An identification number.
flags	The device's current status... a combination of these flags.
	<b>BASS_DEVICE_ENABLED</b> The device is enabled. It will not be possible to initialize the device if this flag is not present.
	<b>BASS_DEVICE_INIT</b> The device is initialized, ie. <a href="#"><u>BASS_MIDI_InInit</u></a> has been called.

### **Platform-specific**

On Windows, *id* consists of a manufacturer identifier in the LOWORD and a product identifier in the HIWORD. This will not uniquely identify a particular device, ie. multiple devices may have the same value. A list of identifiers is available from Microsoft, [here](#). On OSX, *id* is the device's "kMIDIPropertyUniqueID" property value, which is unique to the device. On Linux, *id* contains the device's ALSA client ID in the LOWORD and port ID in the HIWORD.

Depending on the [BASS\\_CONFIG\\_UNICODE](#) config setting, *name* can be in ANSI or UTF-8 form on Windows. It is always in UTF-8 form on other platforms.

**See also**

[BASS\\_MIDI\\_InGetDeviceInfo](#)

# BASS\_MIDI\_ConvertEvents

---

Converts raw MIDI data to [BASS\\_MIDI\\_EVENT](#) structures.

```
DWORD BASS_MIDI_ConvertEvents(  
    BYTE *data,  
    DWORD lengthBASS\_MIDI\_EVENT *events,  
    DWORD count,  
    DWORD flags  
);
```

## Parameters

data	The raw MIDI data.	
length	The length of the data.	
events	Pointer to an array to receive the events... NULL = get the number of events without getting the events themselves.	
count	The maximum number of events to convert.	
flags	A combination of these flags.	
	BASS_MIDI_EVENTS_NORSTATUS	Disable running status, meaning each event must include a status byte.
	BASS_MIDI_EVENTS_TIME	The raw MIDI data includes delta-time info.

**Return value**

If successful, the number of events processed is returned, else -1 is returned. Use [BASS\\_ErrorGetCode](#) to get the error code.

**Error codes**

BASS\_ERROR\_MEM            There is insufficient memory.  
BASS\_ERROR\_UNKNOWN      Some mystery problem!

## Example

Convert some raw MIDI data.

```
BYTE data[7]={0x90, 60, 100, 64, 100, 67, 100}; // the event data
BASS_MIDI_EVENTS events[3];
BASS_MIDI_ConvertEvents(data, 7, events, 3, 0); // convert the event
```

Convert the same data with delta-time info.

```
BYTE data[10]={0, 0x90, 60, 100, 0, 64, 100, 0, 67, 100}; // the event data
BASS_MIDI_EVENTS events[3];
BASS_MIDI_ConvertEvents(data, 10, events, 3, BASS_MIDI_EVENTS_TIME)
```

**See also**

[BASS\\_MIDI\\_StreamCreate](#), [BASS\\_MIDI\\_StreamEvent](#),  
[BASS\\_MIDI\\_StreamGetEvent](#)