Introduction

The FMOD Ex sound system is a revolutionary new audio engine for game developers, multimedia developers, sound designers, musicians and audio engineers, based on the years of experienced of Firelight Technologiesâ€™ previous product FMOD.
It also aims high - to push the boundaries of audio implementation for games and the like while at the same time using minimal resources and being scalable.

This new engine is written from the ground up since FMOD 3 was released and involves years of experience and feedback from FMOD users to create the most feature filled and easy to use product possible, without the drawbacks of legacy implementation that FMOD 3 may have suffered from its years of continuous development.

Some of the most exciting new features, which are described in more detail later are:

- Suite of **built in DSP special effects** which do not rely on any platform or operating system. (for a 100% cross platform audio experience). Includes **high quality I3DL2 compatible reverb**!
- **Next-gen console support**. PS3, Xbox 360 and Wii are fully supported.
- **Sound designer focus and tool**. The new suite of tools and functionality means FMOD is usable by sound designers and musicians and not just programmers. Sound authors will have the ability to create complex audio models and tweak them in real-time over the network (or even internet) while the game/application is still running!
- **Full 3D sound support** including linear/nonlinear/custom rolloff models, multiple listener support, occlusion and obstruction, sound cones, and support for stereo or multichannel samples being played in 3d!.
- **Geometry occlusion engine**. You can supply FMOD with a polygon scene and it will automatically occlude and obstruct direct path and reverb signals for you!
- **Virtual voices** to allow a game to play thousands of sounds at once on limited hardware without worrying about handling the logic to switch sounds off and on themselves.
- Support for **over 20 file formats**.

- **Advanced streaming engine** supporting gapless stitching/sentencing of sounds, low cpu overhead, multiple stream support, over-ridable file callbacks and more.
- **Compressed sample playback**. ADPCM, MPEG and XMA are able to be stored in memory without decompressing or streaming them, as if they were normal static samples!
- **Sub mixing** and channel groups.
- **2D / 3D sound morphing**. Set up a user supplied 5.1 or 7.1 2D mix, and morph between it and directional 3D sound! Great for entering and leaving volumetric sound sources.
- New **advanced 'DSP network' based software engine** to rival the most complex software synthesizer packages, all performed in real-time while the game/application is running! **Matrix panning** allows sound channels to be mapped to any speaker in any combination.
- New **object oriented API** supporting C, C++, C#, Delphi and Visual Basic.
- **Plug-in support** for ultimate flexibility. FMOD and VST plugins are supported. Everything in FMOD Ex has been designed with future expansion in mind.
- **SIMD optimized** (ie SSE, VMX, VFPU, ALTIVEC) mixing and filter routines for low cpu overhead. It is faster to use FMOD's software mixer than go through the driver overhead of DirectSound!

Platform support

FMOD Ex supports the following hardware platforms. No other audio system available supports this many platforms.

- Microsoft Windows series.
- Microsoft Windows series 64bit. (AMD64)
- Linux
- Linux 64bit. (AMD64)
- Macintosh. OS8 / 9 / X and OSX for x86.
- Sony PlayStation 2
- Microsoft Xbox
- Nintendo Gamecube
- Sony Playstation Portable.
- Microsoft Xbox 360.
- PlayStation 3.
- Nintendo Wii.

That's currently 12 platforms! No other game audio library can claim to match anywhere near that many platforms!

Feature list

Unified API.

Samples, streams, music and CD API's are gone. Everything is now a 'Sound'
All types of sounds, including mods, midi files, wavs, oggs, samples, streams, cd tracks and fsb files can be accessed seamlessly through the one API.

Virtual Channels

Virtual channels allow thousands of channels to play on limited hardware/software. Voices are swapped in and out according to 3d distance and priority.

Plug-in System

New file formats, output modes, and encoders can be added or downloaded by the user as DLLs.
VST and Winamp DSP plug-in support for effects is included.

Digital CD Playback

Digital CDDA playback allows dsp effects / spectrum analysis, ripping etc just as if it was a normal PCM file being played back.

C++ API

In FMOD Ex, a new C++ API is available as well as a standard C API.
All new FMOD API features are accessible through simple class types, such as the system class, sound class, channel class, DSP class. C/C++ headers naming conventions closely mapped.
For example - FMOD::System::init() in the C++ header would become FMOD_System_Init() in the C header.

C# and Visual Basic API

FMOD Ex has full support for managed C# and Visual Basic interfaces

Multiple simultaneous soundcard support

FMOD 3 was limited by only supporting 1 sound card at a time, so if you wanted to output to multiple cards at once you would have to instance fmod.dll multiple times.
Multiple output at once support is simply done by initializing multiple 'System' objects.

Multi-speaker output support

Now FMOD has a full multichannel mixer, even 2D sounds can be played in 5.1 (or 7.1!). Sounds can even swap their channel assignments around so left and right of a stereo sound are swapped around, mixed or all placed in the rear left speaker for example.
The way this is available is FMOD supports pan matrices. Any input sound channel can be redirected to any output speaker, and on top of this percentages/fractional levels are supported, so there are no absolute speaker assignments.

Via ASIO, FMOD Ex now also supports full multichannel output access to up to 16 output channels for high end sound devices.

Multi-speaker input support

Multichannel wavs, oggs and FSB files are supported for 5.1 music for example.

Low latency recording support

FMOD Ex now supports super low latency recording, processing and output through a new recording engine.
Via ASIO the recording->DSP->playback latency can be as low as 1-3ms! This is great for realtime processing and playback of recorded audio.

Enhanced Internet features

- Internet audio streaming. Custom internet streaming code is included, which allows for seamless SHOUTcast, Icecast and http streaming support.

- Download capability. A side effect of FMODâ€™s modular file system which supports network files, even static samples can be loaded off the internet.

In fact you can use FMODâ€™s API to write an arbitrary file downloader!

- Voice chat In a future version, sever/client voice chat will be supported for real-time over the internet voice conversations! Compression such as SPEEX etc will be supported for low bandwidth.

File format support

FMOD currently supports a wide range of audio file formats.

- AIFF - (Audio Interchange File Format)
- ASF - (Advanced Streaming format, includes support for the audio tracks in video streams)
- ASX - (playlist format - contains links to other audio files. To access contents, the FMOD Ex tag API is used)
- DLS - (DownLoadable Sound format for midi playback. Can also be used as a stand alone container format in FMOD)
- FLAC - (Lossless compression codec)
- FSB - (FMOD sample bank format generated by FSBank and FMOD designer tool)
- IT - (Impulse tracker sequenced mod format. FMOD Ex also fully supports resonant filters in .IT files, and the per channel or per instrument echo effect send, that can be enabled in ModPlug Tracker. This is **cross platform** effect support and does not require DirectX like other libraries do.)
- M3U - (playlist format - contains links to other audio files. To access contents, the FMOD Ex tag API is used)
- MID - MIDI using operating system or custom DLS patches.
- MOD - (Protracker / Fasttracker and others sequenced mod format)
- MP2 - (MPEG I/II Layer 2)
- MP3 - (MPEG I/II Layer 3, including VBR support)
- OGG - (Ogg Vorbis format)
- PLS - (playlist format - contains links to other audio files. To access contents, the FMOD Ex tag API is used)
- RAW - (Raw file format support. The user can specify the number of channels, bitdepth, format etc)
- S3M - (ScreamTracker 3 sequenced mod format)
- VAG - (PS2 / PSP format, playable on all platforms!)
- WAV - (Microsoft Wave files, inlcluding compressed wavs. PCM, MP3 and IMA ADPCM compressed wav files are supported across all platforms in FMOD Ex, and other compression formats are supported via windows codecs on that platform).
- WAX - (playlist format - contains links to other audio files. To access contents, the FMOD Ex tag API is used)
- WMA - (Windows Media Audio format)
- XM - (FastTracker 2 sequenced format)
- XMA - (Xbox 360 only)

File format plugins are also supported so the number of formats supported is limitless!
Note AAC is not included in FMOD Ex because the only reference source for this is GPL and FMOD Ex does not contain GPL protected code. To support this a user may add their own plugin to support it externally.

Wav Writer output

All output can be written to a wav file, and with encoder plug-ins, it can even be encoded in real-time to MP3 or other file formats!

Sample accurate seeking

Most systems seek to a compression block boundary such as mp3 which decodes in blocks of 1152 samples at a

time. FMOD Ex supports sample accurate seeking and decoding. For example you could seek to sample offset 1,000,000 exactly, and extract 1 sample of audio.
This accuracy is good for DJ type programs that need to sync streams properly.

Enhanced streaming engine

A new low latency stream decoder that spreads the decode burden over time instead of doing it in chunks (cpu spikes!) is included. This means smoother frame-rates in game.

Enhanced sample format support

24bit, 32bit integer and 32bit IEEE float sample support is included.
Alongside standard mono/stereo sample support, now multi-channel sample support is included!
Wav, ogg and user created sounds are examples of sound formats that support multi-channel sound.

Advanced mixing engine

- **Enhanced output channel support**

Most systems only allow mixing to mono or stereo output. FMOD Ex allows mixing to any number of output channels, for example 6 channel output (with panning) to allow for 5.1 or Dolby digital output in real-time for 3d sound!
Stereo and 5.1 are optimized as a special case fast-path for extra speed.

- **Full DSP data flow network based mixing engine.**

New mixing routines with separate resample/mix/effects stages.
This is a node based multiple input/output DSP engine which is extremely flexible and allows submixing, splitting and advanced speaker location and selection.

- **High quality mixing**

All mixing is floating point with full 32bit interpolation.
Resampling modes supported are
o No interpolation
o Linear interpolation
o Cubic interpolation
o 5 point spline interpolation!
All resampling is done with true 32bit precision using a 32bit fractional, it is not downscaled or compromised in any way.

- **Matrix Panning**

Sounds can have their input channels mapped to any output channel through a simple 2D matrix. For example the left and right parts of a stereo sound can be positioned anywhere in a 5.1 speaker array, in any combination, in one speaker, or all speakers. It is totally flexible.

- **Volume ramping**

Linear volume ramps between pan/volume changes are included as standard. This removes clicks in sound that changes pan or volume frequently.

3D Sound enhancements

- **Rolloff models.**

Logarithmic, linear, or custom rolloff models supported (per voice).

- **Geometry API.**

A revolutionary step up in audio realism is supported with FMOD Ex's custom geometry engine. This allows polygon scenes to be added to FMOD so that it can automatically calculate obstruction/occlusion as the user moves around the world.

- **Multiple listener support.**

Multiple 3d listeners for split screen support are supported.

- **Sound cone support.**

Sound cones are supported to give sounds direction.

- **Stereo / multichannel sound support.**

Stereo samples or even multichannel samples can be positioned in 3D, with their component channels (ie left/right parts of a stereo sound) positioned in 3D space, configurable by the user.

- **3D / 2D morphing.**

Now sounds can morph between being totally 3d directional point sources, and descrete 2D sources with speaker levels set by the user! This is great for entering and leaving a volumetric sound source. As an example, a stereo 3D sound can morph between being a directional point source, to a stereo 2D sound that envelopes you, then back again.

User delay on sound playback

A new 'setDelay' function is available so a sound can be specified to start after a certain period of time (samples or ms) - can be called between init and start on a channel

MIDI Support

FMOD Ex includes its own software midi playback, so that midi playback works cross platform.
Patch sets / DLS banks have to currently be provided with the song, or FMOD Ex will take advantage of any found in the operating system.

Stitching / sentencing

Seamless stitching, for sounds allows one sound to end then another starts immediately afterwards without gaps. This is great for commentary or interactive music.

Built in software based special effects.

FMOD Ex hosts a whole suite of special effects surpassing any system available considering it will work on every platform FMOD supports.
Here are some of the effects that will be supported as default. More can be added through plugins.

- Oscillators - sine, square, saw up, saw down, triangle and noise wave oscillators.
- 2 Low-pass with resonance filters.
- High-pass with resonance.
- 2 Echo filters.
- Flange.
- Distortion.
- Normalizer.
- Parametric EQ.
- Realtime pitch shifter (changes pitch not playback speed)
- Chorus.
- Freeverb simple reverb.
- SFX high quality I3DL2 compatible reverb.

Channel groups, and submixing.

Multiple channel groups can be created and channels assigned to these groups.
From there a variety of commands can be issued on a group such as volume, mute, frequency, pause and more.
Master volume can be controlled through the use of a channel group, and multiple channel groups can be used for multiple master volume assignments, which is very useful for things like relative volume of GUI sounds vs in game sounds for example, or music vs special effects volume.
This allows greater flexibility in controlling audio levels.

Submixing allows effects to be placed on groups of channels, without affecting other channels. This is an advanced feature which is really useful for saving CPU usage or keeping some sounds dry while others are affected by DSP effects for example.

Enhanced callback support

- 'latency adjusted' or 'real-time' flag for callbacks. This means you can get a callback at mix time, or audible time (the 2 are different, by the length of time determined by the mixer's buffer size)
- sample accurate user timer callbacks (ms or sample based) for global or per channel

Memory and filesystem overrides

FMOD Ex of course allows the user to override FMOD's file and memory system through callbacks.

FMOD Designer tool and API

- Sound Designer Tool

  This easy to use and flexible sound designer tool allows simple or complex multi-layer/effect/envelope based sound events to be modeled and created by the sound designer. The capabilities would include such things as layering, effects, random behaviour, and stitching of sounds.

  The aim is for a sound designer to totally design the in game audio from an external tool, and simply supply the programmer with assets and an event list to implement. If the audio behaviour needs to be changed within the game, it should be up to the sound designer not the programmer to do this.

  The layering screen allows for complex audio models (such as a car engine with multiple cross fading channels, sounds and effects) to be totally controlled by the author, then all the programmer has to do is call the previously defined set of simple commands, such as SendEvent and UpdateParameter. In the car model cast, the â€˜parameterâ€™ in UpdateParameter might just be â€˜revsâ€™ or â€˜torqueâ€™ or some other English type value, rather than a value defined by a programmer.

- FMOD Event API This is an API for programmers to interface to the data produced by the FMOD Designer tool. This API consists of very simple commands such as:

  o Init
  o Close
  o Load
  o GetEvent / Start
  o UpdateParameter
  All event behaviour is specified by the FMOD Designer tool, not the programmer, to make it totally data driven.

- Network tweaking features

  As part of the sound designer tool, the user can tweak the audio parameters in a game over the network while the game is running! A sound designer now gets even more control over the outcome of the audio mix by being able to alter sound parameters such as volume / frequency / randomization etc while the game is running. This will save hours of time instead of the usual routine of testing, quitting, tweaking, recompiling, running. Even with that old method it can lead to mistakes which take several attempts to perfect. Using the network tweaking tool the sound designer can get it right first time.

TERMINOLOGY / BASIC CONCEPTS.

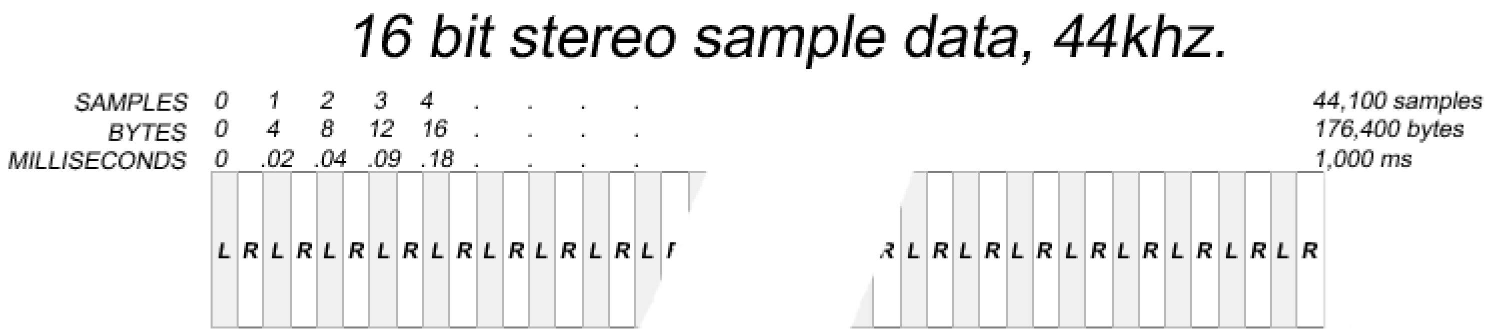
Introduction

Throughout FMOD documentation certain terms and concepts will be used. This section will explain some of these to alleviate confusion.

It is recommended when you see an API function highlighted as a link, that you check the API reference for more detail.

Samples vs bytes vs milliseconds

Within FMOD functions you will see references to PCM samples, bytes and milliseconds.
To understand what the difference is a diagram has been provided to show how raw PCM sample data is stored in FMOD buffers.


16 bit stereo sample data, 44khz.
SAMPLES 0 1 2 3 4
BYTES 0 4 8 12 16
MILLISECONDS 0 .02 .04 .09 .18
44,100 samples
176,400 bytes
1,000 ms
L R L R L R L R L R L R L R L R L
R L R L R L R L R L R L R L R L R L R

In this diagram you will see that a stereo sound has its left/right data interleaved one after the other.

- A left/right pair (a sound with 2 **channels**) is called a **sample**.
- Because this is made up of 16bit data, **1 sample = 4 bytes**.
- If the sample rate, or playback rate is 44.1khz, or 44100 samples per second, then **1 sample is 1/44100th of a second**, or **1/44th of a millisecond**. Therefore 44100 samples = 1 second or 1000ms worth of data.

To convert between the different terminologies, the following formulas can be used.

- **ms** = samples * 1000 / samplerate.
- **samples** = ms * samplerate / 1000.
- **samplerate** = samples * 1000 / ms.
- **bytes** = samples * bits * channels / 8.
- **samples** = bytes * 8 / bits / channels.

Some functions like Sound::getLength provide the length in milliseconds, bytes and samples to avoid needing to do these calculations.

Sounds. Samples vs compressed samples vs streams.

When a sound is loaded, it is either decompressed as a static sample into memory as PCM (samples), loaded into memory in its native format and decompressed at runtime (compressed samples), or streamed and decoded in realtime (in chunks) from an external media such as a harddisk or CD (streams).

- "**Samples**" are good for small sounds that need to be played more than once at a time, for example sound effects. These generally use little or no CPU to play back and can be hardware accelerated. See FMOD_CREATESAMPLE.
- "**Streams**" are good for large sounds that are too large to fit into memory and need to be streamed from disk into a small ringbuffer that FMOD manages. These take a small amount of CPU and disk bandwidth based on the file format. For example mp3 takes more cpu power to decode in real-time than a PCM decompressed wav file does. A streaming sound can only be played once, not multiple times due to it only having 1 file handle per stream and 1 ringbuffer to decode into. See FMOD_CREATESTREAM.
- "**Compressed samples**" are a new advanced option that allows the user to load a certain compressed file format (such as IMA ADPCM, MP2, MP3 and XMA formats currently), and leave them compressed in memory without decompressing them. They are software mixed on the CPU and don't have the 'once only' limitation of streams. They take more cpu than a standard PCM sample, but actually less than a stream due to not doing any disk access and much smaller memory buffers. See FMOD_CREATECOMPRESSEDSAMPLE.

You may notice "Sample" and "Stream" terminology is used here but there is no class name with this terminology in them. That is because all FMOD APIs are now consolidated into one "Sound" type.
By default System::createSound **will want to decode the whole sound fully into memory** (ie, as a decompressed sample).
To have it stream in realtime and save memory, use the FMOD_CREATESTREAM flag when creating a sound, or use the helper function System::createStream which is essentially the same as System::createSound but just has the FMOD_CREATESTREAM flag added in automatically for you.
To make a compressed sample use System::createSound with FMOD_CREATECOMPRESSEDSAMPLE.

Hardware vs Software

FMOD Ex has its support for either hardware accelerated sound playback, via DirectSound or console hardware API's, but FMOD also has its own fallback software mixing mechanism.
With hardware and software based sounds comes certain features and trade-offs when they are used.
Hardware sounds (created with FMOD_HARDWARE usually have lower CPU impact, have lower latency, and can get access to hardware reverb like EAX4 for example.
Hardware sounds are also limited in some ways, for example due to DirectSound limitations on Windows for example, arbitrary loop points are not supported with static samples (it is either loop the whole sound, or don't loop the sample), and non reverb effects cannot be played on them (ie chorus, distortion, lowpass etc).

Software sounds (created with FMOD_SOFTWARE sometimes have higher CPU impact, but can do much more, for example complex looping, realtime analysis, effects and sample accurate synchronization.

Hardware vs Software.

Hardware Pros.

- Usually lower latency. (Although on consoles or ASIO output in windows, using FMOD_SOFTWARE can have extremely low latency as low as 2-5ms)
- Less CPU time. (Although on Windows software is a lot faster due to bad hardware sound card driver design, and inefficiencies in the DirectSound API).
- On Windows, access to EAX2, EAX3, EAX4, I3DL2 reverb per voice. (FMOD Ex has its own high quality I3DL2 reverb solution in software, but may not be as flexible or have the quality of EAX4 for example.).
- Free hardware obstruction / occlusion (this is usually equivalent to a lowpass filter or reverb attenuation which can also be performed in software at some expense to the CPU), but only on EAX compatible sound cards on Windows. FMOD_SOFTWARE is cross platform.
- On PS2, PSP, XBox, GameCube, Wii, hardware voices can play back ADPCM compressed sound data with

no cpu hit.

- On a limited number of soundcards, hardware 3d sounds will be realtime encoded into an AC3 Dolby Digital stream via a digital / optical output on the card so an amplifier can play it in 3D surround sound. FMOD software mixing now supports 5.1 and 7.1 mixing at slightly higher CPU expense, and will work via analog outputs such as soundcards with 3 stereo jacks to run to a 5.1 speaker setup.

Hardware Cons.

- No point to point looping on win32. XBox and GameCube allow point to point looping and PS2 only allows loopstart, so therefore cross platform compatibility cannot be assured.
- No access to hardware effects per voice. Most PC sound cards and consoles do not support hardware accelerated effects such as lowpass, distortion, flange, chorus etc.
- No loop count control. A sound can only be looped infinitely or not at all.
- Inconsistent feature support, for example a PS2 does not support EAX reverb, and 3d sound implementations always sound different.
- Sometimes a lot slower than FMOD software mixing on Windows. Virtual voices that make a lot of state changes when swapping in and out can be very expensive in hardware (noticable framerate drops), but for free in software.

Software Pros.

- Consistent sound on every platform, there is no variation in playback.
- Sample accurate synchronization callbacks and events.
- Compressed sample playback support without using streams.
- Cross platform reverb.
- Complex looping and loop counts.
- Reverse sample playback.
- Spectrum analysis.
- Filters per channel or for the global mix, to perform effects such as lowpass, distortion, flange, chorus etc.
- Complex DSP network construction for realtime sound synthesis.
- Access to final mix buffer to allow analyzing, drawing to screen, or saving to file.

Software Cons.

- Latency on some sound devices (such as win32 waveout output) can be high.
- Memory usage is higher due to allocation of mix units and mix buffers, or simply the fact of having to store sounds in main ram rather than sound ram. (becoming less relevant these days).

Channels and sounds.

When you have loaded your sounds, you will want to play them. When you play them you will use System::playSound, which will return you a pointer to a Channel / FMOD_CHANNEL handle.
The index that System::playSound requires is generally recommended to always be FMOD_CHANNEL_FREE. This will mean FMOD will choose a non playing channel for you to play on.

2D vs 3D.

A 3D sound **source** is a channel that has a position and a velocity. When a 3D channel is playing, its volume, speaker placement and pitch will be affected automatically based on the relation to the **listener**.
A **listener** is the player, or the game camera. It has a position, velocity like a sound **source**, but it also has an *orientation*.

The **listener** and the **source** distance from each other determine the *volume*.
The **listener** and the **source** relative velocity determines the *pitch* (doppler effect).

The orientation of the **listener** to the **source** determines the *pan* or *speaker placement*.

A 2D sound is simply different in that it is not affected by the 3D sound **listener**, and does not have doppler or attenuation or speaker placement affected by it.
A 2D sound can call Channel::setSpeakerMix, Channel::setSpeakerLevels or Channel::setPan, whereas a 3D sound cannot.
A 3D sound can call any function with the word **3D** in the function name, whereas a 2D sound cannot.

For a more detailed description of 3D sound, read the tutorial in the documentation on 3D sound.

GETTING STARTED.

Introduction

The FMOD Ex API has been designed to be intuitive and flexible. In this tutorial an introduction to using the engine as well as the key issues involved in using it effectively will be explained.

Set up. What to include and what to link.

See **"Platform specific issues"** in this documentation to see what files to link into your project to make FMOD Ex function for each platform.

In C/C++, include "**fmod.h**" if you want to use the C interface only. Include "**fmod.hpp**" if you want to use the C++ interface.
Note that the constants, callbacks, defines and enums are stored within fmod.h, so fmod.hpp includes fmod.h. If you are using C++ you will be interchanging between both.

For Delphi, C# and Visual Basic, you will see equivalent headers to use in your application.

Initialization.

The simplest way to initialize fmod is to simply call System::init. Thats it. FMOD will set up the soundcard and other factors using default parameters.

When looking at the documentation for System::init, remember that the **maxchannels** parameter is the number of simultaneous voices you would like to be played in your game at once. This is nothing to do with how many hardware voices the soundcard may have, or how many software mixed voices there may be available.

These voices are virtual voices. This means you can play as many sounds as you want at once and not worry about the issue of hardware or software resources available.
You can safely play EVERY sound in your game simultanously without fear of System::playSound running out of voices or stealing other playing voices, and for this reason, it is acceptable to set maxchannels to a high number. 1, 100, 200, 1000. It is up to you and your type of title.
Note 1000 voices playing at once does not negatively impact performance because the majority of those will not be audible (non audible voices are 'virtualized'). There is only a small cost in sorting and swapping those voices as the FMOD Ex virtual voice manager controls which voices are heard and which aren't.
Let's have a look at an example of initializing FMOD Ex.

```
 FMOD_RESULT result;
 FMOD::System *system;

 result = FMOD::System_Create(&system);    // Create the main system object.
if (result != FMOD_OK)
{
     printf("FMOD error! (%d) %s\n", result, FMOD_ErrorString(result));
    exit(-1);
}

 result = system->init(100, FMOD_INIT_NORMAL, 0);    // Initialize FMOD.
if (result != FMOD_OK)
{
     printf("FMOD error! (%d) %s\n", result, FMOD_ErrorString(result));
    exit(-1);
```

}
Here we have the most basic setup of the FMOD engine. It will use 100 virtual voices.
Note that mod/s3m/xm/it/midi formats use 1 voice when playing. Do not extend the voice count here thinking it will give more voices to these file formats when playing, because they won't. These formats have their own internal pool voices that they use.

Configuration options

The output hardware, FMOD's resource usage, and other types of configuration options can be set if you desire behaviour differing from the default.
These are generally called before System::init.
The main ones are.

- System::setOutput - To choose an alternative output method. For example you can choose between DirectSound, WinMM, ASIO, no-sound, wave-writer or a number of other output options in windows. Each platform will have their own output choices. Don't call this unless you need to. You don't need to call it especially if all you are doing is setting the default. That would be pointless.
- System::setDriver - To choose an alternative sound card driver for a particular output mode. This is useful if you have multiple sound cards and want to choose one beside the default. Again, don't bother calling this if all you are doing is setting it to the default. You should enumerate devices with System::getNumDrivers and System::getDriverInfo if you want to give the user the choice.
- System::setHardwareChannels - Call this if you want to limit the number of audible hardware voices, or request that a minimum number of hardware voices be available before reverting to 100% software mixed voice support. The 'minimum' option is to guarantee a certain number of voices are audible at once.
- System::setSoftwareChannels - Call this if you want to set a different number of audible software mixed voices used by FMOD Channels. This will be purely for polyphony reasons or CPU / memory resource usage reasons. Do not adjust this thinking it will give more voices to mod/s3m/xm/it/midi formats. They do not use this channel pool and have their own internally.
- System::setSoftwareFormat - Call this to change settings in the FMOD software mixer. This includes sample rate, output format (ie integer vs float), output channel count (ie for multi-output channel asio devices for example), memory usage and mixing quality.
- System::setDSPBufferSize - Call this only if there are issues with stuttering on slow machines or bad soundcard drivers. This will affect software mixing latency, and can have adverse effects if misused. Some titles may want to let the user select between 'low latency' and 'compatible' modes, so they can trade off latency to audible stability by adjusting the buffersize.
- System::setSpeakerMode - Call this to set the output speaker mode. This only affects the FMOD software mixing engine. The default is stereo (5.1 on xbox and xbox360 and 7.1 on ps3), and can be changed if desired. Note speaker modes with higher channel counts leads to higher memory usage.

Here is an example of initializing FMOD with some configuration options. **Remember these options are just that. Optional! Do not call these if you don't need to and don't just cut and paste this code without knowing what it does!** For example you can't just go setting the speaker mode to 5.1 if the user doesn't have a 5.1 speaker system!

```
 FMOD_RESULT result;
 FMOD::System *system;

 result = FMOD::System_Create(&system); // Create the main system object.
ERRCHECK(result);

 result = system->setSpeakerMode(FMOD_SPEAKERMODE_5POINT1); // Set the output to 5.1.
ERRCHECK(result);

 result = system->setSoftwareChannels(100); // Allow 100 software mixed voices to be
audible at once.
ERRCHECK(result);

 result = system->setHardwareChannels(32, 64, 32, 64); // Require the soundcard to have at
```

```
has 32 2D and 32 3D hardware voices, and clamp it to using 64 if it has more than this.
ERRCHECK(result);

result = system->init(200, FMOD_INIT_NORMAL, 0); // Initialize FMOD with 200 virtual voices.
ERRCHECK(result);
```

Loading and playing.

To play the sounds you must load them first!
To do this, use System::createSound or System::createStream.
A sound by default will try to decompress the whole sound into memory (if System::createSound is used), that is why if the sound is large, it is better to stream it (by using System::createStream) which means it will decode at runtime, with a small fixed size memory buffer, and not use the memory a sample would.
For more on this see the Terminology/Basic Concepts tutorial.

Here is an example of loading an mp3 file. By default System::createSound will decompress the whole MP3 into 16bit PCM. This could mean the amount of memory used is many times more than the size of the file.

```
FMOD::Sound *sound;
result = system->createSound("../media/wave.mp3", FMOD_DEFAULT, 0, &sound); // FMOD_DEFAULT uses the defaults. These are the same as FMOD_LOOP_OFF | FMOD_2D | FMOD_HARDWARE.
ERRCHECK(result);
```

Here is an example of opening an mp3 file to be streamed. System::createStream will open the file, and pre-buffer a small amount of data so that it will be able to play instantly when System::playSound is called.

```
FMOD::Sound *sound;
result = system->createStream("../media/wave.mp3", FMOD_DEFAULT, 0, &sound); // FMOD_DEFAULT uses the defaults. These are the same as FMOD_LOOP_OFF | FMOD_2D | FMOD_HARDWARE.
ERRCHECK(result);
```

To specifically make a sound software mixed, you must use FMOD_SOFTWARE. This is necessary if you want to use things such as DSP effects, spectrum analysis, getwavedata, point to point looping and other more advanced techniques.

```
FMOD::Sound *sound;
result = system->createSound("../media/wave.mp3", FMOD_SOFTWARE, 0, &sound); // Make the sound software mixed.
ERRCHECK(result);
```

Here is an example of loading an mp3 file into memory as a sample, but not decompressing it when it loads, using the use FMOD_CREATECOMPRESSEDSAMPLE flag. This will automatically make the sound software mixed if FMOD_HARDWARE or FMOD_SOFTWARE is not specified. *Hardware* sound playback cannot support this flag unless the format is ADPCM on Xbox, VAG on PS2/PSP and GCADPCM on Gamecube/Wii. Platforms like PS3 and Xbox 360 are all done one the cpu (usually a different core to the main cpu so it does not affect performance).

```
FMOD::Sound *sound;
result = system->createSound("../media/wave.mp3", FMOD_CREATECOMPRESSEDSAMPLE, 0, &sound); // FMOD_CREATECOMPRESSEDSAMPLE tells the sample to attempt playing it as it is, without decompressing it into memory first. This is only supported for IMA ADPCM, MP2, MP3 and XMA audio formats.
ERRCHECK(result);
```

Warning! This mode is to be used with care. It acts just like a PCM sample, but incurs a heavier CPU cost at runtime. FMOD decodes the sound from its compressed format as it plays it.

Now to play the sound or stream. This is as simple as calling System::playSound.

```
 FMOD::Channel *channel;
 result = system->playSound(FMOD_CHANNEL_FREE, sound, false, &channel);
ERRCHECK(result);
```

This sound is now playing in the background! Your app will continue on from this point.

Things to note about playSound.

- **You do not need to store the channel handle** if you do not want to. That parameter can be 0 or NULL. This is useful if you don't care about updating that instance of the sound, and if it is a one shot sound (ie it does not loop). For example

```
      FMOD::Channel *channel;
 result = system->playSound(FMOD_CHANNEL_FREE, sound, false, 0);
ERRCHECK(result);
```

- **You can start the sound paused**, so you can update its attributes without the change being audible. That is what the 'paused' parameter is used for. For example, if you set it to true, set the volume to 0.5, then unpaused it, the sound would play at half volume. If you had set the paused flag to false and executed the same logic, you may hear the sound play at full volume for a fraction of a second. This can be undesirable.

```
      FMOD::Channel *channel;
 result = system->playSound(FMOD_CHANNEL_FREE, sound, true, &channel);
ERRCHECK(result);
 result = channel->setVolume(0.5f); // Set the volume while it is paused.
ERRCHECK(result);
 result = channel->setPaused(false); // This is where the sound really starts.
ERRCHECK(result);
```

- **A 'channel' is an instance of a sound.** You can play a sound many times at once, and each time you play a sound you will get a new channel handle. Not ethat this is only if it is not a stream. Streams can only be played once at a time, and if you attempt to play it multiple times, it will simply restart the existing stream and return the same handle that it was using before. This is because streams only have 1 stream buffer, and 1 file handle. To play a stream twice at once, open and play it twice.
- **Always use** FMOD_CHANNEL_FREE. This lets FMOD pick the channels for you, meaning that it uses FMOD's channel manager to pick a non playing channel. FMOD_CHANNEL_REUSE can be used if the desired effect is to pass in an existing channel handle and use that for the playsound. It can be used to stop a sound spawning a new instance every time System::playSound is called, and only play once at a time.
- **You do not have to 'free' or 'release' a channel handle.** Channels come from a pool which you created by specifying a channel count in System::init. Channel handles get re-used if old sounds have stopped on them. If all channels are playing, then one of the existing channels will get stolen based on the lowest priority sound. Make sure this doesnt happen by simply increasing the channel count in System::init.
- **A channel becomes invalid once it is finished playing.** This means you can't update it, and doing so would be pointless anyway becuase it isn't going to start again. Referencing a stopped channel will most likely result in an FMOD_ERR_INVALID_HANDLE.

Update. (This is important!)

It is important that System::update be called once per frame. Do not call this more than once per frame, as this is not necessary and is just inefficient.

This function updates the following aspects of FMOD Ex.

- **Platform specific routines** such as the once a frame command packet send to the IOP on the PlayStation 2. Without the update no sound would be audible on this platform.
- **Virtual voice emulation**. Without update being called, virtual voices would pause.
- **3D voice calculation**. If update is not called, sounds will not audibly move in 3D even though the channel or listener has been had its 3D attributes set.
- **Geometry engine**. The FMOD polygon/geometry engine is updated from this function. Without it, the occlusion/obstruction properties defined by the user will not be audible.

- **Non realtime output**. FMOD_OUTPUTTYPE_NOSOUND_NRT and FMOD_OUTPUTTYPE_WAVWRITER_NRT need this function to be called to update to the output. (ie write to the file in FMOD_OUTPUTTYPE_WAVWRITER_NRT).
- **Streaming engine**, if FMOD_INIT_STREAM_FROM_UPDATE is specified. If the user has decided to drive the streaming engine themselves from the main thread, then update must be called regularly or the streamer will stutter and cause buffer underrun.

Shutdown.

Call System::release to close the output device and free all memory associated with that object.
Channels are stopped, but sounds are *not* released. You will have to free them first. You do not have to stop channels yourself.
You can of course do it if you want, it is just redundant, but releasing sounds is good programming practice anyway.
You do not have to call System::close if you are releasing the system object. System::release internally calls System::close anyway.

Resource usage configuration.

In application development, some developers will want to have all disk or memory access going through their own functions rather than using the default system.
In FMOD Ex, you can configure the FMOD file system to use your own file routines with System::setFileSystem.

To make FMOD use your memory system, or to confine FMOD to 1 block of memory that it will not allocate outside of, use Memory_Initialize.

Note! On Xbox and XBox 360 it is actually required for the user to provide FMOD with a block of memory. On Xbox 360 this memory must be allocated with **XPhysicalAlloc**. See **"Platform specific issues"** for more.

TRANSITIONING BETWEEN FMOD 3 AND FMOD EX. API DIFFERENCES

Introduction

This section will describe some of the differences between FMOD 3 and FMOD Ex, if you are used to the old API and have difficulty understanding the difference between the 2 APIs.
It will answer some of the more common questions usually beginning with "What happened to.."

FMOD 3 had streams, sample and music APIs, now what?

All combined into the one class **Sound**. This leads to a much leaner and streamlined API. To create a stream just use System::createStream or FMOD_CREATESTREAM flag with System::createSound.
Music files loaded with the old music api would just be opened as a stream. As you could now load these types of sounds as a static sample (yes you can decode a whole mod into memory as PCM) it would possibly take hundreds of megabytes of ram, so even if you specify System::createSound to load a mod/s3m/xm/it file, it will still open it as a stream. To force it to a sample (not used as often) simply use FMOD_CREATESAMPLE flag in System::createSound.
The old 'music' formats (mod/s3m/xm/it/midi) now being streams means you can also do cool things like place effects on music formats (Channel::addDSP), or treat them like a normal channel with Channel::setVolume / Channel::setFrequency etc and therefore can even 3d position them!.

Channels are now objects instead of just integer handles

FMOD Ex now takes a more object oriented approach than FMOD 3. Channel objects are still reference counted though! So if the channel handle you have is stolen, FMOD Ex will still know not to update the newly playing channel with commands issued from the old channel handle.
Channel stealing should be less prevalent now thanks to virtual channels. You can now allocate a pool of many hundreds or even thousands of channels which will never run out, and they all succeed when you try to play them all at once. This is thanks to FMOD Ex's new **virtual voice system**.

Volume and pan

FMOD 3 used volume 0-255 (silent to full volume) and pan 0-255 (left to right), but now FMOD Ex takes a floating point number for each.
FMOD Ex now uses **0.0 to 1.0** for volume (silent to full), and **-1.0 to +1.0** for pan (left to right, 0.0 = center).

Frequency

FMOD 3 used integer frequencies. FMOD Ex now uses floating point frequencies. Now you can get far greater accuracy for sound playback (ie you can now set 44100.5 instead of having to choose between 44100 and 44101) which is important when trying to do exact playback synchronization between 2 streams of different bpm for example.

FSOUND_GetError is gone

FMOD 3 used a global error code for determining what an error was. This was a pretty bad design choice, as internal and multithreaded FMOD calls could contaminate the global error code.
FMOD Ex now uses a much cleaner error return code for every single function. This is not affected by the previously mentioned issues.

What happened to FSOUND_SetSFXMasterVolume? or How do I perform master volume?

FMOD 3 used this function to scale all non music oriented channel volumes.
FMOD Ex now uses 'ChannelGroups' which are far more powerful, and to scale all channels by a master volume, just use System::getMasterChannelGroup then ChannelGroup::setVolume.
Using ChannelGroups you can now have multiple master volume groups, and other exciting features such as DSP submixing.

MOD/S3M/XM/IT channels used to take up channels in FMOD 3's main channel pool so I had to adjust FSOUND_Init, do I have to do this with System::init?

No. MOD/S3M/XM/IT (and now MIDI) have their own channel pools that do not affect the number in System::init. Just select a number of channels that YOU are going to use, don't worry about what FMOD Ex is doing internally. To you playing a MOD/S3M/XM/IT/MID uses **1** channel.

Where is FSOUND_SetHWND?

FMOD Ex is global focus, or windowless by default. If you really need to focus the audio on a particular window in FMOD_OUTPUTTYPE_DSOUND mode on win32/win64, pass the hwnd as the extradriverdata parameter in System::init

Where is FSOUND_GetCurrentLevels?

Use System::getWaveData or Channel::getWaveData. It is far more flexible.

FSOUND_Update is now System::update.

Call System::update once a frame in your game loop. This is nescessary to update various aspects of FMOD Ex.

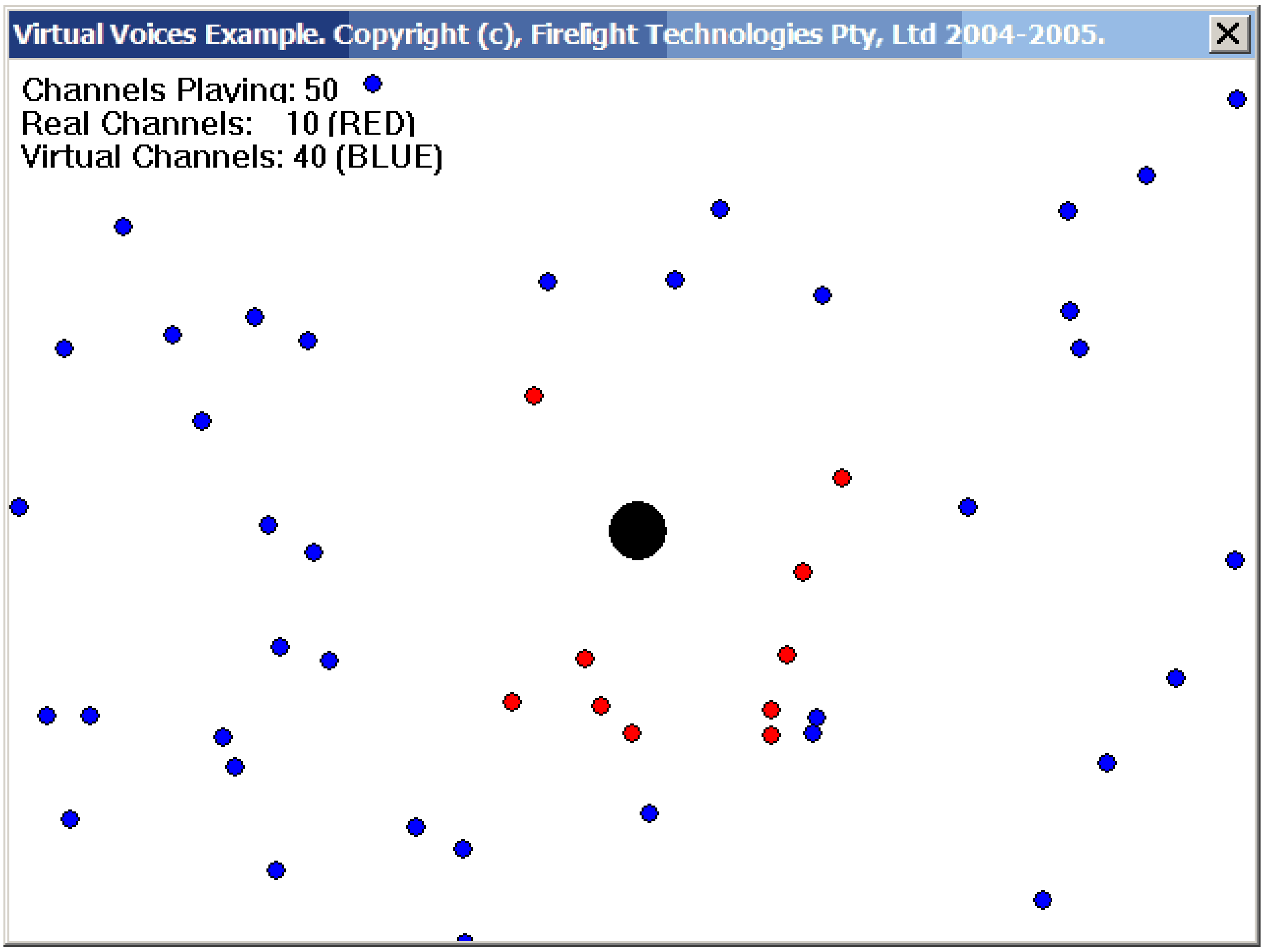
CHANNEL MANAGEMENT AND VIRTUAL VOICES.

Introduction

FMOD Ex now includes an efficient virtual voice management system. This tutorial will explain how it works and what is the advantage of using virtual voices.

What are virtual voices?

What is a virtual voice and how is this different to a hardware or software voice?


Virtual Voices Example. Copyright (c), Firelight Technologies Pty, Ltd 2004-2005.
Channels Playing: 50
Real Channels: 10 (RED)
Virtual Channels: 40 (BLUE)

Notice this screenshot of the FMOD Ex virtual voices example. It is playing 50 sounds at once, but only 10 are audible.
On limited sound hardware, which generally only has 32 to 64 voices, it can be challenging to manage your whole game's audio voice allocation when you want to have hundreds or even thousands of sounds playing at once in world (for example in a dungeon there might be 200 torches burning on walls in various places all playing a crackling burn noise).

FMOD Ex now allows the user to play as many sounds as they require, and will automatically allocate the limited number of hardware or software voices to the most important sounds to the listener.
This could mean in as in the above example, in a 3D world the 10 closest sounds are audible and the rest become

'virtual'.
Notice in the above screenshot red sounds are audible, and the blue sounds are inaudible and 'virtual'. According to the user though, there are actually 50 sounds playing at once.

'Virtual' voices are not allocated to a hardware or software voice, and are usually the least important sounds to the listener. These are 'emulated' voices.
They will update their play cursors and seem to be playing like a normal sound, but will not be audible.
As the user moves around the world, or a 'virtual' voice suddenly becomes more important than one that is actually audible, FMOD's virtual voice manager will swap the two voices, and the sound that was previously virtual will now become audible at its correct position in time.

A voice can be queried if it is virtual or not by using the Channel::isVirtual function. This is usually only for informational purposes.

What if some sounds are more important than others?

First we will take the case of 2D sounds that are all playing at the same volume. How do you make sure one sound stays audible and the others possibly become virtual if too many sounds are playing?
The answer is to use the Channel::setPriority or Sound::setDefaults function.

By making one sound have a higher priority than another, it will be given priority to be audible while its competitor will be swapped out and become virtual.
For example if there were 10 sounds playing and only 10 real voices, and an 11th voice wants to be played. If the new sound has a higher priority than the voices playing, it will be played as audible and one of the original 10 will become virtual, because the new sound is more important.
Important sounds should have higher priority and it is up to the user to decide if some sounds should be more important than others. An example of an important sound might be a 2D menu or GUI sound or beep that needs to be heard above all other sounds.

Volume of a sound is a secondary determining factor between sounds of equal priority. If a group of sounds have the same priority, the loudest sound will be the most important. In a 3d world this usually means the closest sounds will be more important and the further away sounds, or the quieter sounds will be less important and will possibly become virtual.

What if I run out of virtual voices?

If you try to play more sounds than there are virtual voices, then FMOD Ex channel manager will try to find the least important sound and replace it with the new sound. This means the channel that has been replaced will stop and become invalid.

If a channel handle that has been kicked out by a new channel becomes invalid, any commands that are used on that channel handle will return FMOD_ERR_INVALID_HANDLE.

How do I set the number of real voices and virtual voices?

To set the number of virtual voices FMOD Ex will use, call System::init with the number of virtual voices specified in the maxchannels parameter.

For hardware voices, generally you don't set the number of these available on a sound device, such as on a console or sound card. Usually you are provided with a number of hardware channels to use. For example, PlayStation 2 always has 48 hardware voices.

On a sound card, this is variable depending on the manufacturer. You can find out the number of available hardware channels with System::getHardwareChannels.
If you want to limit the number of hardware channels below its capacity, you can use System::setHardwareChannels.
This type of voice is used if the sound is created with FMOD_HARDWARE flag.

To set the number of software mixed channels available, use System::setSoftwareChannels. You can set this to 0 if you don't want any software mixed voices.
This type of voice is used if the sound is created with FMOD_SOFTWARE flag.

How many virtual voices should I set?

How many sounds are you trying to play at once without losing control of the channel handles?
This figure is up to you, but remember that more channels = more CPU and memory usage.

If you have 32 real hardware or software channels available to you and don't want to play more than this at once, then you might only need 32 virtual voices. This will mean a 1 to 1 relationship between real voices and virtual voices and sounds will never become emulated and be swapped out. Instead if you play more than the specified amount of channels, it will 'kick out' other lower priority channels.

If you have 32 real hardware or software channels available and you want to be able to safely play 100 at once, or 1000 at once, then set it to 100 or 1000 at once. Figures around the 1000 mark playing at once might start to show non negligible amounts of CPU and memory usage so be wary of this. Use System::getCPUUsage and FMOD::Memory_GetStats to determine this.

Can I make silent sounds go virtual?

Yes. To do this enable the FMOD_INIT_VOL0_BECOMES_VIRTUAL flag in System::init.

To configure this even further, you can change it from volume 0, to a higher volume, between 0 and 1. For example if you set the level for voices to go virtual at 0.1, everything below this audibility would go virtual. Warning if this is set too high, sounds may appear to 'cut out' before they are silent.
Use System::setAdvancedSettings, and the 'vol0virtualvol' member of FMOD_ADVANCEDSETTINGS.

How do I tell if a Channel is virtual or not?

See Channel::isVirtual

What if I don't like the sound of a voice going from virtual to real and playing half way through the sound, or near the end of the sound? (sounds like a bug!)

You can either use Sound or Channel priorities to stop it going virtual in the first place, or you have the option to have a voice start a from the beginning instead of half way through, by using the FMOD_VIRTUAL_PLAYFROMSTART flag with System::createSound, System::createStream, Sound::setMode or Channel::setMode.
As described above, only the quietest, least important sounds should be swapping in and out, so you shouldn't notice sounds 'swapping in', but if you have a low number of real voices, and they are all loud, then this behaviour could become more noticable and may sound bad.
Another option is to simply call Channel::isVirtual and stop the sound, but don't do this until after a System::update!
After playsound, the virtual voice sorting needs to be done in System::update to process what is really virtual and what isn't.

3D SOUND

Introduction.

This section will introduce you to using 3D sound with FMOD Ex. With it you can easily implement interactive 3D audio and have access to features such as 5.1 or 7.1 speaker output, and automatic attenuation, doppler and more advanced psychoacoustic 3D audio techniques.

Loading sounds as '3D'.

When loading a sound or sound bank, the sound must be created with System::createSound or System::createStream using the FMOD_3D flag.
ie.

```
    result = system->createSound("../media/drumloop.wav", FMOD_3D, 0, &sound);
    if (result != FMOD_OK)
    {
        HandleError(result);
    }
```

This will try and allocate a sound using hardware mixing by default. If there is no hardware mixing available, it will use software mixing as fallback.
To specifically load a sound in hardware or software simply add FMOD_HARDWARE or FMOD_SOFTWARE
ie.

```
    result = system->createSound("../media/drumloop.wav", (FMOD_MODE)(FMOD_HARDWARE | 
FMOD_3D), 0, &sound);
    if (result != FMOD_OK)
    {
        HandleError(result);
    }
```

Note that once the sound is loaded, on Win32 and FMOD_OUTPUTTYPE_DSOUND output (the default on Win32), you can't change the mode from FMOD_3D to FMOD_2D and vice versa. This is a limitation of DirectSound.
Using FMOD_SOFTWARE instead of FMOD_HARDWARE alleviates this issue, and other platforms that support hardware (ie Xbox, PS2, Gamecube) allow switching between 2D and 3D.

It is generally best not to try and switch between 3D and 2D at all, if you want though, you can change the sound or channel's mode to FMOD_3D_HEADRELATIVE at runtime which places the sound always relative to the listener, effectively sounding 2D as it will always follow the listener as the listener moves around.

Distance models and linear rolloff vs logarithmic.

Logarithmic

This is the default FMOD 3D distance model. All sounds naturally attenuate (fade out) in the real world using a logarithmic attenuation. The flag to set to this mode is FMOD_3D_LOGROLLOFF but if you're loading a sound you don't need to set this because it is the default. It is more for the purpose or resetting the mode back to the original if you set it to FMOD_3D_LINEARROLLOFF at some later stage.

When FMOD uses this model, **'mindistance'** of a sound / channel, is the distance that the sound *starts* to attenuate from. This can simulate the sound being smaller or larger. By default, for every doubling of this mindistance, the sound volume will halve. This rolloff rate can be changed with System::set3DSettings.

As an example of relative sound sizes, we can compare a bee and a jumbo jet. At only a meter or 2 away from a bee we will probably not hear it any more. In contrast, a jet will be heard from hundreds of meters away.
In this case we might set the bee's mindistance to 0.1 meters. After a few meters it should fall silent.
The jumbo jet's mindistance could be set to 50 meters. This could take many hundreds of meters of distance between listener and sound before it falls silent.
In this case we now have a more realistic representation of the loudness of the sound, even though each wave file has a fully normalized 16bit waveform within. (ie if you played them in 2D they would both be the same volume).

The **'maxdistance'** does not affect the rate of rolloff, it simply means the distance where the sound *stops* attenuating.
Don't set the maxdistance to a low number unless you want it to artificially stop attenuating. This is usually not wanted. Leave it at its default of 10000.0.

Linear

This is an alternative distance model that FMOD has introduced. It is supported by adding the FMOD_3D_LINEARROLLOFF flag to System::createSound or Sound::setMode / Channel::setMode.
This is a more fake, but usually more game programmer friendly method of attenuation. It allows the **'mindistance'** and **'maxdistance'** settings to change the attenuation behaviour to fading linearly between the two distances.
Effectively the mindistance is the same as the logarithmic method (ie the minimum distance before the sound starts to attenuate, otherwise it is full volume), but the maxdistance now becomes the point where the volume = 0 due to 3D distance.
The attenuation inbetween those 2 points is linear.

Some global 3D settings.

The 3 main configurable settings in FMOD Ex that affect all 3D sounds are:

- Doppler factor. This is just a way to exaggerate or minimize the doppler effect.
- Distance factor. This allows the user to set FMOD to use units that match their own (ie centimeters, meters, feet)
- Rolloff scale. Affects 3d sounds that use FMOD_3D_LOGROLLOFF. Controls how fast all sounds attenuate using this mode.

All 3 settings can be set with System::set3DSettings. Generally the user will not want to set these.

Velocity and keeping it frame rate independent.

Velocity is only required if you want doppler effects. Otherwise you can pass 0 or NULL to both System::set3DListenerAttributes and Channel::set3DAttributes for the velocity parameter, and no doppler effect will be heard.

This must be stressed again. It is important that the velocity passed to FMOD Ex is meters **per second** and not meters **per frame**. Notice the difference.
To get the correct velocity vector, use vectors from physics code etc, and don't just subtract last frames position from the current position. This is affected by framerate. The higher the framerate the smaller the position deltas, and therefore smaller doppler effects, which is incorrect.

If the only way you can get the velocity is to subtract this and last frame's position vectors, then remember to time adjust them from meters per frame back up to meters per second.
This is done simply by scaling the difference vector obtained by subtracting the 2 position vectors, by one over the frame time delta.

Here is an example.

```
vel.x = (pos.x - lastpos.x) * 1000 / timedelta;
vel.z = (pos.y - lastpos.y) * 1000 / timedelta;
vel.z = (pos.z - lastpos.z) * 1000 / timedelta;
```

timedelta is the time since the last frame in milliseconds. This can be obtained with functions such as timeGetTime(). So at 60fps, the timedelta would be 16.67ms. if the source moved 0.1 meters in this time, the actual velocity in meters per second would be:

```
vel = 0.1 * 1000 / 16.67 = 6 meters per second.
```

Similarly, if we only have half the framerate of 30fps, then subtracting position deltas will gives us twice the distance that it would at 60fps (so it would have moved 0.2 meters this time).

```
vel = 0.2 * 1000 / 33.33 = 6 meters per second.
```

Orientation and left-handed vs right-handed coordinate systems.

Getting the correct orientation set up is essential if you want the source to move around you in 3d space.
FMOD Uses a left handed coordinate system by default, (+X = right, +Y = up, +Z = forwards), which is the same as DirectSound3D and A3D.

If you use a different coordinate system, then you will need to flip certain axis or even swap them around inside the call to System::set3DListenerAttributes and Channel::set3DAttributes.
Take the right handed coordinate system, where +X = right, +Y = up, +Z = backwards or towards you. To convert this to FMOD coordinate system simply negate all instances of the Z coordinate for listener and sound position and velocity, as well as listener up and forward vector Z components.

To make things easier for people using the right handed coordinate system, you can initialize FMOD Ex using FMOD_INIT_3D_RIGHTHANDED in System::init and not do any conversion. FMOD will automatically convert its internal 3D calculations to be right handed instead of left handed.

A typical game loop.

3D sound and the FMOD channel management system need to be updated once per frame.
To do this use System::update
This would be a typical example of a game audio loop.

```
 do
{
    UpdateGame();          // here the game is updated and the sources would be moved with
channel->set3DAttributes.

    system->set3DListenerAttributes(0, ?      // update 'ears'

    system->update();    // needed to update 3d engine, once per frame.

} while (gamerunning);
```

Most games usually take the position,velocity and orientation from the camera's vectors and matrix.

Stereo and multichannel sounds can be 3D!

A stereo sound when played as 3d, will be split into 2 mono voices internally which are separately 3d positionable. Multi-channel sounds are also supported, so an 8 channel sound for example will allocate 8 mono voices internally in FMOD.
To rotate the left and right part of the stereo 3d sound in 3D space, use the Channel::set3DSpread function.

By default the subchannels position themselves in the same place, therefore sounding 'mono'.

Split screen / multiple listeners.

In some games, there may be a split screen mode. When it comes to audio, this means that FMOD Ex has to know about having more than 1 listener on the screen at once.
This is easily handled via System::set3DNumListeners and System::set3DListenerAttributes.

If you have 2 player split screen, then for each 'camera' or 'listener' simply call System::set3DListenerAttributes with 0 as the listener number of the first camera, and 1 for the listener number of the second camera.
System::set3DNumListeners would be set to 2.
That's all there is to it. You may notice an audible difference, because fmod does a few things to avoid confusion with the same sound being viewed from different viewpoints.

- 1. It turns off all doppler. This is because one listener might be going towards the sound, and another listener might be going away from the sound. To avoid confusion, the doppler is simply turned off.
- 2. All audio is mono. If to one listener the sound should be coming out of the left speaker, and to another listener it should be coming out of the right speaker, there will be a conflict, and more confusion, so all sounds are simply panned to the middle. This removes confusion.
- 3. Each sound is played only once as it would with a single player game, saving voice and cpu resources. This means the sound's effective audibility is determined by the closest listener to the sound. This makes sense as the sound should be the loudest to the nearest listener. Any listeners that are further away wouldn't have any impact on the volume at this point.

Speaker modes / output.

To get 5.1 sound is easy. If the sound card supports it, then any sound using FMOD_3D and FMOD_HARDWARE will automatically position itself in a surround speaker system, and only the user has to be sure that the speaker settings in the operating system are correct so that the sound device can output the audio in 5.1 or 7.1.
You do not need to call System::setSpeakerMode**!**. This function is only used to configure FMOD Ex's software mixing engine. See the next paragraph on this.

For sounds created with FMOD_SOFTWARE, by default sound is emulated through a simple stereo output. This involves panning and volume attenuation.
To enable FMOD software mixing to use 5.1 output, you can use System::setSpeakerMode. But note! This function increases the CPU mixing burden slightly as it now has to software mix into a 6 or 8 channel buffer instead of a stereo buffer.

FMOD_NONBLOCKING flag and asynchronously loading data

Introduction

FMOD_NONBLOCKING flag is used so that sounds can be loaded without affecting the framerate of the application.
Normally loading operations can take a large or significant amount of time, but with this feature, sounds can be loaded in the background without the application skipping a beat.

Creating the sound.

Simply create the sound as you normally would but add the FMOD_NONBLOCKING flag.

```
 FMOD::Sound *sound;
 result = system->createStream("../media/wave.mp3", FMOD_NONBLOCKING, 0, ? // Creates a
 handle to a stream then commands the FMOD Async loader to open the stream in the
 background.
ERRCHECK(result);
```

Now the sound will open in the background, and you will get a handle to the sound immediately. You cannot do anything with this sound handle except call Sound::getOpenState. Any other attempts to use this sound handle will result in the function returning FMOD_ERR_NOTREADY.

Getting a callback when the sound loads.

When the sound loads or the stream opens, you can specify a callback using the nonblockcallback member of the FMOD_CREATESOUNDEXINFO structure to make the non-blocking open seek to the subsound of your choice.
Firstly the callback definition.

```
 FMOD_RESULT F_CALLBACK nonblockcallback(FMOD_SOUND *sound, FMOD_RESULT result)
{
     FMOD::Sound *snd = (FMOD::Sound *)sound;

     printf("Sound loaded! (%d) %s\n", result, FMOD_ErrorString(result));

     return FMOD_OK;
}
```

And then the createSound call.

```
 FMOD_RESULT result;
 FMOD::Sound *sound;
 FMOD_CREATESOUNDEXINFO exinfo;

memset(?
exinfo.cbsize = sizeof(FMOD_CREATESOUNDEXINFO);
exinfo.nonblockcallback = nonblockcallback;

 result = system->createStream("../media/wave.mp3", FMOD_NONBLOCKING, ?
ERRCHECK(result);
```

Waiting for the sound to be ready and using it.

As mentioned, you will have to call Sound::getOpenState to wait for the sound to load in the background. You could

do this, or just continually try to call the function you want to call (ie System::playSound) until it succeeds.
Here is an example of polling the sound until it is ready, then playing it.

```
 FMOD_RESULT result;
 FMOD::Sound *sound;
 result = system->createStream("../media/wave.mp3", FMOD_NONBLOCKING, 0, &sound); // Creates a handle to a stream then commands the FMOD Async loader to open the stream in the background.
ERRCHECK(result);

 do
{
 FMOD_OPENSTATE state;

 result = tmpsound->getOpenState(&state);
ERRCHECK(result);

if (state == FMOD_OPENSTATE_READY){
 result = system->playSound(FMOD_CHANNEL_FREE, sound, false, &channel);
ERRCHECK(result);
}

GameCode();
} while (1)
```

or

```
 do
{
if (!channel)
{
 result = system->playSound(FMOD_CHANNEL_FREE, sound, false, &channel);
if (result != FMOD_ERR_NOTREADY)
{
ERRCHECK(result);
}
}

GameCode();
} while (1)
```

The second loop will simply retry playsound until it succeeds.

Creating the sound as a streamed FSB file.

An FSB file will have subsounds in it, so if you open it as a stream, you may not want FMOD seeking to the first subsound and wasting time. You can use the initialsubsound member of the FMOD_CREATESOUNDEXINFO structure to make the non-blocking open seek to the subsound of your choice.

```
 FMOD_RESULT result;
 FMOD::Sound *sound;
 FMOD_CREATESOUNDEXINFO exinfo;

memset(&exinfo, 0, sizeof(FMOD_CREATESOUNDEXINFO));
exinfo.cbsize = sizeof(FMOD_CREATESOUNDEXINFO);
exinfo.initialsubsound = 1;

 result = system->createStream("../media/sounds.fsb", FMOD_NONBLOCKING, &exinfo, &sound);
ERRCHECK(result);
```

Then get the subsound you wanted with Sound::getSubSound.

Getting a subsound.

Sound::getSubSound is a free function call normally, all it does is return a pointer to the subsound, whether it be a sample or a stream. It does not execute any special code besides this.

What it would cause if it was a blocking stream though, is System::playSound stalling several milliseconds or more while it seeks and reflushes the stream buffer. Time taken can depend on the file format and media.

If the parent sound was opened using FMOD_NONBLOCKING, then it will set the **subsound** to be FMOD_OPENSTATE_SEEKING and it will become not ready again until the seek and stream buffer flush has completed.
When the stream is ready and System::playSound is called, then the playsound will not stall and will execute immediately because the stream has been flushed.

MEMORY MANAGEMENT AND CONSERVATION TUTORIAL

Introduction

This section will give some pointers on how to use and save memory in FMOD Ex by describing things that may not be so obvious upon first looking at the API.

Using a fixed size memory pool.

To make FMOD stay inside a fixed size memory pool, and not do any external allocs, you can use the FMOD::Memory_Initialize function.
i.e.

```
    result = FMOD::Memory_Initialize(malloc(4*1024*1024), 4*1024*1024, 0,0,0); // allocate 4mb and pass it to FMODEx to use.
    ERRCHECK(result);
```

Note that this uses malloc. On Xbox 360 and Xbox you must use a different operating system alloc such as XPhysicalAlloc otherwise FMOD may not behave correctly. See "Platform specific issues" tutorials for more information on this.

Note that this function allows you to specify your own callbacks for alloc and free. In this case the memory pool pointer and length must be NULL. The 2 features are mutually exclusive.

Lowering sound instance overhead.

The FMOD_LOWMEM flag is used for users wanting to shave some memory usage off of the sound class.
This flag removes memory allocation for certain features like the 'name' field which isn't used often in games. When this happens, Sound::getName will return "(null)".
More memory will be stripped from the sound class in future versions of FMOD Ex when this flag is used. Currently the 'name' field is the biggest user of memory in the sound class so this has been removed first.

Using compressed samples.

To trade CPU usage vs Memory, FMOD Ex has a feature to play ADPCM, XMA and MP2/MP3 data compressed, without needing to decompress it to PCM first. This can save a large amount of memory.
On XBox 360, using this for XMA files incurs next to no extra CPU usage, as the Xbox 360 XMA hardware decoder does the data decompression in realtime.
To enable this use the FMOD_CREATECOMPRESSEDSAMPLE flag. If this flag is used for formats other than the ones specified above, it will be ignored.

With the exception of XMA on Xbox 360 and ADPCM on Xbox, if FMOD_CREATECOMPRESSEDSAMPLE is used with an FMOD_HARDWARE buffer it will generate an FMOD_ERR_NEEDSSOFTWARE error.

Note! If you use FMOD_CREATECOMPRESSEDSAMPLE there will be a 'one off' memory overhead to allocate the appropriate pool of codecs depending on the format being loaded. See the next section on how to control this pool.

Controlling memory usage with settings.

- System::setSoftwareFormat 'maxinputchannels' is default to 6 to allow up to 6 channel wav files to be played through FMOD's software engine. Setting this to a lower number will save memory across the board. If the highest channel count in a sound you are going to use is stereo, then set this to 2.
- For sounds created with FMOD_CREATECOMPRESSEDSAMPLE, System::setAdvancedSettings allows the user to reduce the number of simultaneous XMA/ADPCM or MPEG sounds played at once, to save memory. The defaults are specified in the documentation for this function. Lowering them will reduce memory. Note the pool of codecs for each codec type is only allocated when the first sound of that type is loaded. Reducing XMA to 0 when XMA is never used will not save any memory.
- For streams, setting System::setStreamBufferSize will control the memory usage for the stream buffer used by FMOD for each stream. Lowering the size in this function will reduce memory, but may also lead to stuttering streams. This is purely based on the type of media the FMOD streamer is reading from (ie CDROM is slower than harddisk), so it is to be experimented with based on this.
- Reducing the number of channels used will reduce memory. System::init and System::setSoftwareChannels give control over maximum number of virtual voices and software voices used. You will need to make sure you specify enough voices though to avoid channel stealing.

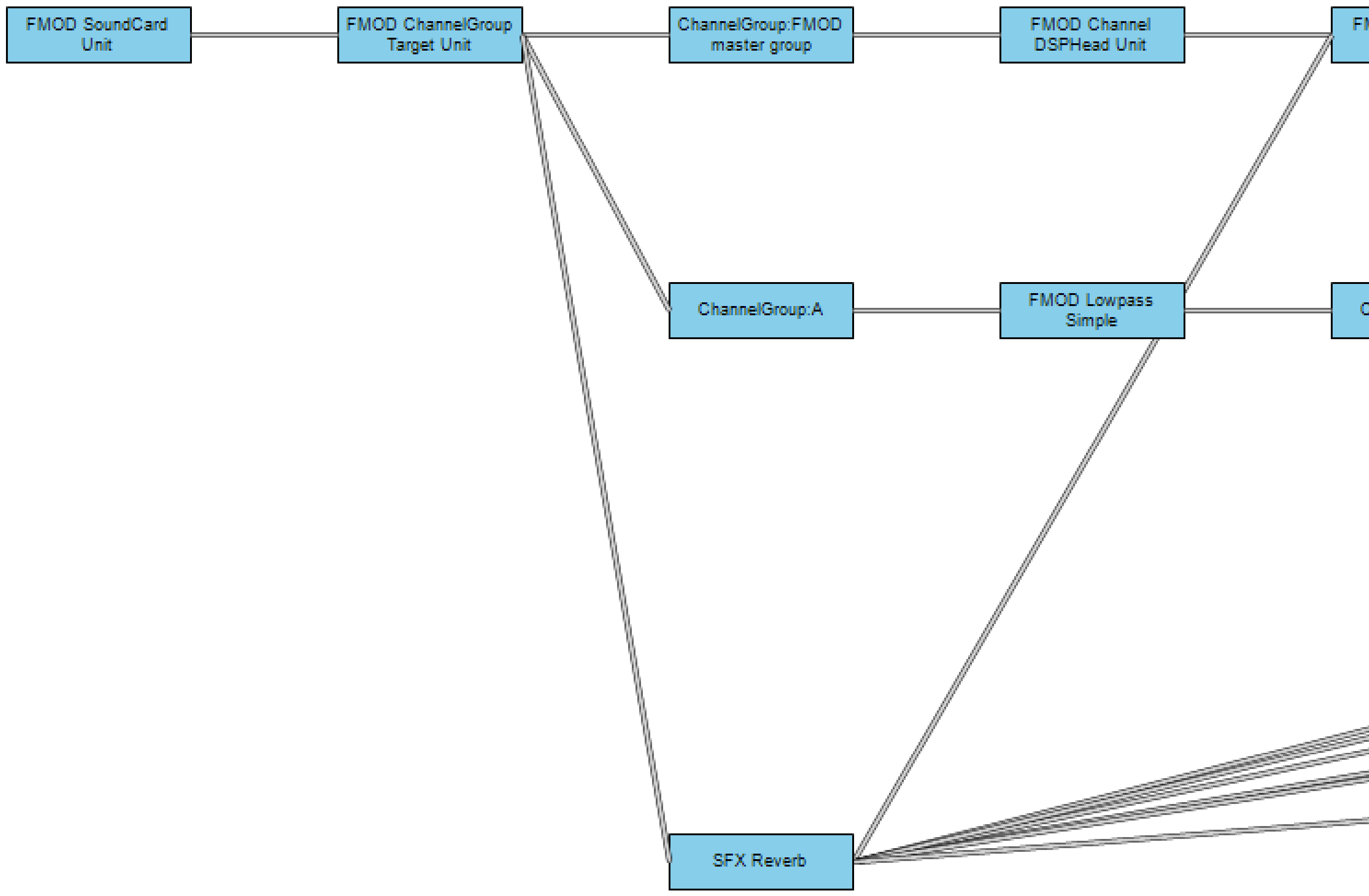
Tracking FMOD memory usage.

Using FMOD::Memory_GetStats is a good way to track FMOD memory usage, and also find the highest amount of memory allocated at any time, so you can adjust the fix memory pool size for the next time.

DSP TUTORIAL

Introduction

This section will introduce you to the FMOD Ex advanced DSP system. With this system you can do custom filters or complicated filter graph networks to create different and dynamic sounding audio.
The FMOD Ex DSP system is an ultimately flexible mixing engine, and goes far beyond FMOD 3's capabilities or any other audio mixing engine available right now.
Its emphasis on quality, flexibility and efficiency makes it an extremely powerful system if used to its full potential.


FMOD SoundCard Unit
FMOD ChannelGroup Target Unit
ChannelGroup:FMOD master group
FMOD Channel DSPHead Unit
ChannelGroup:A
FMOD Lowpass Simple
SFX Reverb

This is what an FMOD DSP network looks like. Audio data flows from the right to the left, until it finally arrives at the soundcard, fully mixed and processed.
The image was taken from a screenshot with the **FMOD DSPNet Listener** tool. You can run this on your own program as long as you specify FMOD_INIT_ENABLE_DSPNET. The tool is located in the /tools directory of the SDK.

Some notes on this example image. Terms in **bold** are nodes that you can reference in the picture.

- When multiple inputs converge into one unit, they are mixed together. This is a submix.
- 7 channels are playing. These are depicted by **FMOD Channel DSPHead Unit**. Any time System::playSound or System::playDSP is called, an **FMOD Channel DSPHead Unit** is attached to the network. When it is stopped, it is disconnected from the network.
- To the right of each DSPHead Unit, there is either an **FMOD WaveTable Unit**, an **FMOD DSP Codec** or an

FMOD Resampler Unit. A WaveTable unit is a unit that plays standard PCM data (ie a standard wav file), a DSP Codec is a compressed realtime sample, created by FMOD_CREATECOMPRESSEDSAMPLE (in this case an mp3), and a Resampler Unit is a generic buffered resampler that is usually used when connecting generic DSP units to a channel (wavetable and DSPcodec usually have a built in resampler, standard DSP units do not so one has to be inserted if it is to be played on a channel, so Channel::setFrequency can work). These things feed data into a **Channel DSPHead Unit**, then it feeds the data onto its parent and so on.

- The 3 channels that have an **FMOD WaveTable Unit**, are playing a pcm wave file, and have been connected to a channelgroup. **ChannelGroup: B**. This was done with Channel::setChannelGroup
- The 3 channels that have an **FMOD DSP Codec** Unit, are playing an mp3 file and are connected to a channelgroup. **ChannelGroup A**. This was done with Channel::setChannelGroup
- The channel that has **UNIT A** attached to it is not attached to a user created ChannelGroup. It is connected to the default system ChannelGroup called the "master channelgroup" which is the ChannelGroup all channels are played on if no other is specified. You can get a handle to this channelgroup with System::getMasterChannelGroup.
- The **UNIT A** unit was created by the user with System::createDSP and was played with System::playDSP. It does nothing but act as a submix target, so the user has added 3 oscillators by creating them with System::createDSPByType, and adding them to **UNIT A** with DSP::addInput.
- You may notice that **ChannelGroup: B** is also connected to **ChannelGroup: A** as an input. This means the result of the submix of **ChannelGroup: B** is fed into **ChannelGroup: A**. The connection of B to A was done with ChannelGroup::addGroup
- ChannelGroup A and ChannelGroup B have **2** boxes each. This is because they both have a *DSP effect applied to them*. Without a DSP effect added, it just has one box, for optimization reasons (less memory, less cpu usage). The first box on the left is the head node, the second box on the right with the same name is the **mix target** for the channels. Inbetween is an **FMOD Echo** DSP (FMOD_DSP_TYPE_ECHO) effect on **ChannelGroup: B**, and a **FMOD Lowpass Simple** (FMOD_DSP_TYPE_LOWPASS_SIMPLE) effect on **ChannelGroup: A**. These were added with ChannelGroup::addDSP
- Reverb has been enabled with System::setReverbProperties and can be seen as a DSP node called **SFX Reverb**. Notice all channel based DSP units have a connection going to it. This is the *wet path*. The other paths go directly to the soundcard and bypass the SFX Reverb unit, therefore it is the *dry path*. To control the wet/dry mix you can use the Channel::setReverbProperties. Internally this function just calls DSP::setInputMix. Only channels are interested in this unit. Things like channelgroups and other units do not need to connect to this unit.
- If there was no reverb enabled, the secondary links/outputs (on the **FMOD WaveTable Unit**/**FMOD DSP Codec**/**FMOD Resampler Unit** units) would be absent.
- At the end of the mix, **ChannelGroup: FMOD master group**, **ChannelGroup: A** and **SFX Reverb** all get submixed into the **FMOD ChannelGroup Target Unit**
- Finally the result of that submix gets sent to the **FMOD SoundCard Unit** which is the final destination.

- If a DSP effect was to be added with System::addDSP, it would be inserted between **FMOD SoundCard Unit** and **FMOD ChannelGroup Target Unit**. You could see why in that case it would affect all sound in the DSP network.
- If a DSP effect was to be added with Channel::addDSP, it would be inserted to the right of the **FMOD Channel DSPHead Unit**. If this happened you should be able to see why only the channel would be affected by the DSP effect.

Playing a sound and following the data flow.

When FMOD plays a *sound* on a channel (using System::playSound), it creates a small sub-network consisting of a **Channel DSP Head** and a **Wavetable Unit**.
When FMOD plays a *DSP* on a channel (System::playDSP), it creates a small sub-network consisting of a **Channel DSP Head** and a **Resampler Unit**. The DSP that was specified by the user is then attached to this as an input.
This section will describe the units in more detail, from the origin of the data through to the soundcard, from right to left.

Wavetable Unit.
This unit reads raw PCM data from the sound buffer and resamples it to the same rate as the soundcard. A **Wavetable Unit** is only connected when the user calls System::playSound.
After being resampled the audio data is then processing/flowing at the rate of the soundcard. This is 48khz by default.

Channel DSP Head.
This unit does nothing. It simply is a place for extra DSP effects to connect to, between the **Wavetable Unit** (if System::playSound was used), or a user specified DSP unit (if System::playDSP was used), and the **ChannelGroup Unit** that it belongs to. By default all channels connect to the **Master Channel Group**.
It is also the unit where the channel volume and pan gets applied.
A Channel DSP Head unit incurs no CPU penalty. The data is simply passed straight to its outputs.

ChannelGroup DSP Heads.
The **Master** ChannelGroup is the default target for **Channel DSP heads**, and is owned by the System object.
When multiple **Channel DSP Heads** are connected to a channel group, they are mixed together. This is the case for any DSP unit with multiple inputs.
Other channel groups may also be created by the user, which means channels may target them instead. This happens when the user calls Channel::setChannelGroup
Channelgroups are there for submixing. Effects can be placed after this point between it and the **ChannelGroup Target Unit**.

ChannelGroup Target Unit.
This is the target DSP unit for all ChannelGroups created by the user (with System::createChannelGroup) and the System ChannelGroup.

FMOD DESIGNER API PROGRAMMER'S TUTORIAL

Introduction.

This section provides more technical information on how to use the FMOD designer API, and how resource allocation is handled to allow the programmer to account for performance and memory issues.
Just to provide some background information, the whole FMOD designer API sits on top of the low level FMOD API. This means it contains an FMOD::System object and uses all of the low level functions of the FMOD api to achieve its functionality.

Files should you receive from the sound designer.

When the sound designer provides you with a project, they must provide you with the following files.

- 1 **.FEV** file. An FEV file is the compiled sound designer project which you will load with EventSystem::load.
- 1 or more **.FSB** files. These files are raw audio data. They do not contain any event or sound designer data.
- Optionally, a project report ***project name*.txt**. This is a file that describes the events to the programmer and any associated notes, along with the parameters for each event and their min/max values.
- Other files are working files (such as .cache), do not ship these.

The programmer must work with the sound designer to organize banks and event groups to conserve memory!

Event groups should be used to control loading strategies, they are not just for aesthetic purposes, they are for loading purposes.
The branches of an event tree are what you use to load when in the game code.
See the "**Event tree group strategies and loading / memory allocation issues**" section below for very important issues related to loading and memory usage.

Creating and initializing the EventSystem object.

Here is a typical bit of initialization code that you would call at the start of your project.

```
 FMOD::EventSystem  *eventsystem = 0;

 result = FMOD::EventSystem_Create(&eventsystem);
ERRCHECK(result);

 result = eventsystem->init(256, FMOD_INIT_NORMAL, 0);
ERRCHECK(result);
```

If you want to configure the lower level FMOD engine before initializing the event system (ie select sound card driver, set speaker mode etc), then call EventSystem::getSystemObject.

```
 FMOD::EventSystem *eventsystem = 0;
 FMOD::System *system = 0;

 result = FMOD::EventSystem_Create(&eventsystem);
ERRCHECK(result);

 result = eventsystem->getSystemObject(&system);
ERRCHECK(result);

 (..Use System API here from fmod.hpp..)

 result = eventsystem->init(256, FMOD_INIT_NORMAL, 0);
ERRCHECK(result);
```

Do not create your own FMOD::System object using FMOD::System_Create.

This will cause 2 system objects to be active (the one you just created plus the one within the EventSystem), which means it will try to open the sound device twice. It also means 2 software mixers would be spawned. This is A Bad Thing.

Also do not try to create multiple EventSystem objects. If you want to load multiple projects, simply load them from the one EventSystem.

Important memory management issue for consoles (Xbox, Xbox 360, PlayStation2, GameCube, PlayaStation Portable, PlayStation 3):

Memory management is a consideration that must be taken note of. On certain machines the default memory allocation is inefficient (ie the page size is way too big on Xbox 360 meaning megabytes of lost memory), or certain FMOD features just wont work without a memory pool (ie Xbox must have one single block of contiguous memory for audio buffers, and on Xbox 360 XMA buffers MUST reside within memory allocated with XPhysicalAlloc, otherwise the system will crash).

Before calling any fmod functions, first allocate a block of memory and pass it to FMOD. From then on it will not allocate any more memory itself.

```
 result = FMOD::Memory_Initialize(memblock, MEMSIZE, 0, 0, 0);
ERRCHECK(result);
```

On Xbox 360 use XPhysicalAlloc.

Note that is the memory FMOD uses for all audio data including wave data, unless it is a machine with dedicated sound ram such as PlayStation 2 and GameCube.
In general it is usually a good idea to start off with a large memory block, then use FMOD::Memory_GetStats to find out the maximum memory usage by FMOD during the progress of the game.
FMOD_ERR_MEMORY will be returned from FMOD functions if it runs out of memory.

Virtual Voices.
The value you pass to EventSystem::Init for the number of 'channels' should be a high number resembling the highest number of voices you want to have playing at once (not audible! there is a difference).

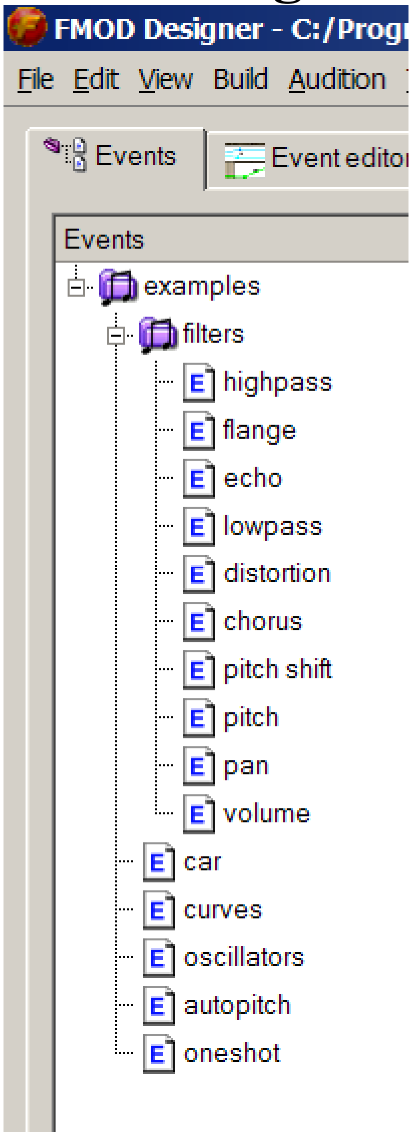
The hardware may only have 32 hardware voices, but this number can be 64, 128, 256 or 1000, because FMOD has a Virtual Voice System.
If you do not give this a high number, then voice stealing will take effect, and voices will drop out seemingly at random.
That is because new, more important sounds will be played, kicking out older voices.

With a high enough number of virtual voices, no voices will be stolen, and FMOD will automatically swap voices in and out based on distance and priority. See the virtual voice tutorial in the tutorial for more information on this.

Load the project.

Load the FEV file with EventSystem::load. This only allocates memory for the event tree structure. It does not allocate memory for sample data / wave bank data or even the memory for the even instances, which are the things you use to play and control the events later in the code.
At this point the memory usage should be low and the memory allocated is for the low level software mixing engine and low level channel structures etc.
You can load multiple FEV files with this function into **EventProjects**.

Traversing the event tree and getting events.



To traverse a tree you start at the root by calling EventProject::getGroup or EventProject::getGroupByIndex by specifying an event group at the root of the tree (ie 'examples' in this case).
From there you can call EventGroup::getGroup / EventGroup::getGroupByIndex to enter subgroups within the tree.
A full path can be entered within the GetGroup function, so that you don't have to manually traverse it yourself one at a time.

Finally you get a handle to an event with EventGroup::getEvent / EventGroup::getEventByIndex.

These functions have loading (disk access) and memory allocation issues, which need to be considered in the next section.

Event tree group strategies and loading / memory allocation issues. (Important!)

Terminology.

event - The leaf node in the tree. The thing you will obtain a handle to so that you can play it (and update its parameters).
event group - 'Folders' that contain events and other event groups. These are used for organizational and loading purposes.
wave banks - The .FSB files to be loaded. When memory allocation occurs for these, it means allocation for the raw PCM or compressed audio data.
event instance memory - Memory required to play the event(s). If an event has a 'max playbacks' value set in the designer tool, FMOD will allocate memory for that many instances, so that they can play simultaneously. Generally the memory footprint for this is small unless the sound designer has specified memory intensive DSP effects such as reverb, echo, chorus or flange. Other types of DSP effects generally do not allocate any memory (IIR effects such as lowpass filter, distortion).

Organization of event hierarchy and banks.

The event tree should be set up by grouping events into logical groups that will be loaded or used together, for example levels in a game, and common data.
The reason for this is that you can load an entire branch's audio data with EventGroup::loadEventData. This data in particular is the (usually) larger **wave bank data**. This means it will load the waves from the **FSB files** referenced by the events in the tree. If it has already loaded sounds from the same FSB referenced from another event group, it will of course not try to re-load them.

Note: Sounds are either loaded selectively from an FSB, meaning you can pretty much put every sound into 1 wavebank, or you, as a programmer may prefer to preload a whole wavebank at once (ie EventSystem::registerMemoryFSB) which means you may want to split waves into logical wavebank groups.

Loading / Allocation overview:

- EventSystem::load loads the FEV file, and only allocates a small amount of memory to hold the **event tree structure**.

- EventGroup::loadEventData loads all of the waves from the FSB files, necessary for the **specified group and its subgroups**. This function is recursive and traverses all subgroups.
- If you do not call EventGroup::loadEventData, FMOD loads the event's wave data, when it needs to when you call EventGroup::getEvent / EventGroup::getEventByIndex. If you do call EventGroup::loadEventData this won't happen. Calling the getevent function without loading the data first will mean a stall occurs as it loads. This is usually undesirable.

- EventSystem::getGroup / EventGroup::getGroup / EventGroup::getGroupByIndex allocate the **event instance memory** (including any DSP effect allocations) for the events in that group only, EVENT_CACHEEVENTS flag is used. This function is not recursive and does **NOT** traverse into subgroups.
- If **cacheevents = false** when getting a group, then FMOD will simply allocate the memory for the **event** when you try to get it. This means EventGroup::getEvent / EventGroup::getEventByIndex does the allocation.
- EventGroup::freeEventData unloads any wave data **AND** frees event instance memory for that group and all

groups below it. This function is recursive and **DOES** traverse into subgroups.

- EventGroup::getEvent / EventGroup::getEventByIndex will not do any disk access if EventGroup::loadEventData was called, or allocate any memory if you have precached it with a **GetGroup** function with **cacheevents = true** .

Just remember these things:
1. Load your data at the loading phase of the game, with EventGroup::loadEventData from the root of a tree, generally for **static banks**.
2. Be selective when it comes to events using **streaming banks** that are in groups. If you only wanted 1 stream to play out of 10 in a group (ie all the tracks in your music group), then don't call EventGroup::loadEventData on the group, just call EventGroup::getEvent / EventGroup::getEventByIndex without having called EventGroup::loadEventData on the group, or using **cacheevents = true** on a **GetGroup** function.
3. EventGroup::freeEventData frees all memory related to a group and its children, including **wave bank** data and the **event instance memory**. If you call this then it will have to re-load and reallocate the data if you try to use that group again.

FMOD event system CPU usage.

Now that the programmer has less control over the content of audio in the title, it may be easier to accidentally use more CPU than desired.
It is generally easy to find out FMOD cpu usage by using the low level functionality of FMOD with EventSystem::getSystemObject and System::getCPUUsage. If the sound designer is using too many DSP effects, then the 'dsp' value will be high.

By default FMOD Designer puts all sounds in hardware, unless there is a DSP effect applied to the event. If there is then FMOD will load it into main ram, and mix the event on the CPU instead of in the audio chip.

Note all FMOD DSP effects are gradually being optimized using whatever SIMD capabilities are available on the machine. If an effect seems slower than it should be, it may possibly not be optimized, and request through support@fmod.org to have it done.
There are 12 effects and 11 platforms to optimize, which means 132 routines have to be optimized, and they also have to be optimized for mono/stereo and multichannel purposes so that is 396 loops to write, so as you can see the most important effects and platforms will be targeted first as we make our way through them.

FMOD DESIGNER NETWORK API PROGRAMMER'S TUTORIAL

Introduction.

This section provides instructions on how to use the FMOD Designer Network API. By using the FMOD Designer Network API, it is possible to use the FMOD Designer tool to connect to your game as it is running and tweak the properties of events as they're playing. This is useful for sound designers as they can, for example, play the game and adjust volume levels of events in realtime as they hear them.

Using the NetEventSystem functions.

To use the FMOD Designer Network API all you need to do is :

- Call NetEventSystem_Init and pass it a pointer to your EventSystem object.
- Call NetEventSystem_Update just after each call you make to EventSystem::update.
- Call NetEventSystem_Shutdown after you call EventSystem::release.
- Link with fmod_event_net.lib as well as fmod_event.lib.

Connecting to your game using FMOD Designer.

When your game is up and running using the FMOD Designer Network API, you can use the FMOD Designer tool to connect to it at any time.
To connect to your game :

- Select "Audition -> Manage Connections..." from the menu.
- Add a new connection and fill in the relevant details. You can use the loopback address 127.0.0.1 if you want to run your game and FMOD Designer on the same machine.
- Click "Connect".
- Load a project containing some or all of the events that will play in the game. Note: FMOD Designer can't trigger new events within your game. If you want to hear the results of your tweaking, you must to trigger the relevant events from the game side.

- Now you can adjust the properties of event using the event property sheet in the event view, and envelopes / settings in event editor view.

- Save your changes in FMOD Designer using "File -> Save" at any time. Note: You must build the project again before these changes are made permanent in the .FEV file.

FMOD Ex and movie players

Introduction

This section describes how to have FMOD happily coexist with various movie playback systems available.

PlayStation 2.

This section describes how to have FMOD happily coexist with various movie playback systems available. The main causes on conflicts between other middleware that uses audio or the IOP, and FMOD are:

Conflict between the SPU2 DMA channels. There are 2 of these. DMA channel 0 and DMA channel 1. By default FMOD uses SPU2 DMA channel 0 for software mixing, and DMA core 1 for uploading sample data and for streaming to. This means DMA core 1 is used when System::createSound is being executed to load a PS2 FSB file, or streaming using System::createStream. To work around this issue see the following tips.

- **Turn off the FMOD software mixer.** This is already done if you are using the _reduced version of the library. This will free up DMA Channel 0. Most of the time you are not going to need the FMOD software mixer. You can do this by using System::init with the FMOD_INIT_DISABLESOFTWARE flag.
- **Swap FMOD's mixer/upload channel usage around.** If the 3rd party software still uses DMA Channel 1 (the channel FMOD uses for bank uploads and streaming), you can either change your 3rd party software to use DMA Channel 0 instead of DMA Channel 1, or tell FMOD to swap its usage around by specifying FMOD_INIT_PS2_SWAPDMACHANNELS. If you didn't turn the software mixer off, this would make FMOD use DMA Channel 0 for streams and sample bank uploads, and DMA Channel 1 for the software mixer.

Conflict on the SIFCMD ports. If your middleware or your own code is using the SIFCMD sony library to communicate with the IOP, then if you dont take care to share the SIFCMD buffers and ports with FMOD, messages will get lost and unexpected behaviour will occur in FMOD and your 3rd party software.

- **If you want to initialize your 3rd party software after FMOD.** Use this information if the code has a way to set up its SIFCMD usage. Note that FMOD uses SIFCMD port **0 and 1**, and has a **buffer size of 16**.
- **If you want to initialize your 3rd party software before FMOD.** If FMOD is initialized first, call System::init and use FMOD_PS2_EXTRADRIVERDATA structure from **fmodps2.h**. Also load your FMODEX.IRX or FMODEXD.IRX with command line parameters to allow the IOP side to get the same information. For more detailed information on this see the comment above the FMOD_PS2_EXTRADRIVERDATA declaration in **fmodps2.h**.

SPU2 ram usage and SPU2 hardware voice usage. Because FMOD and the middleware might not know about each other, they might allocate memory or use SPU2 voices without any regard for the other.

- **Use FMOD_SPU2_Alloc / FMOD_SPU2_GetRawAddress / FMOD_SPU2_Free.** Use these functions to set aside SPU2 ram for other middleware usage.
- **Use FMOD_SPU2_ReserveVoice.** Use this function to set aside an SPU2 voice for other middleware usage.

Xbox 1

The main causes on conflicts between other middleware that uses audio on the XBox, and FMOD are:

"DSOUND: CMcpxAPU::AllocateVoices: Error: Not enough free hardware voices". By default FMOD assumes it is in total control of the audio, so it allocates every XBox audio voice. If another 3rd party software application tries to allocate a hardware voice it will fail.

- Use System::setHardwareChannels. To get around this issue just call System::setHardwareChannels to reduce the count. XBox has around 192 HW2D voices so you could reduce this and still have plenty of voices free.
- **DSP Image incorrect.** FMOD has an internal MCP DSP image that it loads. It is stripped down to save memory, (hundreds of kilobytes) by removing unnescessary features. This may conflict with other 3rd party software that relies on a standard DSP image such as Microsoft's dsstdfx.bin.
- **Use FMOD_SpecifyEffectsImage.** This functionality is not available at this time. Contact support.
- **Make the other software use FMOD's image.** This functionality is not available at this time. Contact support.
- **Needs access to the XBox LPDIRECTSOUND handle.** If the 3rd party software is initialized second, it may want to use the XBox LPDIRECTSOUND handle.
- **Use System::getOutputHandle.** If you need to get a handle to FMOD's internal DirectSound pointer, you can share it by calling System::getOutputHandle and casting it to the appropriate pointer type.

Windows

If you need to get a handle to FMOD's internal DirectSound pointer, you can share it by calling System::getOutputHandle and casting it to the appropriate pointer type.
This also goes for XBox and other platforms.

Bink on PS2

Here is a quick way to get FMOD PS2 and Bink to co-exist.

Initialize FMOD first.

```
 FMOD_System_Init(system, numchannels, FMOD_INIT_PS2_SWAPDMACHANNELS, NULL);
```

Initialize Bink

```
MovieBuffer = sceSifAllocIopHeap( RAD_IOPMemoryAmount( RAD_IOP_NEED_CORE1 ) );
 RAD_IOPMemoryAmount(RAD_IOP_NEED_CORE1) ));
if (!RAD_IOPStartUp( 1, 2, MovieBuffer, RAD_IOP_NEED_CORE1|RAD_IOP_NO_INIT_LIBSD) )
{
sceSifFreeIopHeap(MovieBuffer);
// error
}
 RAD_IOPHardwareVolumes(1, 0x3fff, 0x3fff, 0x3fff, 0x3fff);
 BinkSoundUseRAD_IOP(1);
```

THREADS AND THREAD SAFETY

Introduction

This section will talk about the threads FMOD creates, and thread safety.

FMOD threads types.

FMOD has 4 main threads, and 2 of which are created at the time of System::init, and 2 of which are created only when you use certain flags.
They are

- **Mixer thread**. Software mixing thread, created at System::init.
- **Stream thread**. Thread used for decoding streams. Created the first time a sound is loaded as a stream in System::createSound with FMOD_CREATESTREAM. or System::createStream.
- **Async loading thread**. Created the first time a sound is loaded with the FMOD_NONBLOCKING flag in System::createSound.
- **File reading thread**. Thread used for reading from disk for streams, to then be decoded (decompressed) by the Stream thread. Created the first time a sound is loaded as a stream in System::createSound with FMOD_CREATESTREAM. or System::createStream.

Exceptions.

If FMOD_INIT_STREAM_FROM_UPDATE is used, then the stream thread will not be created.
If FMOD_OUTPUTTYPE_WAVWRITER_NRT or FMOD_OUTPUTTYPE_NOSOUND_NRT are used, then the mixer thread will not be created.

Everything else is run from the **main / game** thread, including System::update calculations.

FMOD threads priorities per platform.

For reference, for your own thread code, here are the thread priorities used in FMOD per platform.

Thread priority table.

Platform	Mixer thread	Stream thread	Async loading thread	File reading thread
Win32/Win64 (SetThreadPriority)	THREAD_PRIORITY_TIME_CRITICAL	THREAD_PRIORITY_HIGHEST	THREAD_PRIORITY_ABOVE_NORMAL	THREAD_PRIORITY_ABOVE_NORMAL
Linux/Linux64 (pthread_setschedparam)	99	94	90	90
Mac/Mac86 (MPSetTaskWeight)	10000	9000	8000	8000

PS2 (EE, CreateThread)	1	10	17	17
PSP (sceKernelCreateThread)	8	12	16	16
PS3 (sys_ppu_thread_create)	0	300	600	600
Xbox (SetThreadPriority)	THREAD_PRIORITY_TIME_CRITICAL	THREAD_PRIORITY_HIGHEST	THREAD_PRIORITY_ABOVE_NORMAL	THREAD_PRIORITY_ABOVE_NORMAL
Xbox 360 (HW Thread 4, SetThreadPriority)	THREAD_PRIORITY_TIME_CRITICAL	THREAD_PRIORITY_HIGHEST	THREAD_PRIORITY_ABOVE_NORMAL	THREAD_PRIORITY_ABOVE_NORMAL
Gamecube (OSCreateThread)	0	8	12	12
Wii (OSCreateThread)	0	8	12	12

FMOD callbacks

FMOD File and memory callbacks can possibly be called from an FMOD thread. Remember that if you specify file or memory callbacks with fmod, to make sure that they are thread safe. FMOD may call these callbacks from the stream thread, or FMOD_NONBLOCKING thread at any time.

Calling FMOD commands from different threads.

Do not call FMOD commands from different threads! This will lead to instability, corruption and possible crashes.

Some people are tempted to put System::update into a separate thread, **Only do this if you criticalsection this and all other calls to fmod.**.
To make FMOD thread safe would involve wrapping every FMOD function in a critical section, which adds unnecessary overhead, so at this time FMOD Ex is remaining 'not thread safe' for the time being.

REVERB NOTES

Introduction

This section will discuss FMOD's reverb parameters. For a more general description of reverb, see here.

The fields of FMOD_REVERB_PROPERTIES (found in 'fmod.h') control both hardware (via EAX) and software (via SFX) instances of reverb. EAX has a few parameters that the software doesn't use, so you can ignore these for the purpose of this discussion. For example, EnvSize is NOT used by the software reverb because it is only meaningful to EAX.

FMOD's software reverb DSP is controlled by parameters defined in the I3DL2 guidelines, which describe the reverberant environment of the listener.

Here's a list of the fields of FMOD_REVERB_PROPERTIES that currently have an effect and a description of what they do within the context of the software reverb. The descriptions are much the same as in 'fmod.h'.

Environment	Turns the reverb off if set to -1
Room	Room effect level (at mid frequencies)
RoomHF	Relative room effect level at high frequencies
DecayTime	Reverberation decay time at mid frequencies
DecayHFRatio	High-frequency to mid-frequency decay time ratio
Reflections	Early reflections level relative to room effect
Reverb	Late reverberation level relative to room effect
ReverbDelay	Late reverberation delay time relative to initial reflection
Diffusion	Echo density in the late reverberation decay
Density	Modal density in the late reverberation decay
HFReference	Reference high frequency (Hz) [see RoomHF]
RoomLF	Relative room effect level at low frequencies
LFReference	Reference low frequency (Hz) [see RoomLF]
Room	Room effect level (at mid frequencies)

Please note :

RoomRolloffFactor is a part of I3DL2, but has no effect within FMOD.

As with most reverberation models, the response is split into sections. This implementation has early reflections and late reverberation, each of which are composed of sets of delay lines having different delay and decay characteristics.

There are a few things to note here:

1) Room, RoomHF, RoomLF, Reflections and Reverb are all measured in milliBels, i.e. 100th of a deciBel, and they're all integers.
2) Room is the input gain
3) Reflections is a gain on the output of the early reflections subsystem
4) Reverb is a gain on the output of the late reverb subsystem

5) RoomLF and LFReference control a low frequency shelving filter on the input
6) RoomHF and HFReference control a high frequency shelving filter on the input
7) Diffusion and Density control the correlation among delay lines in the reverb subsystem

The FMOD_PRESET_* presets can be useful as examples of how these parameters change the nature of the reverb. They give quite a wide scope for representing different environments.

One more thing - it's important to distinguish between FMOD_REVERB_PROPERTIES and FMOD_REVERB_CHANNELPROPERTIES. The latter is just used for controlling the a channel's input gain to the reverb, and doesn't affect the characteristics of the reverb unit itself.

WINDOWS SPECIFIC ISSUES / FEATURES

Installation

- Use **api/fmodex.dll** to use FMOD Ex with all plugins statically compiled into the DLL. This means you can use all the features of FMOD without needing extra plugins accompanying your application. The DLL is bigger because of this.
- Use **api/fmodexp.dll** to use FMOD Ex with plugins external. This DLL needs plugins to function, which you can find in the plugins directory. Plugins in the plugins/ directory need to be used to support all of FMOD Ex's features. Use this if you want a smaller distribution and only need one file format support for example (ie .WAV). The DLL is smaller because of this.

Linking (which library to link to)

If you want to use fmodex.dll: (all plugins compiled into the dll, larger main dll size)

- Visual Studio users - **fmodex_vc.lib**.
- Metrowerks Codewarrior users - **fmodex_vc.lib**.
- Intel compiler users - **fmodex_vc.lib**.
- Borland users - **fmodex_bc.lib**.
- LCC-Win32 users - **fmodex_lcc.lib**.
- Dev-C++, MinGW and CygWin users - **libfmodex.a**.

If you want to use fmodexp.dll: (plugins left external, smaller main dll size).

- Visual Studio users - **fmodexp_vc.lib**.
- Metrowerks Codewarrior users - **fmodexp_vc.lib**.
- Intel compiler users - **fmodexp_vc.lib**.
- Borland users - **fmodexp_bc.lib**.
- LCC-Win32 users - **fmodexp_lcc.lib**.
- Dev-C++, MinGW and CygWin users - **libfmodexp.a**.

Recommended start up sequence (IMPORTANT!).

Due to configuration issues on Windows user's machines, this following code fixes the following issues:

- Speaker configuration in windows being ignored and just deafulting to stereo. (see use of System::getDriverCaps and 'controlpanelspeakermode' parameter, which is then passed to System::setSpeakerMode)
- Stuttering audio due to the user having their 'Hardware accelleration' slide set to 'off' in XP. (see check for FMOD_CAPS_HARDWARE_EMULATED, which then increases the FMOD DSP buffersize to over 200ms with System::setDSPBufferSize)
- Speaker configuration being set to a setting that the soundcard *doesn't actually support* (See check for

FMOD_ERR_OUTPUT_CREATEBUFFER, which then triggers a reinitialization with FMOD_SPEAKERMODE_STEREO)

!!! THIS CODE MUST BE USED FOR SHIPPING GAMES. DO NOT SHIP A GAME WITHOUT A STARTUP SEQUENCE BASED ON THIS CODE !!!

Use the following code as a basis for your Windows start up sequence.

```
     FMOD::System      *system;
     FMOD_RESULT        result;
    unsigned int        version;
     FMOD_SPEAKERMODE   speakermode;
     FMOD_CAPS          caps;

    /*
        Create a System object and initialize.
     */
     result = FMOD::System_Create(?
    ERRCHECK(result);

     result = system->getVersion(?
    ERRCHECK(result);

    if (version getDriverCaps(0, ?
    ERRCHECK(result);

     result = system->setSpeakerMode(speakermode);        /* Set the user selected speaker
mode. */
    ERRCHECK(result);

    if (caps ?    {                                            /* You might want
 to warn the user about this. */
        result = system->setDSPBufferSize(1024, 10);      /* At 48khz, the latency between
issuing an fmod command and hearing it will now be about 213ms. */
        ERRCHECK(result);
    }

     result = system->init(100, FMOD_INIT_NORMAL, 0);     /* Replace with whatever channel
count and flags you use! */
    if (result == FMOD_ERR_OUTPUT_CREATEBUFFER)           /* Ok, the speaker mode selected
isn't supported by this soundcard. Switch it back to stereo... */
    {
        result = system->setSpeakerMode(FMOD_SPEAKERMODE_STEREO);
        ERRCHECK(result);

        result = system->init(100, FMOD_INIT_NORMAL, 0); /* Replace with whatever
channel count and flags you use! */
        ERRCHECK(result);
    }
```

Important issue with Borland, LCC-Win32, Dev-C++, MinGW, Cygwin users and FMOD Ex C++ interface.

Note that due to incompatible linking standards with C++ symbols in libraries across different compilers, you will not be able to use the C++ interface of FMOD Ex with these compilers.

You can only use the FMOD Ex C interface, as at least that has a compatible standard (ie stdcall symbols are always the same format).
Each C++ compiler generates its own version of mangled symbols, and the mentioned compilers are not compatible with the symbols that MSVC produces, which is what FMOD is compiled in, and is the more popular compiler for commercial development at this stage.

Note that the Intel compiler and Codewarrior do not have this problem, they can resolve MSVC style symbols.

Troubleshooting.

Stuttering/skipping sound when using software mixed sounds, or streams.

More commonly known as buffer underrun/overrun, this can be 1 or a combination of factors:

- **Bad soundcard drivers** - This may be solved by upgrading your soundcard drivers. (Note it is recommended you get the latest drivers anyway)
- **CPU issues** - Machine to slow, or whatever your are trying to do with FMOD is too cpu intensive! (ie playing 100 mp3's at once will most likely bring FMOD to its knees, or maybe a user stream callback or DSP callback is spending too much time executing).
- **Mixer buffersize is set too small** - You can increase stability to combat these issues, by increasing FMOD's internal mixing buffer size. This will lead to greater stability but also larger latency on issuing commands to hearing the result. Call System::setDSPBufferSize to alter this. See documentation for System::setDSPBufferSize for more information.
- **Stream buffersize is set too small** - If you are using the FMOD Ex streamer, you might be streaming from a slow media, such as CDROM or over network, or even a fragmented harddisk, therefore FMOD needs more time to fill its streaming buffer before it runs out. See System::setStreamBufferSize to adjust the file read buffer size for the streamer. If the stream is starving because the codec is an expensive codec (and the file media is not to blame) then the problem could be the FMOD stream decode buffer size. You can adjust this using the 'decodebuffersize' member of the FMOD_CREATESOUNDEXINFO structure.
- **Output type** - FMOD_OUTPUTTYPE_DSOUND will provide more solid output than FMOD_OUTPUTTYPE_WINMM in anything except Windows NT. This is a problem with Windows Multimedia Services not being as realtime as it should be. Under NT FMOD_OUTPUTTYPE_WINMM is more stable, as DirectSound in NT is just emulated by using WINMM itself and is actually slower and has longer latency!. **Note!** Please don't feel the need to use System::setOutput if you don't need to. FMOD autodetects the best output mode based on the operating system.

LINUX SPECIFIC ISSUES / FEATURES

Installation

- Use **api/lib/libfmodex.so** to use FMOD Ex with all plugins statically compiled into the library. This means you can use all the features of FMOD without needing extra plugins accompanying your application. The library is bigger because of this.
- Use **api/lib/libfmodexp.so** to use FMOD Ex with plugins external. This library needs plugins to function, which you can find in the plugins directory. Plugins in the plugins/ directory need to be used to support all of FMOD Ex's features. Use this if you want a smaller distribution and only need one file format support for example (ie .WAV). The library is smaller because of this.

Formats not supported.

WMA is the only file format not support on FMOD Ex for linux. This is because FMOD uses a windows codec to be able to decode WMA. This codec is proprietory and owned by Microsoft and is not cross platform.

MAC SPECIFIC ISSUES / FEATURES

Installation

- Use **api/lib/libfmodex.dylib** to use FMOD Ex with all plugins statically compiled into the library. This means you can use all the features of FMOD without needing extra plugins accompanying your application. The library is bigger because of this.
- Use **api/lib/libfmodexp.dylib** to use FMOD Ex with plugins external. This library needs plugins to function, which you can find in the plugins directory. Plugins in the plugins/ directory need to be used to support all of FMOD Ex's features. Use this if you want a smaller distribution and only need one file format support for example (ie .WAV). The library is smaller because of this.

Universal Binaries

All FMOD Ex libraries for Macintosh are shipped as universal binaries, this means FMOD will support applications designed for either PowerPC, x86 or both. This causes the libraries to be around double the size compared with those that only support one variant. If you are only targeting one platform, either PowerPC or x86 you can extract smaller individual libraries from the provided larger one using the **"lipo"** tool.

ie.

```
 lipo -thin ppc  libfmodex.dylib -output libfmodex_ppc.dylib

or

 lipo -thin i386 libfmodex.dylib -output libfmodex_x86.dylib
```

PLAYSTATION 2 SPECIFIC ISSUES / FEATURES

Installation.

EE libraries.
Link this into your project. One of these files must be linked.

GCC / ProDG users.

- **/api/lib/fmodex.a** for general development with all possible features included. Software mixing features will incur higher CPU usage.
- **/api/lib/fmodexD.a** for the same library, but with debug logging which can help to determine any problems if they exist.
- **/api/lib/fmodex_reduced.a** for general development with a smaller library size and features removed. See table below for which features are removed.Recommended for PS2, as it lowest memory usage and cpu usage.
- **/api/lib/fmodexD_reduced.a** for the same library, but with debug logging which can help to determine any problems if they exist.

Codewarrior users.

- **/api/lib/fmodex_cw.lib** for general development with all possible features included. Software mixing features will incur higher CPU usage.
- **/api/lib/fmodexD_cw.lib** for the same library, but with debug logging which can help to determine any problems if they exist.
- **/api/lib/fmodex_reduced_cw.lib** for general development with a smaller library size and features removed. See table below for which features are removed.Recommended for PS2, as it lowest memory usage and cpu usage.
- **/api/lib/fmodexD_reduced_cw.lib** for the same library, but with debug logging which can help to determine any problems if they exist.

IOP module.
Load this into your project at runtime. You have to load the IRX yourself using sceSifLoadModule. More about this follows.

- **/api/fmodex.irx** for general development.
- **/api/fmodexD.irx** for the same IRX, but with debug logging which can help to determine any problems if they exist.

Feature table.

Feature	**fmodex.a**	**fmodex_reduced.a**	**Requires software mixing?**
Streaming audio support	Y	Y	N
3D Sound	Y	Y	N
Virtual voices	Y	Y	N
FMOD Designer API support	Y	Y	N
Nonblocking sound open support	Y	Y	N

Hardware reverb	Y	Y	N
Geometry support / polygon occlusion	Y	N	N
Software mixing	Y	N	Y
Spectrum Analysis	Y	N	Y
Network streaming	N	N	n/a
Recording support	N	N	n/a
File format - FSB	Y	Y	N
File format - VAG	Y	Y	N
File format - AIFF	Y	N	Y
File format - DLS	Y	N	Y
File format - FLAC	Y	N	Y
File format - IT (sequenced music format)	Y	N	Y
File format - MIDI (seqenced music format)	Y	N	Y
File format - MOD (sequenced music format)	Y	N	Y
File format - MP2 / MP3	Y	N	Y
File format - Ogg Vorbis	Y	N	Y
File format - M3U / PLS / ASX (Playlist format)	Y	N	Y
File format - RAW (format specified by user)	Y	N	Y
File format - S3M (sequenced music format)	Y	N	Y
File format - Tag formats - ID3V2, ASF, Ogg tags	Y	N	N
File format - XM (sequenced music format)	Y	N	Y
File format - WAV	Y	N	Y
File format - User created	Y	Y	N
File format - ASF / WMA	N	N	n/a
File format - CDDA	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_PS2	Y	Y	N
Output mode - FMOD_OUTPUTTYPE_WAVWRITER	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_WAVWRITER_NRT	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND_NRT	N	N	n/a
DSP Filter - Oscillator	Y	N	Y
DSP Effect - Lowpass	Y	N	Y
DSP Effect - Lowpass2	Y	N	Y
DSP Effect - Highpass	Y	N	Y
DSP Effect - Echo	Y	N	Y
DSP Effect - Flange	Y	N	Y
DSP Effect - Distortion	Y	N	Y
DSP Effect - Normalize	Y	N	Y
DSP Effect - Parameq	Y	N	Y
DSP Effect - Pitchshift	Y	N	Y
DSP Effect - Chorus	Y	N	Y
DSP Effect - Software reverb	Y	N	Y
DSP Effect - IT echo	Y	N	Y
DSP Effect - SFX reverb	N	N	n/a

In this table "**Requires software mixing?**" is specified to let the user know that the main CPU and RAM will be used to perform the feature which may not be desirable for the programmer.
Most things requiring the FMOD software mixer are removed in the reduced version of the library, to provide simple sound support.

The FMOD designer API can be used with the reduced library as well as long as all banks are marked as hardware in FMOD designer.

Note with a source code license you can easily turn features on and off to reduce code size or create different combinations of features to best suit your needs.

Loading IRX modules required for FMOD to operate.

Put **/api/fmodex.irx** into your modules directory. You have to load the IRX yourself using sceSifLoadModule.

A simple PlayStation 2 application has to do the following to use FMOD.

1.
Load **fmodex.irx** and Sony's **libsd.irx**. This is done from host0 or cdrom0 or whatever file device you store your files on.
Load libsd first, then fmod.

eg.

```
while(sceSifLoadModule(host0:modules/libsd.irx", 0, NULL)
```

Note that the default position for libsd.irx is at /usr/local/sce/iop/modules/ but it may differ on your machine.

What does each module do?

- libsd.irx - This is a sony irx that contains the low level hardware routines needed by fmodex.irx.
- fmodex.irx - This is the fmod library and contains the majority of the functionality.

2.

Remember to Initialize the IOP heap with sceSifInitIopHeap().

This is to be called after the call to sceSifInitRpc(0) and before loading any modules.

eg.

```
sceSifInitRpc(0);
sceSifInitIopHeap();
```

WARNING! If you reboot the IOP you have to call the above again!! Otherwise FMOD will fail to initialize.

3.
Note that mismatching fmodex.a and fmodexirx versions are not tolerated.
FMOD will fail to initialize if they are from different releases of FMOD.

Code warrior users - call mwInit()!

FMODEX is a C++ library and needs to have its global constructors called. Make sure you call mwInit at the start of your program.
If you don't do this you will get errors almost immediately from FMOD::System_Create.

SPU2 Sound ram.

The Playstation 2 has 2MB of sound ram which can store compressed VAG data (3.5:1 compression ADPCM variant), but you can not access all 2mb.
FMODEX uses most of the ram, but some is set aside for hardware work areas.

Input/Output Area (21k)	VAG Sound data (1515k)	CORE0 Reverb work area (256k)	CORE1 Reverb work area (256k)

The first 21kb is the hardware work area which can not be used to store sound data.

If you don't plan to use reverb, you can gain all 512k of the reverb work area back for sound data.
i.e.

Input/Output Area (21k)	VAG Sound data (2027k)

You can do this by specifying FMOD_INIT_PS2_DISABLECORE0REVERB | FMOD_INIT_PS2_DISABLECORE1REVERB in the flags region of System::init.

CORE0 reverb affects hardware voices 0 to 23, and CORE1 reverb affects hardware voices 24 to 47. Currently there is no way to make FMOD play a sound on a particular core but this will be added in a future edition.

EE Thread Priority.

Note that if the EE main thread priority is not changed from the default of 1, FMOD will change it to 32.
No threads can start when the default priority is 1 so this is necessary.
FMOD uses 1 thread to receive messages from the IOP.

Running FMODEX examples.

All examples refer to files that are relative to the elf. In target manager for example, tick the 'Set file serving root to elf' to make host0 the same directory as the elf.

The RBB format - The recommended format for samples and streams.

Although .WAV and .VAG formats are supported, for loading speed and streaming speed it is highly recommended to use .RBB files.
RBB is hardware accelerated, WAV is not.

FSB is compiled native PlayStation 2 SPU2 sound data, arranged so when loaded, it is one read to load the headers first, then the raw wave data (which is stored continuously), and when loaded it is efficiently streamed into SPU2 ram. This is the fastest way to load sound data.

Playstation 2 hardware reverb.

You have access to the hardware PlayStation 2 SPU2 reverb through FMOD's reverb API. Note that the SPU2 Reverb is a lot more primitive than I3DL2 reverb and EAX3.
In the FMOD_REVERB_PROPERTIES structure, only Environment, Room and Flags are supported.

'Environment'

This is a value between 0 and 9 mapping to the sony reverb modes.
You will find 9 special presets for the PlayStation 2 with this environment value set accordingly.
ie
FMOD_PRESET_PS2_ROOM
FMOD_PRESET_PS2_STUDIO_A
FMOD_PRESET_PS2_STUDIO_B
FMOD_PRESET_PS2_STUDIO_C
FMOD_PRESET_PS2_HALL
FMOD_PRESET_PS2_SPACE
FMOD_PRESET_PS2_ECHO
FMOD_PRESET_PS2_DELAY
FMOD_PRESET_PS2_PIPE
The other presets will not work, except for FMOD_PRESET_OFF.

'Room'

This still controls the amount of reverb mixed into the output.
Normally it is in decibels, between -10000 (silent) and 0 (full volume), and it is the same range on the PlayStation 2, but it is a linear scale between -10000 and 0, not a logarithmic one.

'Flags'

This only utilizes the following fields on PlayStation 2.
FMOD_REVERB_FLAGS_CORE0 (hardware voices 0 to 23)
FMOD_REVERB_FLAGS_CORE1 (hardware voices 24 to 47)
This tells the FMOD Engine which core, or set of hardware voices to apply the reverb settings to.
By default (in the presets) it is set to apply to both cores, but you can remove these flags to control each core separately.

Note that Channel::setReverbProperties is supported through the 'Room' parameter only, and that this value is binary, ie -10000 is 'reverb off' for the channel, and anything else is 'reverb on'.

PLAYSTATION 3 SPECIFIC ISSUES / FEATURES

Installation.

FMOD libraries were built using PS3 SDK 210.001.

PPU libraries.

Link this into your project. One of these files must be linked.

- Use **/api/lib/fmodex.a** for general development with all possible features included.
- Use **/api/lib/fmodexL.a** for the same library, but with debug logging which can help to determine any problems if they exist.

The following libraries should also be linked into your project:

- libm.a
- libfs_stub.a
- libspurs_stub.a
- libsysutil_stub.a
- libaudio_stub.a

SPU Threads and SPU Threads priorities.

When using SPU Threads, the SPU version of FMOD requires the SPU modules:

- **/api/fmodex_spu.self**
- **/api/fmodex_spu_mpeg.self**

The path to the SPU modules must be passed in through the **FMOD_PS3_EXTRADRIVERDATA** structure declared in **fmodps3.h**.

If you with to load the SPU modules from memory, pass in pointers rather than the file paths and set the **spu_load_from_memory** member to 1.

The SPU priority of the mixer, mpeg stream decoder and at3 decoder is also specified in this structure. For more information on the **FMOD_PS3_EXTRADRIVERDATA**, please refer to the **fmodps3.h** header.

A pointer to this structure should be passed in as **extradriverdata** to System::init.

ie.

```
      FMOD_PS3_EXTRADRIVERDATA extradriverdata;

     memset(?

     extradriverdata.spu_mixer_elfname_or_spursdata       =
SYS_APP_HOME "/fmodex_spu.self";
```

```
    extradriverdata.spu_streamer_elfname_or_spursdata   =
SYS_APP_HOME "/fmodex_spu_mpeg.self";

    extradriverdata.spu_load_from_memory                = 0;   /* Set this to 1 if we
have passed in pointers to embedded elf data */

    extradriverdata.spu_priority_mixer                  = 16;  /* Default */
    extradriverdata.spu_priority_at3                    = 200; /* Default */
    extradriverdata.spu_priority_streamer               = 200; /* Default */

    extradriverdata.spurs  = 0;   /* Set this to NULL when using SPU Threads */

    extradriverdata.force5point1     = 0;   /* Deprecated. Set to 0 */
    extradriverdata.attenuateDDLE    = 0;   /* Deprecated. Set to 0 */

    result = system->init(32, FMOD_INIT_NORMAL, (void *)?
    ERRCHECK(result);
```

SPURS

When using FMOD with SPURS, load the SPURS modules:

- **/api/fmodex_spurs.elf**
- **/api/fmodex_spurs_mpeg.elf**

into 128 byte aligned memory and pass a pointer to this memory through the **spu_mixer_elfname_or_spursdata** and **spu_streamer_elfname_or_spursdata** members of the **FMOD_PS3_EXTRADRIVERDATA** structure.

You must also pass a pointer to your SPURS instance through **spurs** member of **FMOD_PS3_EXTRADRIVERDATA**.

```
    FMOD_PS3_EXTRADRIVERDATA extradriverdata;
    CellSpurs                *spurs;
    void                     *fmodex_spurs_bin;
    void                     *fmodex_spurs_mpeg_bin;

    cellSysmoduleLoadModule(CELL_SYSMODULE_SPURS);

    spurs = (CellSpurs *)memalign(CELL_SPURS_ALIGN, sizeof(CellSpurs));

    cellSpursInitialize(spurs, 2, 250, ppu_thr_prio, false);

    extradriverdata.spu_mixer_elfname_or_spursdata      =
_binary_fmodex_spurs_elf_start;       /* Pointer to SPURS data */
    extradriverdata.spu_streamer_elfname_or_spursdata   =
_binary_fmodex_spurs_mpeg_elf_start;  /* Pointer to SPURS data */

    memset(?

    extradriverdata.spu_mixer_elfname_or_spursdata      = fmodex_spurs;
    extradriverdata.spu_streamer_elfname_or_spursdata   = fmodex_spurs_mpeg_bin;

    extradriverdata.spu_load_from_memory                = 0;   /* THIS WILL BE IGNORED
*/

    extradriverdata.spu_priority_mixer                  = 16;  /* Default      THIS WILL
BE IGNORED */
    extradriverdata.spu_priority_streamer               = 200; /* Default      THIS WILL
BE IGNORED */
    extradriverdata.spu_priority_at3                    = 200; /* Default */

    extradriverdata.spurs   = spurs;
    extradriverdata.force5point1      = 0;
    extradriverdata.attenuateDDLE     = 0;

    result = system->init(32, FMOD_INIT_NORMAL, (void *)&extradriverdata);
    ERRCHECK(result);
```

For an example on using SPURS, please refer to the PlayStream example provided with the FMOD SDK.

Loading Sounds into RSX Memory.

It is possible to load sounds into RSX memory using the FMOD_LOADSECONDARYRAM flag with System::createSound.

In order to do this, you must pass a pointer to a pool of RSX memory through the **rsx_pool** member in the **FMOD_PS3_EXTRADRIVERDATA** structure as well as the size of the RSX memory pool via **rsx_pool_size** member.

To get the usage of RSX memory pool, use the System::getSoundRAM function.

ie.

```
    FMOD_PS3_EXTRADRIVERDATA extradriverdata;

    void *rsxmemory;
    int   id;
    int   rsx_pool_size = 32 * 1024 * 1024;

    /*
        Some RSX memory
    */
    glGenBuffers(1, &id);
    glBindBuffer(GL_ARRAY_BUFFER, id);
    glBufferData(GL_ARRAY_BUFFER, rsx_pool_size, NULL, GL_DYNAMIC_DRAW);
    rsxmemory = glMapBuffer(GL_ARRAY_BUFFER, GL_READ_WRITE);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    memset(&extradriverdata, 0, sizeof(FMOD_PS3_EXTRADRIVERDATA));

    extradriverdata.spu_mixer_elfname_or_spursdata       =
SYS_APP_HOME"/fmodex_spu.self";
    extradriverdata.spu_streamer_elfname_or_spursdata    =
SYS_APP_HOME"/fmodex_spu_mpeg.self";

    extradriverdata.spu_load_from_memory                 = 0;

    extradriverdata.spu_priority_mixer                   = 16;   /* Default */
    extradriverdata.spu_priority_at3                     = 200;  /* Default */
    extradriverdata.spu_priority_streamer                = 200;  /* Default */

    extradriverdata.spurs  = 0;    /* Set this to NULL when using SPU Threads */

    extradriverdata.rsx_pool          = rsxmemory;              /* Pointer to RSX memory
 pool */
    extradriverdata.rsx_pool_size     = rsx_pool_size;        /* Size of RSX memory pool
 */

    extradriverdata.force5point1      = 0;    /* Deprecated. Set to 0 */
    extradriverdata.attenuateDDLE     = 0;    /* Deprecated. Set to 0 */

    result = system->init(32, FMOD_INIT_NORMAL, (void *)&extradriverdata);
    ERRCHECK(result);
    /*
        Load sound into RSX memory
     */
     result = system->createSound(SYS_APP_HOME "/drumloop.mp3",
FMOD_CREATECOMPRESSEDSAMPLE | FMOD_LOADSECONDARYRAM, 0, ?
    ERRCHECK(result);

    /*
        Get RSX Memory usage
     */
    {
        int current, max, total;

        system->getSoundRam(?
    }
```

Note: There is a performance hit if using RSX memory for audio samples. There is about a 10% performance hit when playing mp3 samples, and about a 50% performance hit when playing PCM samples.

FMOD_ADVANCEDSETTINGS - maxPCMcodecs

The FMOD_ADVANCEDSETTINGS structure has a PS3 specific member, **maxPCMcodecs**. This sets the maximum number of PCM voices that can be played at once.

This includes streams of any format, as well as any sounds that are opened without the FMOD_CREATECOMPRESSEDSAMPLE flag, or sound formats not compatible with the FMOD_CREATECOMPRESSEDSAMPLE flag (compatible formats are mp2/mp3/adpcm). If not set, this defaults to 16.

Typically, all your sound effects would be in FMOD_CREATECOMPRESSEDSAMPLE and **maxPCMcodecs** would be set to the maximum number of streams you anticipate being played at once.

Custom DSP Units

Custom DSP units are currently not supported on the PS3.

AT3 Playback (NOT RECOMMENDED!)

Note: AT3 playback is now disabled and no longer supported.

AT3 playback makes use of Sony's own atrac library which is very inefficient. A new SPURS instance is created for each AT3 stream!! (this means 1 SPU per AT3 stream!). In fact, AT3 support has actually been abandoned by Sony. We highly discourage the use of this format and would recommended mp3/mp2/adpcm/pcm for the PS3.

TRC: requirements for using the SPU selfs (fmodex_spu.self / fmodex_spu_mpeg.self)

TRC 25.5 requires SPU programs that are not loaded from disk to be embedded, as raw .elf into the PPU program.

You can unsign the .self files back to .elf files using the "unfself.exe" utility included in the Sony SDK, before embedding them into your program.

Set the **spu_load_from_memory** member to 1 and point to the address of the embedded elfs using the **spu_mixer_elfname_or_spursdata** and **spu_streamer_elfname_or_spursdata** members of the **FMOD_PS3_EXTRADRIVERDATA** structure.

If you are just simply loading the SPU selfs from disk as per the above examples, you can just leave them as they are.

TRC: requirements for using the SPURS elfs (fmodex_spurs.elf / fmodex_spurs_mpeg.elf)

TRC 17.1.1 requires executable data on the Blu-Ray disc to be only in SELF/SPRX format.

If you are using SPURS you will need to embed the SPURS elf files into your application. The SPURS data can then be pointed to using the **spu_mixer_elfname_or_spursdata** and **spu_streamer_elfname_or_spursdata** members of the **FMOD_PS3_EXTRADRIVERDATA** structure.

The PlayStream example included in the FMOD SDK demonstrates how to load an embedded SPURS elf from the application.

For more information about this, please refer to the "Application Requirements" section of the Sony SDK documentation.

TRC: Audio Formats Supported

The "Audio formats supported at boot" are as follows:

- 7.1ch LPCM
- 5.1ch LPCM, Downmix from 7.1ch LPCM
- 5.1ch Dolby Interactive Encoding
- 2ch LPCM, Downmix from 7.1ch LPCM

Sounds only coming out of left/right speakers?

A common issue is that users incorrectly re-initialize the ps3 audio for movie players with settings that don't match what FMOD did. This causes problems with audio not coming out of all the speakers. You should not re-initialize the audio. The member **cell_audio_config** has been provided in **FMOD_PS3_EXTRADRIVERDATA** which gives information as to how FMOD has initialized PS3 audio. This information can be used for movie players etc.

7.1 surround speakers

On Playstation 3 in 7.1, the extra 2 speakers are not side left/side right, they are 'surround back left'/'surround back right' which locate the speakers behind the listener instead of to the sides like on PC. FMOD_SPEAKER_SBL/FMOD_SPEAKER_SBR are provided to make it clearer what speaker is being addressed on that platform.

Running FMOD Examples.

All examples refer to files in "SYS_APP_HOME". "SYS_APP_HOME" should be set to the SDK media directory, **/examples/media**. In target manager for example, set the "Fileserving root directory" to **/examples/media** .

What to do if audio cuts out or you get garbled sound.

There are two issues that usually cause this:

- If you have a thread that doesn't yield enough, a number of processes can die, including libAudio. This is usually observed during the loading of level data.

For details, please refer to https://ps3.scedev.net/technotes/view/302.

- If you are passing bad floats to FMOD. To check if this is the case, you should link to the logging version of FMOD (libfmodexL.a) and check to see if any functions are returning FMOD_ERR_INVALID_FLOAT.

Analog Audio Output Connections.

If using the analog audio output terminals of the DEH-R1040+ devkits, make sure "AV MULTI" is selected as the audio output in the bootup OSD. Otherwise the audio may sound "tinny".

Note: The analog audio output terminals are mis-labelled on some devkits, for details please refer to https://ps3.scedev.net/technotes/view/160.

PLAYSTATION PORTABLE SPECIFIC ISSUES / FEATURES

Installation.

You will need to use at least version 2.8.0 of the PSP SDK.

Libraries.
Link this into your project. One of these files must be linked.

GCC users.

- **/api/lib/fmodex.a** for general development with all possible features included. Software mixing features will incur higher CPU usage.
- **/api/lib/fmodexD.a** for the same library, but with debug logging which can help to determine any problems if they exist.
- **/api/lib/fmodex_reduced.a** for general development with a smaller library size and features removed. See table below for which features are removed. Recommended for PSP, as it lowest memory usage and cpu usage.
- **/api/lib/fmodexD_reduced.a** for the same library, but with debug logging which can help to determine any problems if they exist.

SNC compiler users.

- **/api/lib/fmodex_SNC.lib** for general development with all possible features included. Software mixing features will incur higher CPU usage.
- **/api/lib/fmodexD_SNC.lib** for the same library, but with debug logging which can help to determine any problems if they exist.
- **/api/lib/fmodex_reduced_SNC.lib** for general development with a smaller library size and features removed. See table below for which features are removed. Recommended for PSP, as it lowest memory usage and cpu usage.
- **/api/lib/fmodexD_reduced_SNC.lib** for the same library, but with debug logging which can help to determine any problems if they exist.

Feature table.

Feature	**fmodex.a**	**fmodex_reduced.a**	**Requires software mixing?**
Streaming audio support	Y	Y	N
3D Sound	Y	Y	N
Virtual voices	Y	Y	N
FMOD Designer API support	Y	Y	N
Nonblocking sound open support	Y	Y	N
Hardware reverb	Y	Y	N
Geometry support / polygon occlusion	Y	N	N
Software mixing	Y	N	Y
Spectrum Analysis	Y	N	Y

Network streaming	N	N	n/a
Recording support	N	N	n/a
File format - FSB	Y	Y	N
File format - VAG	Y	Y	N
File format - AT3	Y	Y	N
File format - AIFF	N	N	Y
File format - DLS	Y	N	Y
File format - FLAC	N	N	Y
File format - IT (sequenced music format)	Y	N	Y
File format - MIDI (seqenced music format)	Y	N	Y
File format - MOD (sequenced music format)	Y	N	Y
File format - MP3	Y	Y	N
File format - Ogg Vorbis	N	N	Y
File format - M3U / PLS / ASX (Playlist format)	Y	N	Y
File format - RAW (format specified by user)	Y	N	Y
File format - S3M (sequenced music format)	Y	N	Y
File format - Tag formats - ID3V2, ASF, Ogg tags	N	N	N
File format - XM (sequenced music format)	Y	N	Y
File format - WAV	Y	N	Y
File format - User created	Y	Y	N
File format - ASF / WMA	N	N	n/a
File format - CDDA	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_PSP	Y	Y	N
Output mode - FMOD_OUTPUTTYPE_WAVWRITER	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_WAVWRITER_NRT	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND_NRT	N	N	n/a
DSP Filter - Oscillator	Y	N	Y
DSP Effect - Lowpass	Y	N	Y
DSP Effect - Lowpass2	Y	N	Y
DSP Effect - Highpass	Y	N	Y
DSP Effect - Echo	Y	N	Y
DSP Effect - Flange	Y	N	Y
DSP Effect - Distortion	Y	N	Y
DSP Effect - Normalize	Y	N	Y
DSP Effect - Parameq	Y	N	Y
DSP Effect - Pitchshift	Y	N	Y
DSP Effect - Chorus	Y	N	Y
DSP Effect - Software reverb	Y	N	Y
DSP Effect - IT echo	Y	N	Y
DSP Effect - SFX reverb	N	N	n/a

In this table "**Requires software mixing?**" is specified to let the user know that the main CPU and RAM will be used to perform the feature which may not be desirable for the programmer.
Most things requiring the FMOD software mixer are removed in the reduced version of the library, to provide simple sound support.
The FMOD designer API can be used with the reduced library as well as long as all banks are marked as hardware in FMOD designer.

Note with a source code license you can easily turn features on and off to reduce code size or create different combinations of features to best suit your needs.

Loading modules and linking libraries required for FMOD to operate.

FMOD requires the following sony libraries and modules to be used as well to get audio support. They are linked and loaded by the user.

- Link weak import stub library libsas_weak.a into your project.
- Link weak import stub library libatrac3plus_stub_weak.a into your project.
- Link weak import stub library libmp3_stub_weak.a into your project.
- Load module SCE_UTILITY_AV_MODULE_SASCORE at runtime.

If you require AT3 playback support, load the following sony run-time modules:

- Load module SCE_UTILITY_AV_MODULE_LIBATRAC3PLUS
- Load module SCE_UTILITY_AV_MODULE_AVCODEC

If you require MP3 playback support, load the following sony run-time modules:

- Load module libmp3.prx
- Load module SCE_UTILITY_AV_MODULE_AVCODEC

Loading the modules.

Load modules using the **sceUtilityLoadAvModule** / **sceUtilityUnloadAvModule** PSP SDK functions.

ie.

```
    if (sceUtilityLoadAvModule(SCE_UTILITY_AV_MODULE_SASCORE)
```

Load the libmp3.prx module using the **sceKernelLoadModule** and **sceKernelStartModule** PSP SDK functions.

ie.

```
    #define MODULE_NAME "host0:/cygwin/usr/local/psp/devkit/module/libmp3.prx"

    SceUID Modid;
    SceInt32 r;

    SceUID id = sceKernelLoadModule(MODULE_NAME, 0, NULL);
    if (id
```

The FSB format - The recommended format for samples.

Although .WAV and .OGG formats are supported, for loading speed it is highly recommended to use .VAG files.

VAG is hardware accelerated, WAV is not.

These are compiled batches of native PlayStation Portable sound data, arranged so when loaded, it is only ready to load the headers first, then the raw vag data (which is continuous), which is streamed into RAM. This is the fastest way to load sound data from UMD.

Streaming ATRAC music.

FMOD supports the playback of .at3 files using the PSP hardware decoder (PSP Media Engine).

By default, the maximum number of at3 decoder instances is 2, however it is possible to increase this using the PSP SDK function **sceAtracReinit(int at3_entry, int at3plus_entry)**. The at3_entry parameter being the number of ATRAC3s to be decoded simultaneously and the at3plus_entry being the number of ATRAC3 plus' to be decoded simultaneously. The relationship between these two parameters is:

at3_entry + at3plus_entry * 2 <= 6

Please refer to the PSP SDK documentation for more details.

Streaming MP3 music.

FMOD supports the playback of .mp3 files using the PSP hardware decoder (PSP Media Engine). Only 44100 Hz mp3s are supported by the decoder.

The maximum number of mp3 decoder instances is 2, therefore it is not possible to play more than 2 mp3 streams at the same time.

Battery considerations.

Note, that even though FMOD supports streaming multiple streams from UMD at once, this is not recommended. On the PlayStation Portable seeking should be avoided at all times to preserve movement of the umd read head and therefore battery life. This also goes for data streaming. Do not stream data and music at the same time if there is seeking involved. Continuous seeking will degrade battery life because it has to mechanically move the seek head.

It may be preferable to play 'in memory' music such as sequenced formats like .MOD/.S3M/.XM or .IT or play music in AT3 format. As ATRAC can store a minute of audio per mb at 128kbs stereo, then you would need 3mb of memory for 3 minutes of music.

XBOX SPECIFIC ISSUES / FEATURES

Installation

- Use **api/lib/fmodxbox.lib** for general development.
- Use **api/lib/fmodxboxD.lib** for the same library, but with logging which can help to determine any problems if they exist.

On FMOD XBox, you **must** call FMOD::Memory_Initialize, and supply a pool of memory with a length.

for example.

```
#define AUDIO_MEMLENGTH (4*1024*1024)

char *mem = malloc(AUDIO_MEMLENGTH);

FMOD::Memory_Initialize(mem, AUDIO_MEMLENGTH, NULL, NULL, NULL);
```

then call

System::init**.**

The reasoning for this is for performance.
FMOD must be able to access sample data within its own memory block to avoid a slowdown issue in the DirectSound implementation on XBox.
IDirectSoundBuffer8::SetBufferData must create a table or an 'SGE list' every time this function is called, and it is slow.
Some games might call this function every time a sound was played, causing significant CPU degradation.
FMOD uses just 1 call to this function (at initialization time), and from then on uses the more efficient method of simply making Directsound XBox bufferdata pointers point to the whole FMOD memory block, and then from then on, it simply specifies an offset within that buffer for each hardware audio sound.

The memory provided must be enough to store all samples and extra system memory overhead for FMOD.

You can call FMOD_Memory_GetStats to determine what FMOD needs as a game runs.
You could run FMOD and supply it with an unrealistically high memory pool (say 8 megabytes), and then call FMOD_Memory_GetStats to determine the maximum amount of RAM fmod needs to store sounds and for FMOD system overhead.

The 8mb Memory Limitation.

Currently for hardware sound effects, there is an 8mb limit for sound effects.
This is due to the XBox DirectSound architecture.

From the XDK Documentation : *"DirectSound buffers are managed in a scatter gather entry (SGE) list. There is a maximum of 2,047 SGEs, which each point to a 4-KB page. This means that a maximum of 8 MB are available for allocating or playing DirectSound buffers simultaneously"*

Future versions may have multiple 8mb pools if it is required by developers but generally this is more than enough for audio.
Remember that data stored within this memory can be XADPCM format which is about 3.5:1 compression.

WMA support.

WMA is supported but generally as a streaming format. The benefits of WMA vs Ogg vorbis for example are negligible, so you could use whatever audio format you like for music streaming.

Formats not supported.

Currently all advertised FMOD formats are supported. This may change as some formats are not used generally (ie .FLAC) and just take up code space that they don't need to.
Note! With a source code license you can remove and add whatever formats you like, which will reduce the size of the library significantly. For example you can easily remove all formats except for XMA if so desired.

5.1 support and speaker settings.

Note that the Xbox dashboard is the only place the speaker settings are selected. This is done by the user and should not be changed or forced by the code in any way, as it will go against the user's selection.
FMOD will automatically use the correct speaker setting that was selected in the dashboard.

XBOX 360 SPECIFIC ISSUES / FEATURES

Installation

- Use **api/lib/fmodxbox360.lib** for general development.
- Use **api/lib/fmodxbox360D.lib** for the same library, but with debug logging which can help to determine any problems if they exist.

FMOD also uses some XDK libraries. You must link with the following.
- **xmp.lib** for XMPGetStatus to determine if the dashboard is playing its own music or not.
- **xaudio.lib** for FMOD sound output support.

Besides this there are no other requirements. It is optional for you to give fmod a block of memory to work within if you like (ie using FMOD::Memory_Initialize), otherwise you can let FMOD simply use the default memory allocators provided by the Xbox 360 operating system.

Memory

This is an important subject, as performance can be bad or even worse, FMOD will not function correctly.

XMA support needs buffers allocated with XPhysicalAlloc. malloc and free will not allow XMA to function correctly.

By default FMOD uses XPhysicalAlloc to allocate memory, but the page size for XPhysicalAlloc is 4096 bytes. As FMOD can do a lot of smaller allocations, this is grossly inefficient and wastes memory.
The solution is to use your own memory manager, or let FMOD manage the memory with FMOD::Memory_Initialize
If you have an efficient, low page size memory manager, that can use memory allocated with XPhysicalAlloc, use the memory callback feature of FMOD::Memory_Initialize.

If not, the easier solution is to allocate just 1 large block of memory with XPhysicalAlloc, then simply give it to FMOD via FMOD::Memory_Initialize (leave callbacks set to NULL). With this you can simply pass the buffer and size, and FMOD will manage the memory internally and not allocate outside of this.

FMOD's memory manager has a 32byte page size. If your memory manager has pages bigger than this, it would be more efficient to use fmod's memory manager.

Multiple CPUs

As the Xbox 360 has multiple cpus, you can specify which cpu and which hardware thread FMOD Ex's threads can operate on.

Before doing this though, note that FMOD has already selected **Thread 4** (CORE2, HW Thread 0) to process its software mixer thread, stream thread, FMOD_NONBLOCKING loader thread, and file thread.

We chose this CPU and thread as this is the same CPU and thread that XAudio runs on by default.
You will not necessarily have to change it, because it won't affect the main game code which is assumed to be running on **Thread 0**.

If you want to change FMOD's core and hardware thread assignments, just use the structure found in **fmodxbox360.h**, which is in api/inc.
This structure is then passed in as the 'extradriverdata' parameter of System::init.

Dashboard music technical requirement.

Xbox 360 TCR states that you must allow the user to select their own music from the dashboard, which should then in turn make the built in game music go quiet or pause.

To do this is simple with FMOD Ex. The mechanism used is for the user to create a special channel group (see System::createChannelGroup) with the name "**music**".
When you play a music track, set the channel group for the channel to this music group, and FMOD will automatically pause it if the user selects a song from the dashboard to play.

If using the "FMOD Designer" sound designer tool, put all music events under the "**master/music**" category (or one of its sub-categories).

XMA support.

To load a .XMA file as a static compressed sample, the FMOD_CREATECOMPRESSEDSAMPLE or FMOD_CREATECOMPRESSEDMEMSAMPLE must be used when loading the file.

Formats not supported.

Currently only .FLAC has been removed, as it is generally an uncommon format to be needed on a console such as the Xbox 360, and its inclusion adds too much to the library size.
Note! With a source code license you can remove and add whatever formats you like, which will reduce the size of the library significantly. For example you can easily remove all formats except for XMA if so desired.

5.1 support and speaker settings.

Note that the Xbox dashboard is the only place the speaker settings are selected. This is done by the user and should not be changed or forced by the code in any way, as it will go against the user's selection.
FMOD will automatically use the correct speaker setting that was selected in the dashboard.

The programmer is to assume everything is in 5.1 internally, and not be concerned about the end users' setup.

GAMECUBE SPECIFIC ISSUES / FEATURES

Installation

Linking FMOD Ex to your application.

- **/api/lib/fmodgc.lib** - Link to this file if you are using **SN Systems** compiler.
- **/api/lib/fmodgcD.lib** - This is the debug version of fmodgc.lib and outputs a log of FMOD's progress and any error messages (in plain english) to the TTY.
- **/api/lib/fmodgc_cw.a** - Link to this file if you are using the **Metrowerks Codewarrior** compiler.
- **/api/lib/fmodgc_cwD.a** - This is the debug version of fmodgc_cw.lib and outputs a log of FMOD's progress and any error messages (in plain english) to the TTY.

Running the examples.

Simply load their .dsp files into Dev Studio and hit F7.

NOTE: You will need to copy all files in the media directory to
$DVDROOT/fmod. For example:

```
copy media\*.* c:\DolphinSDK1.0\dvddata\fmod
```

Formats not supported.

WMA is not support on FMOD Ex for Gamecube. This is because FMOD uses a windows codec to be able to decode WMA. This codec is proprietory and owned by Microsoft and is not cross platform.
FLAC is not supported on FMOD Ex for GameCube. The FLAC codec is rarely used and just takes up unescessary code space. If it is need then contact us at support@fmod.org.

Wii SPECIFIC ISSUES / FEATURES

Installation.

Libraries.
Link this into your project. One of these files must be linked.

- **/api/lib/fmodwii.a** for general development with all possible features included.
- **/api/lib/fmodwiiL.a** for the same library, but with debug logging which can help to determine any problems if they exist.
- **/api/lib/fmodwii_reduced.a** for general development with a smaller library size and features removed. See table below for which features are removed.
- **/api/lib/fmodwiiL_reduced.a** for the same library, but with debug logging which can help to determine any problems if they exist.

The following libraries should also be linked into your project:

- base.a
- os.a
- exi.a
- si.a
- db.a
- vi.a
- gx.a
- dvd.a
- ai.a
- ax.a
- axfx.a
- mix.a
- dsp.a
- ipc.a
- fs.a
- pad.a
- wpad.a
- wenc.a
- wud.a
- euart.a
- sc.a
- nand.a

Feature table.

Feature	fmodwii.a	fmodwii_reduced.a	Requires software mixing?
Streaming audio support	Y	Y	N

3D Sound	Y	Y	N
Virtual voices	Y	Y	N
FMOD Designer API support	Y	Y	N
Nonblocking sound open support	Y	Y	N
Hardware reverb	Y	Y	N
Geometry support / polygon occlusion	Y	N	N
Software mixing	Y	N	Y
Spectrum Analysis	Y	N	Y
Network streaming	N	N	n/a
Recording support	N	N	n/a
File format - FSB	Y	Y	N
File format - DSP	Y	Y	N
File format - AIFF	Y	N	Y
File format - DLS	Y	N	Y
File format - FLAC	N	N	Y
File format - IT (sequenced music format)	Y	N	Y
File format - MIDI (seqenced music format)	Y	N	Y
File format - MOD (sequenced music format)	Y	N	Y
File format - MP2 / MP3	Y	N	Y
File format - Ogg Vorbis	Y	N	Y
File format - M3U / PLS / ASX (Playlist format)	Y	N	Y
File format - RAW (format specified by user)	Y	N	Y
File format - S3M (sequenced music format)	Y	N	Y
File format - Tag formats - ID3V2, ASF, Ogg tags	Y	N	Y
File format - XM (sequenced music format)	Y	N	Y
File format - WAV	Y	N	Y
File format - User created	Y	Y	N
File format - ASF / WMA	N	N	n/a
File format - CDDA	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_WII	Y	Y	N
Output mode - FMOD_OUTPUTTYPE_WAVWRITER	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_WAVWRITER_NRT	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND	N	N	n/a
Output mode - FMOD_OUTPUTTYPE_NOSOUND_NRT	N	N	n/a
DSP Filter - Oscillator	Y	N	Y
DSP Effect - Lowpass	Y	N	Y
DSP Effect - Lowpass2	Y	N	Y
DSP Effect - Highpass	Y	N	Y
DSP Effect - Echo	Y	N	Y
DSP Effect - Flange	Y	N	Y
DSP Effect - Distortion	Y	N	Y
DSP Effect - Normalize	Y	N	Y
DSP Effect - Parameq	Y	N	Y
DSP Effect - Pitchshift	Y	N	Y
DSP Effect - Chorus	Y	N	Y
DSP Effect - Software reverb	Y	N	Y
DSP Effect - IT echo	Y	N	Y

In this table "**Requires software mixing?**" is specified to let the user know that the main CPU and RAM will be used to perform the feature which may not be desirable for the programmer.
Most things requiring the FMOD software mixer are removed in the reduced version of the library, to provide simple sound support.
The FMOD designer API can be used with the reduced library as well as long as all banks are marked as hardware in FMOD designer.

Note with a source code license you can easily turn features on and off to reduce code size or create different combinations of features to best suit your needs.

DVD Read Priority

It is possible to set the DVD read priority using the **dvdreadpriority** member in the **FMOD_WII_INFO** structure. A pointer to this structure is passed to System::init as the **extradriverdata** parameter. The setting can be in the range 0 - 3 with 0 being the highest and 3 the lowest. The default setting is 1.

Wii Controller Speaker Support

FMOD supports the playing of sounds out of the Wii controller speaker using the Wii specific API functions found in **fmodwii.h**.

The **FMOD_Wii_Controller_Command** function can be used to turn the Wii controller speaker on and off using the **FMOD_WII_CONTROLLER_CMD_SPEAKERON** and **FMOD_WII_CONTROLLER_CMD_SPEAKEROFF** commands.

Use the **FMOD_Channel_SetControllerSpeaker** function to set a channel to play out of the controller speaker. The last argument of the function, of type **FMOD_WII_CONTROLLER**, specifies which controller the channel is to be played out of. These can be or'd together to play a single channel out of multiple controllers. You can set this argument to **FMOD_WII_CONTROLLER_NONE** on a channel currently playing out of a controller speaker to just play out of regular speakers again.

Example of usage:

```
    /*
        Turn on the controller speaker. This should be done in your WPAD connect
callback, or
        when you know the controller is connected.
    */
    FMOD_Wii_Controller_Command(chan, FMOD_WII_CONTROLLER_CMD_SPEAKERON);

 .
 .
 .

    /*
        Play a sound with channel paused
    */
    result = system->playSound(FMOD_CHANNEL_FREE, sound1, true, ?
    ERRCHECK(result);

    /*
        Set channel to play out of FMOD_WII_CONTROLLER_0
    */
    result = FMOD_Channel_SetControllerSpeaker((FMOD_CHANNEL *)channel,
FMOD_WII_CONTROLLER_0);
    ERRCHECK(result);
    /*
        Unpause channel
    */
    result = channel->setPaused(false);
    ERRCHECK(result);
```

The **controllerspeaker** example demonstrates the usage of the Wii controller speaker.

It is possible to have FMOD mute/unmute the controller speaker depending on whether a sound is playing. This may help to reduce the battery life of the controller. This feature can be turned on by setting the **muteremotespeakerifnosound** member in the **FMOD_WII_INFO** structure to 1.

Using MEM2 for FMOD.

By default, all memory allocations in FMOD will use OSAlloc which will allocate from the current heap. If you wish to use MEM2, you may allocate a block of memory in MEM2 and pass it to FMOD to use.
ie.

```
    void                *arenaMem2Lo;
    void                *arenaMem2Hi;
    MEMHeapHandle        gExpHeap;

    /*
        Allocate a block of memory for FMOD to use in MEM2 (optional)
    */
    arenaMem2Lo = OSGetMEM2ArenaLo();
    arenaMem2Hi = OSGetMEM2ArenaHi();
    gExpHeap    = MEMCreateExpHeap(arenaMem2Lo, (u32)arenaMem2Hi - (u32) arenaMem2Lo);

    memblock = MEMAllocFromExpHeap(gExpHeap, 16 * 1024 * 1024);

    FMOD::Memory_Initialize(memblock, 16*1024*1024, NULL, NULL, NULL);
```

Handling disk eject.

When playing a stream and the disk is ejected, it will keep re-trying the read automatically until the disk is re-inserted. The **starving** parameter in function Sound::getOpenState will let you know when this is the case and you can mute the stream accordingly to prevent stutters.

If are callilng functions that read from disk in the main thread, or using the FMOD_NONBLOCKING flag, you will get the FMOD_ERR_FILE_DISKEJECTED error returned.

Firelight Technologies FMOD Ex

API Reference

Firelight Technologies FMOD Ex

C++ Reference

Interfaces

System Interface

System::getSoftwareChannels
System::getSoftwareFormat
System::getSoundRAM
System::getSpeakerMode
System::getSpectrum
System::getStreamBufferSize
System::getUserData
System::getVersion
System::getWaveData
System::init
System::isRecording
System::loadGeometry
System::loadPlugin
System::lockDSP
System::playDSP
System::playSound
System::recordStart
System::recordStop
System::release
System::set3DListenerAttributes
System::set3DNumListeners
System::set3DRolloffCallback
System::set3DSettings
System::set3DSpeakerPosition
System::setAdvancedSettings
System::setCallback
System::setDSPBufferSize
System::setDriver
System::setFileSystem
System::setGeometrySettings
System::setHardwareChannels
System::setNetworkProxy
System::setNetworkTimeout
System::setOutput
System::setOutputByPlugin
System::setPluginPath
System::setRecordDriver
System::setReverbAmbientProperties
System::setReverbProperties
System::setSoftwareChannels
System::setSoftwareFormat
System::setSpeakerMode
System::setStreamBufferSize
System::setUserData
System::unloadPlugin
System::unlockDSP
System::update

System::addDSP

This function adds a pre-created DSP unit or effect to the head of the System DSP chain.?

Syntax

```
  FMOD_RESULT System::addDSP(
    FMOD::DSP *  dsp
);
```

Parameters

dsp

A pointer to a pre-created DSP unit to be inserted at the head of the System DSP chain.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is a wrapper function to insert a DSP unit at the top of the System DSP chain.
It disconnects the head unit from its input, then inserts the unit at the head and reconnects the previously disconnected input back as as an input to the new unit.
It is effectively the following code.

```
int numinputs;
system->getDSPHead(?
 dsphead->getNumInputs(?
if (numinputs > 1)
{
       return FMOD_ERR_DSP_TOOMANYCONNECTIONS;

}
 dsphead->getInput(0, ?
 dsphead->disconnectFrom(next);
 dsphead->addInput(dsp);
 dsp->addInput(next);
 dsp->setActive(true);
```

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDSPHead
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- Channel::addDSP
- ChannelGroup::addDSP
- DSP::remove

System::attachFileSystem

Function to allow a user to 'piggyback' on FMOD's file reading routines. This allows users to capture data as FMOD reads it, which may be useful for ripping the raw data that FMOD reads for hard to support sources (for example internet streams or cdda streams).?

Syntax

```
  FMOD_RESULT System::attachFileSystem(
    FMOD_FILE_OPENCALLBACK  useropen,
    FMOD_FILE_CLOSECALLBACK  userclose ,
    FMOD_FILE_READCALLBACK  userread,
    FMOD_FILE_SEEKCALLBACK  userseek
);
```

Parameters

useropen

Pointer to an open callback which is called after a file is opened by FMOD.

userclose

Pointer to a close callback which is called after a file is closed by FMOD.

userread

Pointer to a read callback which is called after a file is read by FMOD.

userseek

Pointer to a seek callback which is called after a file is seeked into by FMOD.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setFileSystem
- FMOD_FILE_OPENCALLBACK
- FMOD_FILE_CLOSECALLBACK

- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK

System::close

Closes the system object without freeing the object's memory, so the system handle will still be valid.
?Closing the output renders objects created with this system object invalid. Make sure any sounds, channelgroups, geometry and dsp objects are released before closing the system object.
?

Syntax

```
FMOD_RESULT System::close();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- System::release

System::createChannelGroup

Creates a channel group object. These objects can be used to assign channels to for group channel settings, such as volume.
?Channel groups are also used for sub-mixing. Any channels that are assigned to a channel group get submixed into that channel group's DSP.?

Syntax

```
  FMOD_RESULT System::createChannelGroup(
  const char *  name,
   FMOD::ChannelGroup **  channelgroup
);
```

Parameters

name

Label to give to the channel group for identification purposes. Optional (can be null).

channelgroup

Address of a variable to receive a newly created FMOD::ChannelGroup object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See the channel group class definition for the types of operations that can be performed on 'groups' of channels.
The channel group can for example be used to have 2 seperate groups of master volume, instead of one global master volume.
A channel group can be used for sub-mixing, ie so that a set of channels can be mixed into a channel group, then can have effects applied to it without affecting other channels.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getMasterChannelGroup
- Channel::setChannelGroup

- ChannelGroup::release

System::createCodec

Creates an in memory file format codec to be used by FMOD by passing in a codec description structure.
?Once this is created, FMOD will use it to open user defined file formats.?

Syntax

```
  FMOD_RESULT System::createCodec(
   FMOD_CODEC_DESCRIPTION *  description
);
```

Parameters

description

Address of a FMOD_CODEC_DESCRIPTION structure, containing information about the codec.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_CODEC_DESCRIPTION

System::createDSP

Creates a user defined DSP unit object to be inserted into a DSP network, for the purposes of sound filtering or sound generation.?

Syntax

```
  FMOD_RESULT System::createDSP(
   FMOD_DSP_DESCRIPTION *  description,
   FMOD::DSP **  dsp
);
```

Parameters

description

Address of an FMOD_DSP_DESCRIPTION structure containing information about the unit to be created.

dsp

Address of a variable to receive a newly created FMOD::DSP object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A DSP unit can generate or filter incoming data.
The data is created or filtered through use of the read callback that is defined by the user.
See the definition for the FMOD_DSP_DESCRIPTION structure to find out what each member means.
To be active, a unit must be inserted into the FMOD DSP network to be heard. Use functions such as System::addDSP, Channel::addDSP or DSP::addInput to do this.
For more information and a detailed description (with diagrams) see the tutorial on the DSP system in the documentation.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_DESCRIPTION
- System::createDSPByType

- System::createDSPByIndex
- System::addDSP
- Channel::addDSP
- DSP::addInput
- DSP::setActive

System::createDSPByIndex

Creates a DSP unit object which is either built in or loaded as a plugin, to be inserted into a DSP network, for the purposes of sound filtering or sound generation.
?This function creates a DSP unit that can be enumerated by using System::getNumPlugins and System::getPluginInfo
.?

Syntax

```
  FMOD_RESULT System::createDSPByIndex(
  int index,
   FMOD::DSP ** dsp
);
```

Parameters

index

The index into the list of enumerable DSP plugins to create.

dsp

Address of a variable to receive a newly created FMOD::DSP object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A DSP unit can generate or filter incoming data.
To be active, a unit must be inserted into the FMOD DSP network to be heard. Use functions such as System::addDSP, Channel::addDSP or DSP::addInput to do this.
For more information and a detailed description (with diagrams) see the tutorial on the DSP system in the documentation.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins
- System::getPluginInfo

- System::createDSPByType
- System::createDSP
- System::addDSP
- Channel::addDSP
- DSP::addInput
- DSP::setActive

System::createDSPByType

Creates an FMOD defined built in DSP unit object to be inserted into a DSP network, for the purposes of sound filtering or sound generation.
?This function is used to create special effects that come built into FMOD.?

Syntax

```
  FMOD_RESULT System::createDSPByType(
    FMOD_DSP_TYPE  type,
    FMOD::DSP **  dsp
);
```

Parameters

type

A pre-defined DSP effect or sound generator described by a FMOD_DSP_TYPE.

dsp

Address of a variable to receive a newly created FMOD::DSP object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A DSP unit can generate or filter incoming data.
To be active, a unit must be inserted into the FMOD DSP network to be heard. Use functions such as System::addDSP, Channel::addDSP , ChannelGroup::addDSP or DSP::addInput to do this.
For more information and a detailed description (with diagrams) see the tutorial on the DSP system in the documentation.

Note! Winamp DSP and VST plugins will only return the first plugin of this type that was loaded!
To access all VST or Winamp DSP plugins the System::createDSPByIndex function! Use the index returned by System::loadPlugin if you don't want to enumerate them all.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_TYPE
- System::createDSP
- System::createDSPByIndex
- System::addDSP
- System::loadPlugin
- Channel::addDSP
- ChannelGroup::addDSP
- DSP::addInput
- DSP::setActive

System::createGeometry

Geometry creation function. This function will create a base geometry object which can then have polygons added to it.?

Syntax

```
FMOD_RESULT System::createGeometry(
  int  maxpolygons,
  int  maxvertices,
  FMOD::Geometry **  geometry
);
```

Parameters

maxpolygons

Maximum number of polygons within this object.

maxvertices

Maximum number of vertices within this object.

geometry

Address of a variable to receive a newly created FMOD::Geometry object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Polygons can be added to a geometry object using Geometry::AddPolygon.

A geometry object stores its list of polygons in a structure optimized for quick line intersection testing and efficient insertion and updating.
The structure works best with regularly shaped polygons with minimal overlap.
Many overlapping polygons, or clusters of long thin polygons may not be handled efficiently.
Axis aligned polygons are handled most efficiently.

The same type of structure is used to optimize line intersection testing with multiple geometry objects.

It is important to set the value of maxworldsize to an appropriate value using System::setGeometrySettings.
Objects or polygons outside the range of maxworldsize will not be handled efficiently.
Conversely, if maxworldsize is excessively large, the structure may lose precision and efficiency may drop.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setGeometrySettings
- System::loadGeometry
- Geometry::AddPolygon

System::createReverb

Creates a 'virtual reverb' object. This object reacts to 3d location and morphs the reverb environment based on how close it is to the reverb object's center.
?Multiple reverb objects can be created to achieve a multi-reverb environment.?

Syntax

```
FMOD_RESULT System::createReverb(
  FMOD::Reverb ** reverb
);
```

Parameters

reverb

Address of a pointer to a Reverb object to receive the newly created virtual reverb object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Use System::setReverbAmbientProperties to set a 'background' default reverb environment. This is a reverb that will be morphed to if the listener is not within any virtual reverb zones.
By default the ambient reverb is set to 'off'.
Creating multiple reverb objects does not impact performance. These are 'virtual reverbs'. There will still be only 1 physical reverb DSP running that just morphs between the different virtual reverbs.
System::setReverbProperties can still be used in conjunction with the 3d based virtual reverb system. This allows 2d sounds to have reverb. If this call is used at the same time virtual reverb objects are active, 2 physical reverb dsps will be used, incurring a small memory and cpu hit.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::release
- System::setReverbAmbientProperties
- System::getReverbAmbientProperties
- System::setReverbProperties
- System::getReverbProperties

System::createSound

Loads a sound into memory, or opens it for streaming.?

Syntax

```
  FMOD_RESULT System::createSound(
  const char *  name_or_data,
   FMOD_MODE  mode,
   FMOD_CREATESOUNDEXINFO  * exinfo,
   FMOD::Sound **  sound
);
```

Parameters

name_or_data

Name of the file or URL to open, or pointer to memory block if FMOD_OPENMEMORY/ FMOD_OPENMEMORY_POINT is used. For CD playback the name should be a drive letter with a colon, example "D:" (windows only).

mode

Behaviour modifier for opening the sound. See FMOD_MODE. Also see remarks for more.

exinfo

Pointer to a FMOD_CREATESOUNDEXINFO which lets the user provide extended information while playing the sound. Optional. Specify 0 or NULL to ignore.

sound

Address of a variable to receive a newly created FMOD::Sound object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Important! By default (FMOD_CREATESAMPLE) FMOD will try to load and decompress the whole sound into memory! Use FMOD_CREATESTREAM to open it as a stream and have it play back in realtime! FMOD_CREATECOMPRESSEDSAMPLE can also be used for certain formats.

- To open a file or URL as a stream, so that it decompresses / reads at runtime, instead of loading / decompressing into memory all at the time of this call, use the FMOD_CREATESTREAM flag. This is like a 'stream' in FMOD 3.
- To open a file or URL as a compressed sound effect that is not streamed and is not decompressed into memory

at load time, use FMOD_CREATECOMPRESSEDSAMPLE. This is supported with MPEG (mp2/mp3), ADPCM (wav on all platforms and vag on ps2/psp) and XMA files only. This is useful for those who want realtime compressed soundeffects, but not the overhead of disk access.
- To open a CD drive, use the drive as the name, for example on the windows platform, use "D:"
- To open a sound as 2D, so that it is not affected by 3D processing, use the FMOD_2D flag. 3D sound commands will be ignored on these types of sounds.
- To open a sound as 3D, so that it is treated as a 3D sound, use the FMOD_3D flag. Calls to Channel::setPan will be ignored on these types of sounds.
- To use FMOD software mixing buffers, use the FMOD_SOFTWARE flag. This gives certain benefits, such as DSP processing, spectrum analysis, loop points, 5.1 mix levels, 2d/3d morphing, and more.
- To use the soundcard's hardware to play the sound, use the FMOD_HARDWARE flag. This gives certain benefits such as EAX reverb, dolby digital output on some devices, and better 3d sound virtualization using headphones.

Note that FMOD_OPENRAW, FMOD_OPENMEMORY, FMOD_OPENMEMORY_POINT and FMOD_OPENUSER will not work here without the exinfo structure present, as more information is needed.

Use FMOD_NONBLOCKING to have the sound open or load in the background. You can use Sound::getOpenState to determine if it has finished loading / opening or not. While it is loading (not ready), sound functions are not accessable for that sound.

To account for slow devices or computers that might cause buffer underrun (skipping/stuttering/repeating blocks of audio), use System::setStreamBufferSize.
To play WMA files on Windows, the user must have the latest Windows media player codecs installed (Windows Media Player 9). The user can download this as an installer (wmfdist.exe) from www.fmod.org download page if they desire or you may wish to redistribute it with your application (this is allowed). This installer does NOT install windows media player, just the necessary WMA codecs needed.
PlayStation 2 Note: You can pre-pend "host0:" or "cdrom0:" if you like. FMOD will automatically add "host0:" to the filename if it is not found.
Specifying FMOD_OPENMEMORY_POINT will POINT to your memory rather allocating its own sound buffers and duplicating it internally
This means you cannot free the memory while FMOD is using it, until after Sound::release is called. With FMOD_OPENMEMORY_POINT, for PCM formats, only WAV, FSB and RAW are supported. For compressed formats, only those formats supported by FMOD_CREATECOMPRESSEDSAMPLE are supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- FMOD_CREATESOUNDEXINFO
- Sound::getOpenState
- System::setStreamBufferSize
- Channel::setPan

System::createSoundGroup

Creates a sound group, which can store handles to multiple Sound pointers.?

Syntax

```
  FMOD_RESULT System::createSoundGroup(
  const char *  name,
   FMOD::SoundGroup **  soundgroup
);
```

Parameters

name

Name of sound group.

soundgroup

Address of a variable to recieve a pointer to a sound group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Once a SoundGroup is created, Sound::setSoundGroup is used to put a sound in a SoundGroup.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::release
- Sound::setSoundGroup

System::createStream

Opens a sound for streaming. This function is a helper function that is the same as System::createSound but has the FMOD_CREATESTREAM flag added internally.?

Syntax

```
  FMOD_RESULT System::createStream(
  const char * name_or_data,
   FMOD_MODE mode,
   FMOD_CREATESOUNDEXINFO * exinfo,
   FMOD::Sound ** sound
);
```

Parameters

name_or_data

Name of the file or URL to open. For CD playback this may be a drive letter with a colon, example "D:".

mode

Behaviour modifier for opening the sound. See FMOD_MODE. Also see remarks for more.

exinfo

Pointer to a FMOD_CREATESOUNDEXINFO which lets the user provide extended information while playing the sound. Optional. Specify 0 or NULL to ignore.

sound

Address of a variable to receive a newly created FMOD::Sound object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that a stream only has 1 decode buffer and file handle, and therefore can only be played once. It cannot play multiple times at once because it cannot share a stream buffer if the stream is playing at different positions.
Open multiple streams to have them play concurrently.

- To open a file or URL as a stream, so that it decompresses / reads at runtime, instead of loading / decompressing into memory all at the time of this call, use the FMOD_CREATESTREAM flag. This is like a 'stream' in FMOD 3.
- To open a file or URL as a compressed sound effect that is not streamed and is not decompressed into memory at load time, use FMOD_CREATECOMPRESSEDSAMPLE. This is supported with MPEG (mp2/mp3),

ADPCM (wav on all platforms and vag on ps2/psp) and XMA files only. This is useful for those who want realtime compressed soundeffects, but not the overhead of disk access.

- To open a CD drive, use the drive as the name, for example on the windows platform, use "D:"
- To open a sound as 2D, so that it is not affected by 3D processing, use the FMOD_2D flag. 3D sound commands will be ignored on these types of sounds.
- To open a sound as 3D, so that it is treated as a 3D sound, use the FMOD_3D flag. Calls to Channel::setPan will be ignored on these types of sounds.
- To use FMOD software mixing buffers, use the FMOD_SOFTWARE flag. This gives certain benefits, such as DSP processing, spectrum analysis, loop points, 5.1 mix levels, 2d/3d morphing, and more.
- To use the soundcard's hardware to play the sound, use the FMOD_HARDWARE flag. This gives certain benefits such as EAX reverb, dolby digital output on some devices, and better 3d sound virtualization using headphones.

Note that FMOD_OPENRAW, FMOD_OPENMEMORY, FMOD_OPENMEMORY_POINT and FMOD_OPENUSER will not work here without the exinfo structure present, as more information is needed.

Use FMOD_NONBLOCKING to have the sound open or load in the background. You can use Sound::getOpenState to determine if it has finished loading / opening or not. While it is loading (not ready), sound functions are not accessable for that sound.

To account for slow devices or computers that might cause buffer underrun (skipping/stuttering/repeating blocks of audio), use System::setStreamBufferSize.
Note that FMOD_CREATESAMPLE will be ignored, overriden by this function because this is simply a wrapper to System::createSound that provides the FMOD_CREATESTREAM flag. The FMOD_CREATESTREAM flag overrides FMOD_CREATESAMPLE.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- FMOD_CREATESOUNDEXINFO
- Sound::getOpenState
- System::setStreamBufferSize
- System::createSound
- Channel::setPan

System::get3DListenerAttributes

This retrieves the position, velocity and orientation of the specified 3D sound listener.?

Syntax

```
FMOD_RESULT System::get3DListenerAttributes(
  int listener,
   FMOD_VECTOR *  pos,
   FMOD_VECTOR *  vel,
   FMOD_VECTOR *  forward,
   FMOD_VECTOR * up
);
```

Parameters

listener

Listener ID in a multi-listener environment. Specify 0 if there is only 1 listener.

pos

Address of a variable that receives the position of the listener in world space, measured in distance units. Optional. Specify 0 or NULL to ignore.

vel

Address of a variable that receives the velocity of the listener measured in distance units **per second**. Optional. Specify 0 or NULL to ignore.

forward

Address of a variable that receives the forwards orientation of the listener. Optional. Specify 0 or NULL to ignore.

up

Address of a variable that receives the upwards orientation of the listener. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DListenerAttributes
- FMOD_VECTOR

System::get3DNumListeners

Retrieves the number of 3D listeners.?

Syntax

```
  FMOD_RESULT System::get3DNumListeners(
  int *  numlisteners
);
```

Parameters

numlisteners

Address of a variable that receives the current number of 3D listeners in the 3D scene.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DNumListeners

System::get3DSettings

Retrieves the global doppler scale, distance factor and rolloff scale for all 3D sound in FMOD.?

Syntax

```
FMOD_RESULT System::get3DSettings(
  float *  dopplerscale,
  float *  distancefactor,
  float *  rolloffscale
);
```

Parameters

dopplerscale

Address of a variable that receives the scaling factor for doppler shift. Optional. Specify 0 or NULL to ignore.

distancefactor

Address of a variable that receives the relative distance factor to FMOD's units. Optional. Specify 0 or NULL to ignore.

rolloffscale

Address of a variable that receives the scaling factor for 3D sound rolloff or attenuation. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DSettings

System::get3DSpeakerPosition

Retrieves the current speaker position information for the selected speaker.?

Syntax

```
  FMOD_RESULT System::get3DSpeakerPosition(
    FMOD_SPEAKER speaker,
    float * x,
    float * y,
    bool * active
);
```

Parameters

speaker

The selected speaker of interest to return the x and y position.

x

Address of a variable that receives the 2D X position relative to the listener. Optional. Specify 0 or NULL to ignore.

y

Address of a variable that receives the 2D Y position relative to the listener. Optional. Specify 0 or NULL to ignore.

active

Address of a variable that receives the active state of a speaker.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See the System::set3DSpeakerPosition for more information on speaker positioning.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DSpeakerPosition
- FMOD_SPEAKERMODE
- FMOD_SPEAKER

System::getAdvancedSettings

Retrieves the advanced settings value set for the system object.?

Syntax

```
  FMOD_RESULT System::getAdvancedSettings(
    FMOD_ADVANCEDSETTINGS * settings
);
```

Parameters

settings

Address of a variable to receive the contents of the FMOD_ADVANCEDSETTINGS structure specified by the user.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_ADVANCEDSETTINGS
- System::setAdvancedSettings

System::getCDROMDriveName

Gets information on the selected cdrom drive.?

Syntax

```
  FMOD_RESULT System::getCDROMDriveName(
  int  drive,
  char *  drivename,
  int  drivenamelen,
  char * scsiname,
  int scsinamelen,
  char *  devicename,
  int  devicenamelen
);
```

Parameters

drive

The enumerated number of the CDROM drive to query. 0 based.

drivename

Address of a variable that receives the name of the drive letter or name depending on the operating system.

drivenamelen

Length in bytes of the target buffer to receieve the string.

scsiname

Address of a variable that receives the SCSI address of the drive. This could also be used to pass to System::createSound, or just used for information purposes.

scsinamelen

Length in bytes of the target buffer to receieve the string.

devicename

Address of a variable that receives the name of the physical device. This is usually a string defined by the manufacturer. It also contains the drive's vendor ID, product ID and version number.

devicenamelen

Length in bytes of the target buffer to receieve the string.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Enumerate CDROM drives by finding out how many there are with System::getNumCDROMDrives.

Platforms Supported

Win32, Win64, PlayStation 3

See Also

- System::getNumCDROMDrives
- System::createSound

System::getCPUUsage

Retrieves in percent of CPU time - the amount of cpu usage that FMOD is taking for streaming/mixing and System::update combined.?

Syntax

```
FMOD_RESULT System::getCPUUsage(
  float * dsp,
  float * stream,
  float * update,
  float * total
);
```

Parameters

dsp

Address of a variable that receives the current dsp mixing engine cpu usage. Result will be from 0 to 100.0f. Optional. Specify 0 or NULL to ignore.

stream

Address of a variable that receives the current streaming engine cpu usage. Result will be from 0 to 100.0f. Optional. Specify 0 or NULL to ignore.

update

Address of a variable that receives the current System::update cpu usage. Result will be from 0 to 100.0f. Optional. Specify 0 or NULL to ignore.

total

Address of a variable that receives the current total cpu usage. Result will be from 0 to 100.0f. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This value is slightly smoothed to provide more stable readout (and to round off spikes that occur due to multitasking/operating system issues).

NOTE! On ps3 and xbox360, the dsp and stream figures are NOT main cpu/main thread usage. On PS3 this is the percentage of SPU being used. On Xbox 360 it is the percentage of a hardware thread being used which is on a totally different CPU than the main one.

Do not be alarmed if the usage for these platforms reaches over 50%, this is normal and should be ignored if you are playing a lot of compressed sounds and are using effects. The only value on the main cpu / main thread to take note of here that will impact your framerate is the update value, and this is typically very low (ie less than 1%).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::update

System::getChannel

Retrieves a handle to a channel by ID.?

Syntax

```
  FMOD_RESULT System::getChannel(
  int channelid,
   FMOD::Channel ** channel
);
```

Parameters

channelid

Index in the FMOD channel pool. Specify a channel number from 0 to the 'maxchannels' value specified in System::init minus 1.

channel

Address of a variable that receives a pointer to the requested channel.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is mainly for getting handles to existing (playing) channels and setting their attributes.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::init

System::getChannelsPlaying

Retrieves the number of currently playing channels.?

Syntax

```
  FMOD_RESULT System::getChannelsPlaying(
  int * channels
);
```

Parameters

channels

Address of a variable that receives the number of currently playing channels.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

System::getDSPBufferSize

Retrieves the buffer size settings for the FMOD software mixing engine.?

Syntax

```
  FMOD_RESULT System::getDSPBufferSize(
  unsigned int *  bufferlength,
  int *  numbuffers
);
```

Parameters

bufferlength

Address of a variable that receives the mixer engine block size in samples. Default = 1024. (milliseconds = 1024 at 48khz = 1024 / 48000 * 1000 = 10.66ms). This means the mixer updates every 21.3ms. Optional. Specify 0 or NULL to ignore.

numbuffers

Address of a variable that receives the mixer engine number of buffers used. Default = 4. To get the total buffersize multiply the bufferlength by the numbuffers value. By default this would be 4*1024. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See documentation on System::setDSPBufferSize for more information about these values.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setDSPBufferSize

System::getDSPHead

Returns a pointer to the head DSP unit of the DSP network. This unit is the closest unit to the soundcard and all sound data comes through this unit.?

Syntax

```
  FMOD_RESULT System::getDSPHead(
    FMOD::DSP **  dsp
);
```

Parameters

dsp

Address of a variable that receives the pointer to the head DSP unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this unit if you wish to connect custom DSP units to the output or filter the global mix by inserting filter units between this one and the incoming channel mixer unit.
Read the tutorial on DSP if you wish to know more about this. It is not recommended using this if you do not understand how the FMOD Ex DSP network is connected.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getDSPHead
- ChannelGroup::getDSPHead

System::getDriver

Returns the currently selected driver number. Drivers are enumerated when selecting a driver with System::setDriver or other driver related functions such as System::getNumDrivers or System::getDriverInfo?

Syntax

```
  FMOD_RESULT System::getDriver(
  int *  driver
);
```

Parameters

driver

Address of a variable that receives the currently selected driver ID. 0 = primary or main sound device as selected by the operating system settings.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setDriver
- System::getNumDrivers
- System::getDriverInfo

System::getDriverCaps

Returns information on capabilities of the current output mode for the selected sound device.?

Syntax

```
  FMOD_RESULT System::getDriverCaps(
  int id,
   FMOD_CAPS * caps,
  int * minfrequency,
  int * maxfrequency,
   FMOD_SPEAKERMODE * controlpanelspeakermode
);
```

Parameters

id

Enumerated driver ID. This must be in a valid range delimited by System::getNumDrivers.

caps

Address of a variable that receives the capabilities of the device. Optional. Specify 0 or NULL to ignore.

minfrequency

Address of a variable that receives the minimum frequency allowed with sounds created with FMOD_HARDWARE. If Channel::setFrequency is used FMOD will clamp the frequency to this minimum. Optional. Specify 0 or NULL to ignore.

maxfrequency

Address of a variable that receives the maximum frequency allowed with sounds created with FMOD_HARDWARE. If Channel::setFrequency is used FMOD will clamp the frequency to this maximum. Optional. Specify 0 or NULL to ignore.

controlpanelspeakermode

Address of a variable that receives the speaker mode set by the operating system control panel. Use this to pass to System::setSpeakerMode if you want to set up FMOD's software mixing engine to match. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_CAPS
- System::init
- System::close
- System::getNumDrivers
- System::getHardwareChannels
- System::setSpeakerMode
- Channel::setFrequency

System::getDriverInfo

Retrieves identification information about a sound device specified by its index, and specific to the output mode set with System::setOutput.?

Syntax

```
FMOD_RESULT System::getDriverInfo(
  int id,
  char * name,
  int namelen,
   FMOD_GUID * guid
);
```

Parameters

id

Index of the sound driver device. The total number of devices can be found with System::getNumDrivers.

name

Address of a variable that receives the name of the device. Optional. Specify 0 or NULL to ignore.

namelen

Length in bytes of the target buffer to receieve the string. Required if name parameter is not NULL.

guid

Address of a variable that receives the GUID that uniquely identifies the device. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumDrivers
- System::setOutput

System::getGeometrySettings

Retrieves the maximum world size for the geometry engine.?

Syntax

```
  FMOD_RESULT System::getGeometrySettings(
    float * maxworldsize
);
```

Parameters

maxworldsize

Pointer to a float to receieve the maximum world size.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setGeometrySettings

System::getHardwareChannels

Returns the number of available hardware mixed 2d and 3d channels.?

Syntax

```
  FMOD_RESULT System::getHardwareChannels(
  int *  num2d,
  int *  num3d,
  int *  total
);
```

Parameters

num2d

Address of a variable that receives the number of available hardware mixed 3d channels. Optional. Specify 0 or NULL to ignore.

num3d

Address of a variable that receives the number of available hardware mixed 2d channels. Optional. Specify 0 or NULL to ignore.

total

Address of a variable that receives the total number of available hardware mixed channels. Usually total = num3d + num2d, but on some platforms like PS2 and GameCube, 2D and 3D voices share the same channel pool so total, num2d and num3d will all be the same number. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If total doesn't equal num3d + num2d, it usually means the 2d and 3d hardware voices share the same actual hardware voice pool.
For example if it said 32 for each value, then if 30 3d voices were playing, then only 2 voices total would be available, for 2d or 3d playback. They are not separate.

NOTE: If this is called before System::init, you must call System::getDriverCaps first.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii, Solaris

See Also

- System::init
- System::getChannelsPlaying
- System::setHardwareChannels
- System::getDriverCaps

System::getMasterChannelGroup

Retrieves a handle to the internal master channel group. This is the default channel group that all channels play on. ?This channel group can be used to do things like set the master volume for all playing sounds. See the ChannelGroup API for more functionality.?

Syntax

```
  FMOD_RESULT System::getMasterChannelGroup(
    FMOD::ChannelGroup ** channelgroup
);
```

Parameters

channelgroup

Address of a variable that receives a pointer to the master System object channel group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createChannelGroup
- ChannelGroup::setVolume
- ChannelGroup::getVolume

System::getMasterSoundGroup

Retrieves the default sound group, where all sounds are placed when they are created.?

Syntax

```
  FMOD_RESULT System::getMasterSoundGroup(
    FMOD::SoundGroup ** soundgroup
);
```

Parameters

soundgroup

Address of a pointer to a SoundGroup object to receive the master sound group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a user based soundgroup is deleted/released, the sounds will be put back into this sound group.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::release
- SoundGroup::getSystemObject
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- SoundGroup::getName
- SoundGroup::getNumSounds
- SoundGroup::getSound
- SoundGroup::getNumPlaying
- SoundGroup::setUserData
- SoundGroup::getUserData

System::getNetworkProxy

Retrieves the URL of the proxy server used in internet streaming.?

Syntax

```
  FMOD_RESULT System::getNetworkProxy(
  char *  proxy,
  int  proxylen
);
```

Parameters

proxy

Address of a variable that receives the proxy server URL.

proxylen

Size of the buffer in bytes to receive the string.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, PlayStation 3

See Also

- System::setNetworkProxy

System::getNetworkTimeout

Retrieve the timeout value for network streams?

Syntax

```
  FMOD_RESULT System::getNetworkTimeout(
  int *  timeout
);
```

Parameters

timeout

The timeout value in ms.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, PlayStation 3

System::getNumCDROMDrives

Retrieves the number of available CDROM drives on the user's machine.?

Syntax

```
  FMOD_RESULT System::getNumCDROMDrives(
  int * numdrives
);
```

Parameters

numdrives

Address of a variable that receives the number of CDROM drives.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getCDROMDriveName

System::getNumDrivers

Retrieves the number of soundcard devices on the machine, specific to the output mode set with System::setOutput.?

Syntax

```
  FMOD_RESULT System::getNumDrivers(
  int *  numdrivers
);
```

Parameters

numdrivers

Address of a variable that receives the number of output drivers.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If System::setOutput is not called it will return the number of drivers available for the default output type. Use this for enumerating sound devices. Use System::getDriverInfo to get the device's name.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDriver
- System::getDriverInfo
- System::setOutput
- System::getOutput

System::getNumPlugins

Retrieves the number of available plugins loaded into FMOD at the current time.?

Syntax

```
  FMOD_RESULT System::getNumPlugins(
   FMOD_PLUGINTYPE  plugintype,
  int *  numplugins
);
```

Parameters

plugintype

Specify the type of plugin type such as FMOD_PLUGINTYPE_OUTPUT, FMOD_PLUGINTYPE_CODEC or FMOD_PLUGINTYPE_DSP.

numplugins

Address of a variable that receives the number of available plugins for the selected type.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_PLUGINTYPE

System::getOutput

Retrieves the current output system FMOD is using to address the hardware.?

Syntax

```
  FMOD_RESULT System::getOutput(
    FMOD_OUTPUTTYPE *  output
);
```

Parameters

output

Address of a variable that receives the current output type.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_OUTPUTTYPE

System::getOutputByPlugin

Returns the currently selected output as an id in the list of output plugins.?

Syntax

```
  FMOD_RESULT System::getOutputByPlugin(
  int * index
);
```

Parameters

index

Address of a variable that receives the currently selected output as an index in a plugin list.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins
- System::setOutputByPlugin
- System::setOutput

System::getOutputHandle

Retrieves a pointer to the system level output device module. This means a pointer to a DirectX "LPDIRECTSOUND", or a WINMM handle, or with something like with FMOD_OUTPUTTYPE_NOSOUND output, the handle will be null or 0.?

Syntax

```
  FMOD_RESULT System::getOutputHandle(
    void ** handle
);
```

Parameters

handle

Address of a variable that receives the handle to the output mode's native hardware API object (see remarks).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Must be called after System::init.
Cast the resulting pointer depending on what output system pointer you are after.

FMOD_OUTPUTTYPE_WINMM Pointer to type HWAVEOUT is returned.
FMOD_OUTPUTTYPE_DSOUND Pointer to type DIRECTSOUND is returned.
FMOD_OUTPUTTYPE_ASIO Pointer to type AsioDrivers is returned.
FMOD_OUTPUTTYPE_OSS File handle is returned, (cast to int).
FMOD_OUTPUTTYPE_ESD Handle of type int is returned, as returned by so_esd_open_sound (cast to int).
FMOD_OUTPUTTYPE_ALSA Pointer to type snd_pcm_t is returned.
FMOD_OUTPUTTYPE_MAC Handle of type SndChannelPtr is returned.
FMOD_OUTPUTTYPE_Xbox Pointer to type DIRECTSOUND is returned.
FMOD_OUTPUTTYPE_PS2 NULL / 0 is returned.
FMOD_OUTPUTTYPE_GC NULL / 0 is returned.
FMOD_OUTPUTTYPE_NOSOUND NULL / 0 is returned.
FMOD_OUTPUTTYPE_WAVWRITER NULL / 0 is returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_OUTPUTTYPE
- System::setOutput
- System::init

System::getPluginInfo

Retrieves information to display for the selected plugin.?

Syntax

```
  FMOD_RESULT System::getPluginInfo(
   FMOD_PLUGINTYPE  plugintype,
  int  index,
  char *  name,
  int  namelen,
  unsigned int *  version
);
```

Parameters

plugintype

Specify the type of plugin type such as FMOD_PLUGINTYPE_OUTPUT, FMOD_PLUGINTYPE_CODEC or FMOD_PLUGINTYPE_DSP.

index

Index into the enumerated list of output plugins.

name

Address of a variable that receives the name of the plugin.

namelen

Length in bytes of the target buffer to receieve the string.

version

Version number set by the plugin.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins

System::getRecordDriver

Retrieves the currently selected recording driver, usually set with System::setRecordDriver.?

Syntax

```
 FMOD_RESULT System::getRecordDriver(
  int * driver
);
```

Parameters

driver

Address of a variable to receive the currently selected recording driver.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The number of drivers available can be retrieved with System::getRecordNumDrivers.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::setRecordDriver
- System::getRecordNumDrivers

System::getRecordDriverCaps

Returns information on capabilities of the current output mode for the selected recording sound device.?

Syntax

```
  FMOD_RESULT System::getRecordDriverCaps(
  int id,
   FMOD_CAPS * caps,
  int * minfrequency,
  int * maxfrequency
);
```

Parameters

id

Enumerated driver ID. This must be in a valid range delimited by System::getRecordNumDrivers.

caps

Address of a variable that receives the capabilities of the device. Optional. Specify 0 or NULL to ignore.

minfrequency

Address of a variable that receives the minimum frequency allowed for sounds used with recording. Optional. Specify 0 or NULL to ignore.

maxfrequency

Address of a variable that receives the maximum frequency allowed for sounds used with recording. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- FMOD_CAPS
- System::getRecordNumDrivers

System::getRecordDriverInfo

Retrieves identification information about a sound device specified by its index, and specific to the output mode set with System::setOutput.?

Syntax

```
FMOD_RESULT System::getRecordDriverInfo(
  int id,
  char * name,
  int namelen,
  FMOD_GUID * guid
);
```

Parameters

id

Index into the enumerated list of record devices up to the value returned by System::getRecordNumDrivers.

name

Address of a variable that receives the name of the recording device. Optional. Specify 0 or NULL to ignore.

namelen

Length in bytes of the target buffer to receieve the string. Required if name parameter is not NULL.

guid

Address of a variable that receives the GUID that uniquely identifies the device. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::setOutput
- System::getRecordNumDrivers

System::getRecordNumDrivers

Retrieves the number of recording devices available for this output mode. Use this to enumerate all recording devices possible so that the user can select one.?

Syntax

```
  FMOD_RESULT System::getRecordNumDrivers(
  int *  numdrivers
);
```

Parameters

numdrivers

Address of a variable that receives the number of recording drivers available for this output mode.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::GetRecordDriverInfo

System::getRecordPosition

Retrieves the current recording position of the record buffer in PCM samples.?

Syntax

```
  FMOD_RESULT System::getRecordPosition(
  unsigned int *  position
);
```

Parameters

position

Address of a variable to receieve the current recording position in PCM samples.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

System::getReverbAmbientProperties

Retrieves the default reverb envrionment for the virtual reverb system.?

Syntax

```
  FMOD_RESULT System::getReverbAmbientProperties(
    FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a pointer to a FMOD_REVERB_PROPERTIES to receieve the settings for the current ambient reverb setting.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default the ambient reverb is set to 'off'. This is the same as FMOD_REVERB_PRESET_OFF.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_PROPERTIES
- System::setReverbAmbientProperties
- System::createReverb

System::getReverbProperties

Retrieves the current reverb environment.?

Syntax

```
  FMOD_RESULT System::getReverbProperties(
    FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a variable that receives the current reverb environment description.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setReverbProperties
- Channel::setReverbProperties
- Channel::getReverbProperties

System::getSoftwareChannels

Retrieves the maximum number of software mixed channels possible. Software mixed voices are used by sounds loaded with FMOD_SOFTWARE.?

Syntax

```
  FMOD_RESULT System::getSoftwareChannels(
  int *  numsoftwarechannels
);
```

Parameters

numsoftwarechannels

Address of a variable that receives the current maximum number of software voices available.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setSoftwareChannels

System::getSoftwareFormat

Retrieves the output format for the software mixer.?

Syntax

```
  FMOD_RESULT System::getSoftwareFormat(
  int * samplerate,
   FMOD_SOUND_FORMAT * format,
  int * numoutputchannels,
  int * maxinputchannels,
   FMOD_DSP_RESAMPLER * resamplemethod,
  int * bits
);
```

Parameters

samplerate

Address of a variable that receives the mixer's output rate. Optional. Specify 0 or NULL to ignore.

format

Address of a variable that receives the mixer's output format. Optional. Specify 0 or NULL to ignore.

numoutputchannels

Address of a variable that receives the number of output channels to initialize the mixer to, for example 1 = mono, 2 = stereo. 8 is the maximum for soundcards that can handle it. Optional. Specify 0 or NULL to ignore.

maxinputchannels

Address of a variable that receives the maximum channel depth on sounds that are loadable or creatable. Specify 0 or NULL to ignore.

resamplemethod

Address of a variable that receives the current resampling (frequency conversion) method for software mixed sounds. Specify 0 or NULL to ignore.

bits

Address of a variable that receives the number of bits per sample. Useful for byte->sample conversions. for example FMOD_SOUND_FORMAT_PCM16 is 16. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that the settings returned here may differ from the settings provided by the user with System::setSoftwareFormat . This is because the driver may have changed it because it will not initialize to anything else.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setSoftwareFormat
- FMOD_SOUND_FORMAT
- FMOD_DSP_RESAMPLER

System::getSoundRAM

Retrieves the amount of dedicated sound ram available if the platform supports it. Currently only support on GameCube.
?Most platforms use main ram to store audio data, so this function usually isn't necessary.?

Syntax

```
  FMOD_RESULT System::getSoundRAM(
  int * currentalloced,
  int * maxalloced,
  int * total
);
```

Parameters

currentalloced

Address of a variable that receives the currently allocated sound ram memory at time of call. Optional. Specify 0 or NULL to ignore.

maxalloced

Address of a variable that receives the maximum allocated sound ram memory since System::init. Optional. Specify 0 or NULL to ignore.

total

Address of a variable that receives the total amount of sound ram available on this device.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In the future Will support PlayStation 2 (SPU2 RAM), and Creative X-Fi sound card when it comes out as it has dedicated sound ram.

Platforms Supported

GameCube, PlayStation 3

See Also

- Memory_GetStats

System::getSpeakerMode

Retrieves the current speaker mode.?

Syntax

```
  FMOD_RESULT System::getSpeakerMode(
    FMOD_SPEAKERMODE *  speakermode
);
```

Parameters

speakermode

Address of a variable that receives the current speaker mode.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setSpeakerMode
- FMOD_SPEAKERMODE

System::getSpectrum

Retrieves the spectrum from the currently playing output signal.?

Syntax

```
  FMOD_RESULT System::getSpectrum(
   float * spectrumarray,
  int  numvalues,
  int channeloffset,
   FMOD_DSP_FFT_WINDOW  windowtype
);
```

Parameters

spectrumarray

Address of a variable that receives the spectrum data. This is an array of floating point values. Data will range is 0.0 to 1.0. Decibels = 10.0f * (float)log10(val) * 2.0f; See remarks for what the data represents.

numvalues

Size of array in floating point values being passed to the function. Must be a power of 2. (ie 128/256/512 etc). Min = 64. Max = 8192.

channeloffset

Channel of the signal to analyze. If the signal is multichannel (such as a stereo output), then this value represents which channel to analyze. On a stereo signal 0 = left, 1 = right.

windowtype

"Pre-FFT" window method. This filters the PCM data before entering the spectrum analyzer to reduce transient frequency error for more accurate results. See FMOD_DSP_FFT_WINDOW for different types of fft window techniques possible and for a more detailed explanation.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The larger the numvalues, the more CPU the FFT will take. Choose the right value to trade off between accuracy / speed.
The larger the numvalues, the more 'lag' the spectrum will seem to inherit. This is because the FFT window size stretches the analysis back in time to what was already played. For example if the window size happened to be 44100 and the output rate was 44100 it would be analyzing the past second of data, and giving you the average spectrum over that time period.

If you are not displaying the result in dB, then the data may seem smaller than it should be. To display it you may want to normalize the data - that is, find the maximum value in the resulting spectrum, and scale all values in the array by 1 / max. (ie if the max was 0.5f, then it would become 1).
To get the spectrum for both channels of a stereo signal, call this function twice, once with channeloffset = 0, and again with channeloffset = 1. Then add the spectrums together and divide by 2 to get the average spectrum for both channels.

What the data represents.
To work out what each entry in the array represents, use this formula

```
entry_hz = (output_rate / 2) / numvalues
```

The array represents amplitudes of each frequency band from 0hz to the nyquist rate. The nyquist rate is equal to the output rate divided by 2.
For example when FMOD is set to 44100hz output, the range of represented frequencies will be 0hz to 22049hz, a total of 22050hz represented.
If in the same example, 1024 was passed to this function as the numvalues, each entry's contribution would be as follows.

```
entry_hz = (44100 / 2) / 1024
entry_hz = 21.53 hz
```

Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_FFT_WINDOW
- Channel::getSpectrum
- ChannelGroup::getSpectrum
- System::getWaveData

System::getStreamBufferSize

Returns the current internal buffersize settings for streamable sounds.?

Syntax

```
  FMOD_RESULT System::getStreamBufferSize(
  unsigned int *  filebuffersize,
   FMOD_TIMEUNIT *  filebuffersizetype
);
```

Parameters

filebuffersize

Address of a variable that receives the current stream file buffer size setting. Default is 16384 (FMOD_TIMEUNIT_RAWBYTES). Optional. Specify 0 or NULL to ignore.

filebuffersizetype

Address of a variable that receives the type of unit for the current stream file buffer size setting. Can be FMOD_TIMEUNIT_MS, FMOD_TIMEUNIT_PCM, FMOD_TIMEUNIT_PCMBYTES or FMOD_TIMEUNIT_RAWBYTES. Default is FMOD_TIMEUNIT_RAWBYTES. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- System::setStreamBufferSize

System::getUserData

Retrieves the user value that that was set by calling the System::setUserData function.?

Syntax

```
  FMOD_RESULT System::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the System::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setUserData

System::getVersion

Returns the current version of FMOD Ex being used.?

Syntax

```
  FMOD_RESULT System::getVersion(
  unsigned int * version
);
```

Parameters

version

Address of a variable that receives the current FMOD Ex version.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The version is a 32bit hexadecimal value formated as 16:8:8, with the upper 16bits being the major version, the middle 8bits being the minor version and the bottom 8bits being the development version. For example a value of 00040106h is equal to 4.01.06.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init

System::getWaveData

Retrieves a pointer to a block of PCM data that represents the currently playing audio mix.
?This function is useful for a very easy way to plot an oscilliscope.
?

Syntax

```
FMOD_RESULT System::getWaveData(
  float * wavearray,
  int  numvalues,
  int channeloffset
);
```

Parameters

wavearray

Address of a variable that receives the currently playing waveform data. This is an array of floating point values.

numvalues

Number of floats to write to the array. Maximum value = 16384.

channeloffset

Offset into multichannel data. For mono output use 0. Stereo output will use 0 = left, 1 = right. More than stereo output - use the appropriate index.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This is the actual resampled, filtered and volume scaled data of the final output, at the time this function is called.

Do not use this function to try and display the whole waveform of the sound, as this is more of a 'snapshot' of the current waveform at the time it is called, and could return the same data if it is called very quickly in succession. See the DSP API to capture a continual stream of wave data as it plays, or see Sound::lock / Sound::unlock if you want to simply display the waveform of a sound.

This function allows retrieval of left and right data for a stereo sound individually. To combine them into one signal, simply add the entries of each seperate buffer together and then divide them by 2.

Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getSpectrum
- Channel::getWaveData
- ChannelGroup::getWaveData
- Sound::lock
- Sound::unlock

System::init

Initializes the system object, and the sound device. This has to be called at the start of the user's program.
?**You must create a system object with FMOD::System_create.**?

Syntax

```
  FMOD_RESULT System::init(
  int  maxchannels,
   FMOD_INITFLAGS   flags,
   void *  extradriverdata
);
```

Parameters

maxchannels

The maximum number of channels to be used in FMOD. They are also called 'virtual channels' as you can play as many of these as you want, even if you only have a small number of hardware or software voices. See remarks for more.

flags

See FMOD_INITFLAGS. This can be a selection of flags bitwise OR'ed together to change the behaviour of FMOD at initialization time.

extradriverdata

Driver specific data that can be passed to the output plugin. For example the filename for the wav writer plugin. See FMOD_OUTPUTTYPE for what each output mode might take here. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Virtual channels.
These types of voices are the ones you work with using the FMOD::Channel API.
The advantage of virtual channels are, unlike older versions of FMOD, you can now play as many sounds as you like without fear of ever running out of voices, or playsound failing.
You can also avoid 'channel stealing' if you specify enough virtual voices.

As an example, you can play 1000 sounds at once, even on a 32 channel soundcard.
FMOD will only play the most important/closest/loudest (determined by volume/distance/geometry and priority settings) voices, and the other 968 voices will be virtualized without expense to the CPU. The voice's cursor positions are updated.
When the priority of sounds change or emulated sounds get louder than audible ones, they will swap the actual voice

resource over (ie hardware or software buffer) and play the voice from its correct position in time as it should be heard.
What this means is you can play all 1000 sounds, if they are scattered around the game world, and as you move around the world you will hear the closest or most important 32, and they will automatically swap in and out as you move.

Currently the maximum channel limit is 4093.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_INITFLAGS
- System::close
- System_Create
- FMOD_OUTPUTTYPE

System::isRecording

Retrieves the state of the FMOD recording API, ie if it is currently recording or not.?

Syntax

```
  FMOD_RESULT System::isRecording(
    bool * recording
);
```

Parameters

recording

Address of a variable to receive the current recording state. True or non zero if the FMOD recording api is currently in the middle of recording, false or zero if the recording api is stopped / not recording.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Recording can be started with System::recordStart.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::recordStart
- System::recordStop

System::loadGeometry

Creates a geometry object from a block of memory which contains pre-saved geometry data, saved by Geometry::save.?

Syntax

```
  FMOD_RESULT System::loadGeometry(
  const void *  data,
  int  datasize,
    FMOD::Geometry  **  geometry
);
```

Parameters

data

Address of data containing pre-saved geometry data.

datasize

Size of geometry data block in bytes.

geometry

Address of a variable to receive a newly created FMOD::Geometry object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::save
- System::createGeometry

System::loadPlugin

Loads an FMOD plugin. This could be a DSP, file format or output plugin.?

Syntax

```
FMOD_RESULT System::loadPlugin(
  const char *  filename,
  FMOD_PLUGINTYPE *  plugintype,
  int *  index
);
```

Parameters

filename

Filename of the plugin to be loaded.

plugintype

Address of a variable that receives the type of plugin that has been loaded (if successful).

index

Index into the plugin list for that plugin type. Use this for DSP plugins created with System::createDSPByIndex.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Once the plugin is loaded, it can be enumerated and used.
For file format plugins, FMOD will automatically try to use them when System::createSound is used.
For DSP plugins, you can enumerate them with System::getNumPlugins and System::getPluginInfo.
Plugins can be created for FMOD by the user. See the relevant section in the documentation on creating plugins.
The format of the plugin is dependant on the operating system.
On Win32 and Win64 the .dll format is used
On Linux, the .so format is used.
On Macintosh, the .shlib format is used?

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh.

See Also

- System::setPluginPath
- System::createSound
- System::getNumPlugins
- System::getPluginInfo
- System::createDSPByIndex

System::lockDSP

Mutual exclusion function to lock the FMOD DSP engine (which runs asynchronously in another thread), so that it will not execute. If the FMOD DSP engine is already executing, this function will block until it has completed.
?The function may be used to synchronize DSP network operations carried out by the user.
?An example of using this function may be for when the user wants to construct a DSP sub-network, without the DSP engine executing in the background while the sub-network is still under construction.
?

Syntax

```
FMOD_RESULT System::lockDSP();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Once the user no longer needs the DSP engine locked, it must be unlocked with System::unlockDSP.
Note that the DSP engine should not be locked for a significant amount of time, otherwise inconsistency in the audio output may result. (audio skipping/stuttering).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::unlockDSP

System::playDSP

Plays a DSP unit object and its input network on a particular channel.?

Syntax

```
  FMOD_RESULT System::playDSP(
    FMOD_CHANNELINDEX channelid,
    FMOD::DSP *  dsp,
    bool  paused,
    FMOD::Channel ** channel
);
```

Parameters

channelid

Use the value FMOD_CHANNEL_FREE to get FMOD to pick a free channel. Otherwise specify a channel number from 0 to the 'maxchannels' value specified in System::init minus 1.

dsp

Pointer to the dsp unit to play. This is opened with System::createDSP, System::createDSPByType, System::createDSPByIndex.

paused

True or false flag to specify whether to start the channel paused or not. Starting a channel paused allows the user to alter its attributes without it being audible, and unpausing with Channel::setPaused actually starts the dsp running.

channel

Address of a channel handle pointer that receives the newly playing channel. If FMOD_CHANNEL_REUSE is used, this can contain a previously used channel handle and FMOD will re-use it to play a dsp on.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a dsp is played, it will use the dsp's default frequency, volume, pan, levels and priority.

A dsp defined as FMOD_3D will by default play at the position of the listener.

To change channel attributes before the dsp is audible, start the channel paused by setting the paused flag to true, and calling the relevant channel based functions. Following that, unpause the channel with Channel::setPaused.

If FMOD_CHANNEL_FREE is used as the channel index, it will pick an arbitrary free channel and use channel management. (As described below).
If FMOD_CHANNEL_REUSE is used as the channel index, FMOD Ex will re-use the channel handle that is passed in as the 'channel' parameter. If NULL or 0 is passed in as the channel handle it will use the same logic as FMOD_CHANNEL_FREE and pick an arbitrary channel.

Channels are reference counted. If a channel is stolen by the FMOD priority system, then the handle to the stolen voice becomes invalid, and Channel based commands will not affect the new channel playing in its place.
If all channels are currently full playing a dsp or sound, FMOD will steal a channel with the lowest priority dsp or sound.
If more channels are playing than are currently available on the soundcard/sound device or software mixer, then FMOD will 'virtualize' the channel. This type of channel is not heard, but it is updated as if it was playing. When its priority becomes high enough or another sound stops that was using a real hardware/software channel, it will start playing from where it should be. This technique saves CPU time (thousands of sounds can be played at once without actually being mixed or taking up resources), and also removes the need for the user to manage voices themselves.
An example of virtual channel usage is a dungeon with 100 torches burning, all with a looping crackling sound, but with a soundcard that only supports 32 hardware voices. If the 3D positions and priorities for each torch are set correctly, FMOD will play all 100 sounds without any 'out of channels' errors, and swap the real voices in and out according to which torches are closest in 3D space.
Priority for virtual channels can be changed in the sound's defaults, or at runtime with Channel::setPriority.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_CHANNELINDEX
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- Channel::setPaused
- Channel::setPriority
- DSP::setDefaults
- System::init

System::playSound

Plays a sound object on a particular channel.?

Syntax

```
  FMOD_RESULT System::playSound(
    FMOD_CHANNELINDEX channelid,
    FMOD::Sound * sound,
    bool paused,
    FMOD::Channel ** channel
);
```

Parameters

channelid

Use the value FMOD_CHANNEL_FREE to get FMOD to pick a free channel. Otherwise specify a channel number from 0 to the 'maxchannels' value specified in System::init minus 1.

sound

Pointer to the sound to play. This is opened with System::createSound.

paused

True or false flag to specify whether to start the channel paused or not. Starting a channel paused allows the user to alter its attributes without it being audible, and unpausing with Channel::setPaused actually starts the sound.

channel

Address of a channel handle pointer that receives the newly playing channel. If FMOD_CHANNEL_REUSE is used, this can contain a previously used channel handle and FMOD will re-use it to play a sound on.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a sound is played, it will use the sound's default frequency, volume, pan, levels and priority.

A sound defined as FMOD_3D will by default play at the position of the listener.

To change channel attributes before the sound is audible, start the channel paused by setting the paused flag to true, and calling the relevant channel based functions. Following that, unpause the channel with Channel::setPaused.

If FMOD_CHANNEL_FREE is used as the channel index, it will pick an arbitrary free channel and use channel

management. (As described below).
If FMOD_CHANNEL_REUSE is used as the channel index, FMOD Ex will re-use the channel handle that is passed in as the 'channel' parameter. If NULL or 0 is passed in as the channel handle it will use the same logic as FMOD_CHANNEL_FREE and pick an arbitrary channel.

Channels are reference counted. If a channel is stolen by the FMOD priority system, then the handle to the stolen voice becomes invalid, and Channel based commands will not affect the new sound playing in its place.
If all channels are currently full playing a sound, FMOD will steal a channel with the lowest priority sound.
If more channels are playing than are currently available on the soundcard/sound device or software mixer, then FMOD will 'virtualize' the channel. This type of channel is not heard, but it is updated as if it was playing. When its priority becomes high enough or another sound stops that was using a real hardware/software channel, it will start playing from where it should be. This technique saves CPU time (thousands of sounds can be played at once without actually being mixed or taking up resources), and also removes the need for the user to manage voices themselves.
An example of virtual channel usage is a dungeon with 100 torches burning, all with a looping crackling sound, but with a soundcard that only supports 32 hardware voices. If the 3D positions and priorities for each torch are set correctly, FMOD will play all 100 sounds without any 'out of channels' errors, and swap the real voices in and out according to which torches are closest in 3D space.
Priority for virtual channels can be changed in the sound's defaults, or at runtime with Channel::setPriority.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_CHANNELINDEX
- System::createSound
- Channel::setPaused
- Channel::setPriority
- Sound::setDefaults
- Sound::setVariations
- System::init

System::recordStart

Starts the recording engine recording to the specified recording sound.?

Syntax

```
  FMOD_RESULT System::recordStart(
    FMOD::Sound * sound,
    bool  loop
);
```

Parameters

sound

User created sound for the user to record to.

loop

Boolean flag to tell the recording engine whether to continue recording to the provided sound from the start again, after it has reached the end. If this is set to true the data will be continually be overwritten once every loop. See remarks.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::recordStop

System::recordStop

Stops the recording engine from recording to the specified recording sound.?

Syntax

```
FMOD_RESULT System::recordStop();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::recordStart

System::release

Closes and frees a system object and its resources.?

Syntax

```
FMOD_RESULT System::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function also calls System::close, so calling close before this function is not necessary.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System_Create
- System::init
- System::close

System::set3DListenerAttributes

This updates the position, velocity and orientation of the specified 3D sound listener.?

Syntax

```
  FMOD_RESULT System::set3DListenerAttributes(
  int  listener,
  const FMOD_VECTOR *  pos,
  const FMOD_VECTOR *  vel,
  const FMOD_VECTOR *  forward,
  const FMOD_VECTOR * up
);
```

Parameters

listener

Listener ID in a multi-listener environment. Specify 0 if there is only 1 listener.

pos

The position of the listener in world space, measured in distance units. You can specify 0 or NULL to not update the position.

vel

The velocity of the listener measured in distance units **per second**. You can specify 0 or NULL to not update the velocity of the listener.

forward

The forwards orientation of the listener. This vector must be of unit length and perpendicular to the up vector. You can specify 0 or NULL to not update the forwards orientation of the listener.

up

The upwards orientation of the listener. This vector must be of unit length and perpendicular to the forwards vector. You can specify 0 or NULL to not update the upwards orientation of the listener.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default, FMOD uses a left-handed co-ordinate system. This means +X is right, +Y is up, and +Z is forwards.
To change this to a right-handed coordinate system, use FMOD_INIT_3D_RIGHTHANDED. This means +X is

left, +Y is up, and +Z is forwards.

To map to another coordinate system, flip/negate and exchange these values.

Orientation vectors are expected to be of UNIT length. This means the magnitude of the vector should be 1.0.

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.

Always remember to use **units per second**, *not* units per frame as this is a common mistake and will make the doppler effect sound wrong.
For example, Do not just use (pos - lastpos) from the last frame's data for velocity, as this is not correct. You need to time compensate it so it is given in units per **second**.
You could alter your pos - lastpos calculation to something like this.

```
vel = (pos-lastpos) / time_taken_since_last_frame_in_seconds.
```
I.e. at 60fps the formula would look like this vel = (pos-lastpos) / 0.0166667.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::get3DListenerAttributes
- FMOD_INITFLAGS
- System::set3DSettings
- System::get3DSettings
- FMOD_VECTOR

System::set3DNumListeners

Sets the number of 3D 'listeners' in the 3D sound scene. This function is useful mainly for split-screen game purposes.?

Syntax

```
  FMOD_RESULT System::set3DNumListeners(
  int  numlisteners
);
```

Parameters

numlisteners

Number of listeners in the scene. Valid values are from 1-4 inclusive. Default = 1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If the number of listeners is set to more than 1, then panning and doppler are turned off. All sound effects will be mono.
FMOD uses a 'closest sound to the listener' method to determine what should be heard in this case.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::get3DNumListeners
- System::set3DListenerAttributes

System::set3DRolloffCallback

When FMOD wants to calculate 3d volume for a channel, this callback can be used to override the internal volume calculation based on distance.?

Syntax

```
  FMOD_RESULT System::set3DRolloffCallback(
    FMOD_3D_ROLLOFFCALLBACK  callback
);
```

Parameters

callback

Pointer to a C function of type FMOD_3D_ROLLOFFCALLBACK, that is used to override the FMOD volume calculation. Default is 0 or NULL. Setting the callback to null will return 3d calculation back to FMOD.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function overrides FMOD_3D_LOGROLLOFF, FMOD_3D_LINEARROLLOFF, FMOD_3D_CUSTOMROLLOFF. To allow FMOD to calculate the 3d volume again, use 0 or NULL as the callback.

NOTE: When using the event system, call Channel::getUserData from your FMOD_3D_ROLLOFFCALLBACK to get the event instance handle of the event that spawned the channel in question.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_3D_ROLLOFFCALLBACK
- System::set3DListenerAttributes
- System::get3DListenerAttributes
- Channel::getUserData

System::set3DSettings

Sets the global doppler scale, distance factor and log rolloff scale for all 3D sound in FMOD.?

Syntax

```
  FMOD_RESULT System::set3DSettings(
    float  dopplerscale,
    float  distancefactor,
    float  rolloffscale
);
```

Parameters

dopplerscale

Scaling factor for doppler shift. Default = 1.0.

distancefactor

Relative distance factor to FMOD's units. Default = 1.0. (1.0 = 1 metre).

rolloffscale

Scaling factor for 3D sound rolloff or attenuation for FMOD_3D_LOGROLLOFF based sounds only (which is the default type). Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The doppler scale is a general scaling factor for how much the pitch varies due to doppler shifting in 3D sound. Doppler is the pitch bending effect when a sound comes towards the listener or moves away from it, much like the effect you hear when a train goes past you with its horn sounding. With dopplerscale you can exaggerate or diminish the effect.
FMOD's effective speed of sound at a doppler factor of 1.0 is 340 m/s.

The distance factor is the FMOD 3D engine relative distance factor, compared to 1.0 meters.
Another way to put it is that it equates to "how many units per meter' does your engine have".
For example. If you are using feet then scale would equal 3.28.
Note! This only affects doppler! If you keep your min/max distance, custom rolloff curves and positions in scale relative to each other the volume rolloff will not change.
If you set this, the mindistance of a sound will automatically set itself to this value when it is created in case the user forgets to set the mindistance to match the new distancefactor.

The rolloff scale sets the global attenuation rolloff factor for FMOD_3D_LOGROLLOFF based sounds only (which

is the default).
Normally volume for a sound will scale at mindistance / distance. This gives a logarithmic attenuation of volume as the source gets further away (or closer).
Setting this value makes the sound drop off faster or slower. The higher the value, the faster volume will attenuate, and conversely the lower the value, the slower it will attenuate.
For example a rolloff factor of 1 will simulate the real world, where as a value of 2 will make sounds attenuate 2 times quicker.
rolloffscale has no effect for FMOD_3D_LINEARROLLOFF or FMOD_3D_CUSTOMROLLOFF.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::get3DSettings
- Sound::set3DMinMaxDistance
- Sound::get3DMinMaxDistance
- Channel::set3DAttributes
- Channel::get3DAttributes

System::set3DSpeakerPosition

This function allows the user to specify the position of their actual physical speaker to account for non standard setups.
?It also allows the user to disable speakers from 3D consideration in a game.
?The funtion is for describing the 'real world' speaker placement to provide a more natural panning solution for 3d sound. Graphical configuration screens in an application could draw icons for speaker placement that the user could position at their will.?

Syntax

```
FMOD_RESULT System::set3DSpeakerPosition(
  FMOD_SPEAKER  speaker,
  float  x,
  float  y,
  bool  active
);
```

Parameters

speaker

The selected speaker of interest to position.

x

The 2D X offset in relation to the listening position. For example -1.0 would mean the speaker is on the left, and +1.0 would mean the speaker is on the right. 0.0 is the speaker is in the middle.

y

The 2D Y offset in relation to the listening position. For example -1.0 would mean the speaker is behind the listener, and +1 would mean the speaker is in front of the listener.

active

Enables or disables speaker from 3D consideration. Useful for disabling center speaker for vocals for example, or the LFE. x and y can be anything in this case.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! This only affects software mixed 3d sounds, created with FMOD_SOFTWARE and FMOD_3D.

A typical 7.1 setup would look like this.

```
system->set3DSpeakerPosition(FMOD_SPEAKER_FRONT_LEFT,    -1.0f,  1.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_FRONT_RIGHT,    1.0f,  1.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_FRONT_CENTER,   0.0f,  1.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_LOW_FREQUENCY,  0.0f,  0.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_BACK_LEFT,     -1.0f, -1.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_BACK_RIGHT,     1.0f, -1.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_SIDE_LEFT,     -1.0f,  0.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_SIDE_RIGHT,     1.0f,  0.0f, true);
```

A typical stereo setup would look like this.

```
system->set3DSpeakerPosition(FMOD_SPEAKER_FRONT_LEFT,    -1.0f,  0.0f, true);
system->set3DSpeakerPosition(FMOD_SPEAKER_FRONT_RIGHT,    1.0f,  0.0f, true);
```

You could use this function to make sounds in front of your come out of different physical speakers. If you specified for example that FMOD_SPEAKER_SIDE_RIGHT was in front of you at and you organized the other speakers accordingly the 3d audio would come out of the side right speaker when it was in front instead of the default which is only to the side.
This function is also useful if speakers are not 'perfectly symmetrical'. For example if the center speaker was closer to the front left than the front right, this function could be used to position that center speaker accordingly and FMOD would skew the panning appropriately to make it sound correct again.

The 2d coordinates used are only used to generate angle information. Size / distance does not matter in FMOD's implementation because it is not FMOD's job to attenuate or amplify the signal based on speaker distance. If it amplified the signal in the digital domain the audio could clip/become distorted. It is better to use the amplifier's analogue level capabilities to balance speaker volumes.
Calling System::setSpeakerMode overrides these values, so this function must be called after this.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::get3DSpeakerPosition
- System::setSpeakerMode
- FMOD_SPEAKERMODE
- FMOD_SPEAKER

System::setAdvancedSettings

Sets advanced features like configuring memory and cpu usage for FMOD_CREATECOMPRESSEDSAMPLE usage.?

Syntax

```
  FMOD_RESULT System::setAdvancedSettings(
    FMOD_ADVANCEDSETTINGS * settings
);
```

Parameters

settings

Pointer to FMOD_ADVANCEDSETTINGS structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_ADVANCEDSETTINGS
- System::getAdvancedSettings
- FMOD_MODE

System::setCallback

Sets a callback for a system for a specific event.?

Syntax

```
 FMOD_RESULT System::setCallback(
   FMOD_SYSTEM_CALLBACKTYPE  type,
   FMOD_SYSTEM_CALLBACK  callback
);
```

Parameters

type

The callback type, for example a 'device list changed' callback.

callback

Pointer to a callback to receive the event when it happens.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Currently some system callbacks are driven by System::update and will only occur when this function is called. This has the main advantage of far less complication due to thread issues, and allows all FMOD commands, including loading sounds and playing new sounds from the callback.
The only disadvantage is that callbacks are not asynchronous and are bound by the latency caused by the rate the user calls the update command.
Callbacks are stdcall. Use F_CALLBACK inbetween your return type and function name.
Example:

```
 FMOD_RESULT F_CALLBACK mycallback(FMOD_SYSTEM *system, FMOD_SYSTEM_CALLBACKTYPE type, 
unsigned int commanddata1, unsigned int commanddata2)
{


       FMOD::System *cppsystem = (FMOD::System *)system;


      // More code goes here.


       return FMOD_OK;


}
```

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::update
- FMOD_SYSTEM_CALLBACK
- FMOD_SYSTEM_CALLBACKTYPE

System::setDSPBufferSize

Sets the FMOD internal mixing buffer size. This function is used if you need to control mixer latency or granularity.?Smaller buffersizes lead to smaller latency, but can lead to stuttering/skipping/instable sound on slower machines or soundcards with bad drivers.?

Syntax

```
  FMOD_RESULT System::setDSPBufferSize(
  unsigned int  bufferlength,
  int  numbuffers
);
```

Parameters

bufferlength

The mixer engine block size in samples. Use this to adjust mixer update granularity. Default = 1024. (milliseconds = 1024 at 48khz = 1024 / 48000 * 1000 = 21.33ms). This means the mixer updates every 21.33ms.

numbuffers

The mixer engine number of buffers used. Use this to adjust mixer latency. Default = 4. To get the total buffersize multiply the bufferlength by the numbuffers value. By default this would be 4*1024.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The FMOD software mixer mixes to a ringbuffer. The size of this ringbuffer is determined here. It mixes a block of sound data every 'bufferlength' number of samples, and there are 'numbuffers' number of these blocks that make up the entire ringbuffer.
Adjusting these values can lead to extremely low latency performance (smaller values), or greater stability in sound output (larger values).

Warning! The 'buffersize' is generally best left alone. Making the granularity smaller will just increase CPU usage (cache misses and DSP network overhead). Making it larger affects how often you hear commands update such as volume/pitch/pan changes. Anything above 20ms will be noticable and sound parameter changes will be obvious instead of smooth.

FMOD chooses the most optimal size by default for best stability, depending on the output type, and if the drivers are emulated or not (for example DirectSound is emulated using waveOut on NT). It is not recommended changing this value unless you really need to. You may get worse performance than the default settings chosen by FMOD.

To convert from milliseconds to 'samples', simply multiply the value in milliseconds by the sample rate of the output (ie 48000 if that is what it is set to), then divide by 1000.

The values in milliseconds and average latency expected from the settings can be calculated using the following code.

```
 FMOD_RESULT result;
unsigned int blocksize;
int numblocks;
 float ms;

 result = system->getDSPBufferSize(?
 result = system->getSoftwareFormat(?

ms = (float)blocksize * 1000.0f / (float)frequency;

 printf("Mixer blocksize        = %.02f ms\n", ms);
 printf("Mixer Total buffersize = %.02f ms\n", ms * numblocks);
 printf("Mixer Average latency  = %.02f ms\n", ms * ((float)numblocks - 1.5f));
```

Platform notes: Some output modes (such as FMOD_OUTPUTTYPE_ASIO) will change the buffer size to match their own internal optimal buffer size. Use System::getDSPBufferSize after calling System::init to see if this is the case.
Linux output modes will ignore numbuffers and just write the buffer size to the output every time it can. It does not use a ringbuffer.
Xbox 360 defaults to 256 sample buffersize and 4 for numblocks. This gives a 5.333ms granularity with roughly a 10-15ms latency.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDSPBufferSize
- System::getSoftwareFormat
- System::init
- System::close

System::setDriver

Selects a soundcard driver.?This function is used when an output mode has enumerated more than one output device, and you need to select between them.?

Syntax

```
  FMOD_RESULT System::setDriver(
  int  driver
);
```

Parameters

driver

Driver number to select. 0 = primary or main sound device as selected by the operating system settings. Use System::getNumDrivers to select a specific device.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If this function is called after FMOD is already initialized with System::init, the current driver will be shutdown and the newly selected driver will be initialized / started.

When switching output driver after System::init there are a few considerations to make:

All sounds must be created with FMOD_SOFTWARE, creating even one FMOD_HARDWARE sound will cause this function to return FMOD_ERR_NEEDSSOFTWARE.

The driver that you wish to change to must support the current output format, sample rate, and number of channels. If it does not, FMOD_ERR_OUTPUT_INIT is returned and driver state is cleared. You should now call System::setDriver with your original driver index to restore driver state (providing that driver is still available / connected) or make another selection.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDriver

- System::getNumDrivers
- System::getDriverInfo
- System::setOutput
- System::init
- System::close

System::setFileSystem

Specify user callbacks for FMOD's internal file manipulation functions.
?If ANY of the callback functions are set to 0/ NULL, then FMOD will switch back to its own file routines.
?This function is useful for replacing FMOD's file system with a game system's own file reading API.
?

Syntax

```
  FMOD_RESULT System::setFileSystem(
    FMOD_FILE_OPENCALLBACK  useropen,
    FMOD_FILE_CLOSECALLBACK  userclose,
    FMOD_FILE_READCALLBACK  userread,
    FMOD_FILE_SEEKCALLBACK  userseek,
  int  blockalign
);
```

Parameters

useropen

Callback for opening a file. Specifying 0 / null will disable file callbacks.

userclose

Callback for closing a file. Specifying 0 / null will disable file callbacks.

userread

Callback for reading from a file. Specifying 0 / null will disable file callbacks.

userseek

Callback for seeking within a file. Specifying 0 / null will disable file callbacks.

blockalign

Internal minimum file block alignment. FMOD will read data in at least chunks of this size if you ask it to. Specifying 0 means there is no file buffering at all (this could adversely affect streaming). Do NOT make this a large value, it is purely a setting for minimum sector size alignment to aid seeking and reading on certain media. It is not for stream buffer sizes, that is what System::setStreamBufferSize is for. It is recommened just to pass -1. Large values just mean large memory usage with no benefit. Specify -1 to not set this value. Default = 2048.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This has no effect on sounds loaded with FMOD_OPENMEMORY or FMOD_CREATEUSER.

This function can be used to set user file callbacks, or if required, they can be turned off by specifying 0.
This function can be used purely to set the 'buffersize' parameter, and ignore the callback aspect of the function.

Warning : This function can cause unpredictable behaviour if not used properly. You must return the right values, and each command must work properly, or FMOD will not function, or it may even crash if you give it invalid data.
You must also return FMOD_ERR_FILE_EOF from a read callback if the number of bytes read is smaller than the number of bytes requested.

FMOD's default filsystem buffers reads every 2048 bytes by default. This means every time fmod reads one byte from the API (say if it was parsing a file format), it simply mem copies the byte from the 2k memory buffer, and every time it needs to, refreshes the 2k buffer resulting in a drastic reduction in file I/O. Large reads go straight to the pointer instead of the 2k buffer if it is buffer aligned. This value can be increased or decreased by the user. A buffer of 0 means all reads go directly to the pointer specified. 2048 bytes is the size of a CD sector on most CD ISO formats so it is chosen as the default, for optimal reading speed from CD media.

NOTE! Do not force a cast from your function pointer to the FMOD_FILE_xxxCALLBACK type! Never try to 'force' fmod to accept your function. If there is an error then find out what it is. Remember to include F_CALLBACK between the return type and the function name, this equates to stdcall which you must include otherwise (besides not compiling) it will cause problems such as crashing and callbacks not being called.
NOTE! Your file callbacks must be thread safe. If not unexpected behaviour may occur. FMOD calls file functions from asynchronous threads, such as the streaming thread, and thread related to FMOD_NONBLOCKING flag.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- System::attachFileSystem
- FMOD_FILE_OPENCALLBACK
- FMOD_FILE_CLOSECALLBACK
- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK

System::setGeometrySettings

Sets the maximum world size for the geometry engine for performance / precision reasons.?

Syntax

```
  FMOD_RESULT System::setGeometrySettings(
    float  maxworldsize
);
```

Parameters

maxworldsize

Maximum size of the world from the centerpoint to the edge using the same units used in other 3D functions.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Setting maxworldsize should be done first before creating any geometry.
It can be done any time afterwards but may be slow in this case.

Objects or polygons outside the range of maxworldsize will not be handled efficiently.
Conversely, if maxworldsize is excessively large, the structure may loose precision and efficiency may drop.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createGeometry
- System::getGeometrySettings

System::setHardwareChannels

This function allows the user to request a minimum number of hardware voices to be present on the soundcard to allow hardware 3D sound acceleration, or clamp the number of hardware 3D voices to a maximum value.?

Syntax

```
  FMOD_RESULT System::setHardwareChannels(
  int min2d,
  int max2d,
  int min3d,
  int max3d
);
```

Parameters

min2d

Minimum number of hardware voices on a soundcard required to actually support hardware 2D sound. If the soundcard does not match this value for number of hardware voices possible, FMOD will place the sound into software mixed buffers instead hardware mixed buffers to guarantee the number of sounds playable at once is guaranteed.

max2d

Maximum number of hardware voices to be used by FMOD. This clamps the polyphony of hardware 2D voices to a user specified number. This could be used to limit the number of 2D hardware voices possible at once so that it doesn't sound noisy, or the user might want to limit the number of channels used for 2D hardware support to avoid problems with certain buggy soundcard drivers that report they have many channels but actually don't.

min3d

Minimum number of hardware voices on a soundcard required to actually support hardware 3D sound. If the soundcard does not match this value for number of hardware voices possible, FMOD will place the sound into software mixed buffers instead hardware mixed buffers to guarantee the number of sounds playable at once is guaranteed.

max3d

Maximum number of hardware voices to be used by FMOD. This clamps the polyphony of hardware 3D voices to a user specified number. This could be used to limit the number of 3D hardware voices possible at once so that it doesn't sound noisy, or the user might want to limit the number of channels used for 3D hardware support to avoid problems with certain buggy soundcard drivers that report they have many channels but actually don't.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 'min' value sets the minimum allowable hardware channels before FMOD drops back to 100 percent software based buffers for sounds even if they are allocated with FMOD_HARDWARE.
This is helpful for minimum spec cards, and not having to 'guess' how many hardware channels they might have. This way you can guarantee and assume a certain number of channels for your application and always allocate with FMOD_HARDWARE | FMOD_3D without fear of the playsound failing.

The 'max' value function has nothing to do with the 'min' value, in that this is not a function that forces FMOD channels into software mode if a card has less than or more than a certain number of channels.
This parameter only sets a limit on hardware channels playable at once, so if your card has 96 hardware channels, and you set max to 10, then you will only have 10 hardware 3D channels to use.
The 'buggy soundcard driver' issue in the description for the 'max' parameter is to do with one known sound card driver in particular, the default Windows XP SoundBlaster Live drivers. They report over 32 possible voices, but actually only support 32, and when you use the extra voices the driver can act unpredictably causing either sound dropouts or a crash.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getHardwareChannels
- System::init
- System::close

System::setNetworkProxy

Set a proxy server to use for all subsequent internet connections.?

Syntax

```
  FMOD_RESULT System::setNetworkProxy(
  const char *  proxy
);
```

Parameters

proxy

The name of a proxy server in host:port format e.g. www.fmod.org:8888 (defaults to port 80 if no port is specified).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Basic authentication is supported. To use it, this parameter must be in user:password@host:port format e.g. bob:sekrit123@www.fmod.org:8888 Set this parameter to 0 / NULL if no proxy is required.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, PlayStation 3

See Also

- System::getNetworkProxy

System::setNetworkTimeout

Set the timeout for network streams.?

Syntax

```
  FMOD_RESULT System::setNetworkTimeout(
  int  timeout
);
```

Parameters

timeout

The timeout value in ms.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, PlayStation 3

See Also

- System::getNetworkTimeout

System::setOutput

This function selects the output mode for the platform. This is for selecting different OS specific API's which might have different features.?

Syntax

```
  FMOD_RESULT System::setOutput(
   FMOD_OUTPUTTYPE  output
);
```

Parameters

output

Output type to select. See type list for different output types you can select.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is not necessary to call. It is only if you want to specifically switch away from the default output mode for the operating system. The most optimal mode is selected by default for the operating system. For example **FMOD_OUTPUTTYPE_DSOUND** is selected on all operating systems except for Windows NT, where FMOD_OUTPUTTYPE_WINMM is selected because it is lower latency / faster.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_OUTPUTTYPE
- System::init
- System::close

System::setOutputByPlugin

Selects an output type based on the enumerated list of outputs including FMOD and 3rd party output plugins.?

Syntax

```
  FMOD_RESULT System::setOutputByPlugin(
  int index
);
```

Parameters

index

Index into the enumerated list of output plugins.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins
- System::getOutputByPlugin
- System::setOutput
- System::init
- System::close

System::setPluginPath

Specify a base search path for plugins so they can be placed somewhere else than the directory of the main executable.?

Syntax

```
  FMOD_RESULT System::setPluginPath(
  const char *  path
);
```

Parameters

path

A character string containing a correctly formatted path to load plugins from.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 'plugin' version of FMOD relies on plugins, so when System::init is called it tries to load all FMOD registered plugins.
This path is where it will attempt to load from.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::loadPlugin
- System::init

System::setRecordDriver

Selects a recording driver.
?This function is used when an output mode has enumerated more than one record device, and you need to select between them.?

Syntax

```
  FMOD_RESULT System::setRecordDriver(
  int  driver
);
```

Parameters

driver

Record driver number to select.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function must be called before System::recordStart or after System::recordStop.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Solaris

See Also

- System::getRecordDriver
- System::getRecordNumDrivers
- System::GetRecordDriverInfo
- System::recordStart
- System::setOutput
- System::init
- System::close

System::setReverbAmbientProperties

Sets a 'background' default reverb environment for the virtual reverb system. This is a reverb preset that will be morphed to if the listener is not within any virtual reverb zones.
?By default the ambient reverb is set to 'off'.?

Syntax

```
FMOD_RESULT System::setReverbAmbientProperties(
  FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a FMOD_REVERB_PROPERTIES structure containing the settings for the desired ambient reverb setting.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

There is one reverb DSP dedicated to providing a 3D reverb effect. This DSP's properties are a weighted sum of all the contributing virtual reverbs.
The default 3d reverb properties specify the reverb properties in the 3D volumes which has no virtual reverbs defined.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_PROPERTIES
- System::getReverbAmbientProperties
- System::createReverb

System::setReverbProperties

Sets parameters for the global reverb environment.
?Reverb parameters can be set manually, or automatically using the pre-defined presets given in the fmod.h header.?

Syntax

```
FMOD_RESULT System::setReverbProperties(
  const FMOD_REVERB_PROPERTIES * prop
);
```

Parameters

prop

Address of an FMOD_REVERB_PROPERTIES structure which defines the attributes for the reverb.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

With FMOD_HARDWARE on Windows using EAX, the reverb will only work on FMOD_3D based sounds. FMOD_SOFTWARE does not have this problem and works on FMOD_2D and FMOD_3D based sounds.

On PlayStation 2, the reverb is limited to only a few reverb types that are not configurable. Use the FMOD_PRESET_PS2_xxx presets.
On Xbox, it is possible to apply reverb to FMOD_2D and FMOD_HARDWARE based voices using this function. By default reverb is turned off for FMOD_2D hardware based voices.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_PROPERTIES
- System::getReverbProperties
- Channel::setReverbProperties
- Channel::getReverbProperties

System::setSoftwareChannels

Sets the maximum number of software mixed channels possible. Software mixed voices are used by sounds loaded with FMOD_SOFTWARE.?

Syntax

```
  FMOD_RESULT System::setSoftwareChannels(
  int  numsoftwarechannels
);
```

Parameters

numsoftwarechannels

The maximum number of FMOD_SOFTWARE mixable voices to be allocated by FMOD. If you don't require software mixed voices specify 0. Default = 32.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

32 voices are allocated by default to be played simultaneously in software.
To turn off the software mixer completely including hardware resources used for the software mixer, specify FMOD_INIT_SOFTWARE_DISABLE in System::init.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- FMOD_INITFLAGS
- System::init
- System::close
- System::getSoftwareChannels

System::setSoftwareFormat

Sets the output format for the software mixer. This includes the bitdepth, sample rate and number of output channels.
?Do not call this unless you explicity want to change something. Calling this could have adverse impact on the performance and panning behaviour.
?

Syntax

```
  FMOD_RESULT System::setSoftwareFormat(
  int  samplerate,
   FMOD_SOUND_FORMAT  format,
  int  numoutputchannels,
  int  maxinputchannels,
   FMOD_DSP_RESAMPLER  resamplemethod
);
```

Parameters

samplerate

The soundcard's output rate. default = 48000.

format

The soundcard's output format. default = FMOD_SOUND_FORMAT_PCM16.

numoutputchannels

The number of output channels / speakers to initialize the soundcard to. 0 = keep speakermode setting (set with System::setSpeakerMode). If anything else than 0 is specified then the speakermode will be overriden and will become FMOD_SPEAKERMODE_RAW, meaning logical speaker assignments (as defined in FMOD_SPEAKER) become innefective and cannot be used. Channel::setPan will also fail. Default = 2 (FMOD_SPEAKERMODE_STEREO).

maxinputchannels

Optional. Specify 0 to ignore. Default = 6. Maximum channel count in loaded/created sounds to be supported. This is here purely for memory considerations and affects how much memory is used in the software mixer when allocating matrices for panning. Do not confuse this with recording, or anything to do with how many voices you can play at once. This is purely for setting the largest type of sound you can play (ie 1 = mono, 2 = stereo, etc.). Most of the time the user will not play sounds any larger than mono or stereo, so setting this to 2 would save memory and cover most sounds that are playable.

resamplemethod

Software engine resampling method. default = FMOD_DSP_RESAMPLER_LINEAR. See FMOD_DSP_RESAMPLER for different types.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! The settings in this function *may* be overriden by the output mode.
FMOD_OUTPUTTYPE_ASIO will always change the output mode to FMOD_SOUND_FORMAT_PCMFLOAT to be compatible with the output formats selectable by the ASIO control panel.
FMOD_OUTPUTTYPE_ASIO will also change the samplerate specified by the user to the one selected in the ASIO control panel.
Use System::getSoftwareFormat after System::init to determine what the output has possibly changed the format to.
Call it after System::init.

It is dependant on the output whether it will force a format change and override these settings or not.

If the output does not support the output mode specified System::init will fail, and you will have to try another setting.

Note! When this function is called with a output channel count greater than 0, the speaker mode is set to FMOD_SPEAKERMODE_RAW. FMOD does not know when you specify a number of output channels what type of speaker system it is connected to, so Channel::setPan or Channel::setSpeakerMix will then fail to work.
Calling System::setSpeakerMode will override the output channel speaker count.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getSoftwareFormat
- System::setSpeakerMode
- System::init
- System::close
- Channel::setPan
- Channel::setSpeakerMix
- FMOD_SPEAKER
- FMOD_SPEAKERMODE
- FMOD_SOUND_FORMAT
- FMOD_DSP_RESAMPLER

System::setSpeakerMode

Sets the speaker mode in the hardware and FMOD software mixing engine.?

Syntax

```
  FMOD_RESULT System::setSpeakerMode(
    FMOD_SPEAKERMODE  speakermode
);
```

Parameters

speakermode

Speaker mode specified from the list in FMOD_SPEAKERMODE.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Speaker modes that are supported on each platform are as follows.

Win32 - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_QUAD, FMOD_SPEAKERMODE_SURROUND, FMOD_SPEAKERMODE_5POINT1, FMOD_SPEAKERMODE_7POINT1, FMOD_SPEAKERMODE_PROLOGIC.
Win64 - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_QUAD, FMOD_SPEAKERMODE_SURROUND, FMOD_SPEAKERMODE_5POINT1, FMOD_SPEAKERMODE_7POINT1, FMOD_SPEAKERMODE_PROLOGIC.
Linux - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_PROLOGIC.
Mac - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_PROLOGIC.
Xbox - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_5POINT1, FMOD_SPEAKERMODE_PROLOGIC.
PS2 - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_PROLOGIC.
GC - FMOD_SPEAKERMODE_RAW, FMOD_SPEAKERMODE_MONO, FMOD_SPEAKERMODE_STEREO, FMOD_SPEAKERMODE_PROLOGIC.

NOTE! If System::setSoftwareFormat is called after this function with a valid output channel count, the speakermode is set to FMOD_SPEAKERMODE_RAW.
If this function is called after System::setSoftwareFormat, then it will overwrite the channel count specified in that

function.
The channel count that is overwritten for each speaker mode is as follows.
- FMOD_SPEAKERMODE_RAW - Channel count is unaffected.
- FMOD_SPEAKERMODE_MONO - Channel count is set to 1.
- FMOD_SPEAKERMODE_STEREO - Channel count is set to 2.
- FMOD_SPEAKERMODE_QUAD - Channel count is set to 4.
- FMOD_SPEAKERMODE_SURROUND - Channel count is set to 5.
- FMOD_SPEAKERMODE_5POINT1 - Channel count is set to 6.
- FMOD_SPEAKERMODE_7POINT1 - Channel count is set to 8.
- FMOD_SPEAKERMODE_PROLOGIC - Channel count is set to 2.
These channel counts are the channel width of the FMOD DSP system, and affect software mixed sounds (sounds created with FMOD_SOFTWARE flag) only.
Hardware sounds are not affected, but will still have the speaker mode appropriately set if possible. (On Windows or Xbox the speaker mode is set by the user in the control panel / dashboard, not by FMOD).

Windows note! Sound will not behave correctly unless your control panel has set the speaker mode to the correct setup.
For example if FMOD_SPEAKERMODE_7POINT1 is set on a speaker system that has been set to 'stereo' in the windows control panel, sounds can dissapear and come out of the wrong speaker. Make sure your users know about this.
If using WinMM output, note that some soundcard drivers do not support multichannel output correctly (ie Creative cards). Other soundcards do.
Only DirectSound and ASIO have reliably working multichannel output.

To set the speaker mode to that of the windows control panel, use System::getDriverCaps.
For example

```
FMOD_SPEAKERMODE speakermode;
FMOD_RESULT      result;
result = system->getDriverCaps(0, 0, 0, 0, &speakermode);   // Get speakermode from default driver.
ERRCHECK(result);
result = system->setSpeakerMode(speakermode);
ERRCHECK(result);
```

Windows note! If the speakermode is not actually supported (ie even though the user set the speaker mode to 7.1 in windows, the soundcard might not be able to handle it), you will get FMOD_ERR_OUTPUT_CREATEBUFFER error under Windows. Change the speaker mode to FMOD_SPEAKERMODE_STEREO and re-initialize if this happens.

Calling this function resets any speaker positions set with System::set3DSpeakerPosition. This function must be called before calling System::set3DSpeakerPosition.

This function cannot be called after FMOD is already activated with System::init.
It must be called before System::init, or after System::close.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getSpeakerMode
- FMOD_SPEAKERMODE

- System::init
- System::close
- System::setSoftwareFormat
- System::set3DSpeakerPosition
- System::getDriverCaps
- FMOD_RESULT

System::setStreamBufferSize

Sets the internal buffersize for streams opened after this call.
?Larger values will consume more memory (see remarks), whereas smaller values may cause buffer under-run/starvation/stuttering caused by large delays in disk access (ie CDROM or netstream), or cpu usage in slow machines, or by trying to play too many streams at once.
?

Syntax

```
  FMOD_RESULT System::setStreamBufferSize(
  unsigned int  filebuffersize,
   FMOD_TIMEUNIT  filebuffersizetype
);
```

Parameters

filebuffersize

Size of stream file buffer. Default is 16384 (FMOD_TIMEUNIT_RAWBYTES).

filebuffersizetype

Type of unit for stream file buffer size. Must be FMOD_TIMEUNIT_MS, FMOD_TIMEUNIT_PCM, FMOD_TIMEUNIT_PCMBYTES or FMOD_TIMEUNIT_RAWBYTES. Default is FMOD_TIMEUNIT_RAWBYTES.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note this function does not affect streams created with FMOD_OPENUSER, as the buffer size is specified in System::createSound.
This function does not affect latency of playback. All streams are pre-buffered (unless opened with FMOD_OPENONLY), so they will always start immediately.
Seek and Play operations can sometimes cause a reflush of this buffer.

If FMOD_TIMEUNIT_RAWBYTES is used, the memory allocated is 2 * the size passed in, because fmod allocates a double buffer.
If FMOD_TIMEUNIT_MS, FMOD_TIMEUNIT_PCM or FMOD_TIMEUNIT_PCMBYTES is used, and the stream is infinite (such as a shoutcast netstream), then FMOD cannot calculate a compression ratio to work with when the file is opened. This means it will then base the buffersize on FMOD_TIMEUNIT_PCMBYTES, or in other words the number of PCM bytes, but this will be incorrect for compressed formats.
Use FMOD_TIMEUNIT_RAWBYTES for these type (infinite / undetermined length) of streams for more accurate read sizes.

Note to determine the actual memory usage of a stream, including sound buffer and other overhead, use Memory_GetStats before and after creating a sound.

Note that the stream may still stutter if the codec uses a large amount of cpu time, which impacts the smaller, internal 'decode' buffer.
The decode buffer size is changeable via FMOD_CREATESOUNDEXINFO.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- System::createSound
- System::getStreamBufferSize
- Sound::getOpenState
- Channel::setMute
- Memory_GetStats
- FMOD_CREATESOUNDEXINFO

System::setUserData

Sets a user value that the System object will store internally. Can be retrieved with System::getUserData.?

Syntax

```
  FMOD_RESULT System::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using System::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getUserData

System::unloadPlugin

Unloads a plugin from memory.?

Syntax

```
  FMOD_RESULT System::unloadPlugin(
   FMOD_PLUGINTYPE  plugintype,
  int  index
);
```

Parameters

plugintype

Specify the type of plugin type such as FMOD_PLUGINTYPE_OUTPUT, FMOD_PLUGINTYPE_CODEC or FMOD_PLUGINTYPE_DSP.

index

Index into the enumerated list of output plugins.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins

System::unlockDSP

Mutual exclusion function to unlock the FMOD DSP engine (which runs asynchronously in another thread) and let it continue executing.?

Syntax

```
FMOD_RESULT System::unlockDSP();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The DSP engine must be locked with System::lockDSP before this function is called.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::lockDSP

System::update

Updates the FMOD system. This should be called once per 'game' tick, or once per frame in your application.?

Syntax

```
FMOD_RESULT System::update();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This updates the following things.

- 3D Sound. 3D positioning will not update if this function is not called.
- Virtual voices. If more voices are played than there are real hardware/software voices, this function must be called to handle the virtualization.
- *_NRT output modes. This function must be called to drive the output for these output modes.
- FMOD_INIT_STREAM_FROM_UPDATE. This function must be called to update the streamer if this flag has been used.
- Callbacks. This function must be called to fire callbacks if they are specified.
- FMOD_NONBLOCKING. This function must be called to make sounds opened with FMOD_NONBLOCKING flag to work properly.

If FMOD_OUTPUTTYPE_NOSOUND_NRT or FMOD_OUTPUTTYPE_WAVWRITER_NRT output modes are used, this function also drives the software / DSP engine, instead of it running asynchronously in a thread as is the default behaviour.
This can be used for faster than realtime updates to the decoding or DSP engine which might be useful if the output is the wav writer for example.

If FMOD_INIT_STREAM_FROM_UPDATE is used, this function will update the stream engine. Combining this with the non realtime output will mean smoother captured output.

Warning! Do not be tempted to call this function from a different thread to other FMOD commands! This is dangerous and will cause corruption/crashes. This function is not thread safe, and should be called from the same thread as the rest of the FMOD commands.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- FMOD_INITFLAGS
- FMOD_OUTPUTTYPE
- FMOD_MODE

Sound Interface

Sound::addSyncPoint

Adds a sync point at a specific time within the sound. These points can be user generated or can come from a wav file with embedded markers.?

Syntax

```
FMOD_RESULT Sound::addSyncPoint(
  unsigned int  offset,
  FMOD_TIMEUNIT  offsettype,
  const char *  name,
  FMOD_SYNCPOINT **  point
);
```

Parameters

offset

offsettype

name

point

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In sound forge, a marker can be added a wave file by clicking on the timeline / ruler, and right clicking then selecting 'Insert Marker/Region'.
Riff wrapped mp3 files are also supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSyncPoints
- Sound::getSyncPoint
- Sound::getSyncPointInfo
- Sound::deleteSyncPoint

Sound::deleteSyncPoint

Deletes a syncpoint within the sound. These points can be user generated or can come from a wav file with embedded markers.?

Syntax

```
  FMOD_RESULT Sound::deleteSyncPoint(
    FMOD_SYNCPOINT *  point
);
```

Parameters

point

Address of an FMOD_SYNCPOINT object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In sound forge, a marker can be added a wave file by clicking on the timeline / ruler, and right clicking then selecting 'Insert Marker/Region'.
Riff wrapped mp3 files are also supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSyncPoints
- Sound::getSyncPoint
- Sound::getSyncPointInfo
- Sound::addSyncPoint

Sound::get3DConeSettings

Retrieves the inside and outside angles of the sound projection cone.?

Syntax

```
  FMOD_RESULT Sound::get3DConeSettings(
    float * insideconeangle,
    float * outsideconeangle,
    float * outsidevolume
);
```

Parameters

insideconeangle

Address of a variable that receives the inside angle of the sound projection cone, in degrees. This is the angle within which the sound is at its normal volume. Optional. Specify 0 or NULL to ignore.

outsideconeangle

Address of a variable that receives the outside angle of the sound projection cone, in degrees. This is the angle outside of which the sound is at its outside volume. Optional. Specify 0 or NULL to ignore.

outsidevolume

Address of a variable that receives the cone outside volume for this sound. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::set3DConeSettings
- Channel::set3DConeSettings

Sound::get3DCustomRolloff

Retrieves a pointer to the sound's current custom rolloff curve.?

Syntax

```
  FMOD_RESULT Sound::get3DCustomRolloff(
    FMOD_VECTOR **  points,
  int *  numpoints
);
```

Parameters

points

Address of a variable to receive the pointer to the current custom rolloff point list. Optional. Specify 0 or NULL to ignore.

numpoints

Address of a variable to receive the number of points int he current custom rolloff point list. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_VECTOR
- Sound::set3DCustomRolloff
- Channel::set3DCustomRolloff
- Channel::get3DCustomRolloff

Sound::get3DMinMaxDistance

Retrieve the minimum and maximum audible distance for a sound.?

Syntax

```
  FMOD_RESULT Sound::get3DMinMaxDistance(
    float * min,
    float * max
);
```

Parameters

min

Pointer to value to be filled with the minimum volume distance for the sound. See remarks for more on units. Optional. Specify 0 or NULL to ignore.

max

Pointer to value to be filled with the maximum volume distance for the sound. See remarks for more on units. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.
See System::set3DSettings for more on this.
The default units for minimum and maximum distances are 1.0 and 10,000.0f.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::set3DMinMaxDistance
- Channel::set3DMinMaxDistance
- Channel::get3DMinMaxDistance
- System::set3DSettings

Sound::getDefaults

Retrieves a sound's default attributes for when it is played on a channel with System::playSound.?

Syntax

```
  FMOD_RESULT Sound::getDefaults(
    float *  frequency,
    float *  volume,
    float *  pan,
  int *  priority
);
```

Parameters

frequency

Address of a variable that receives the default frequency for the sound. Optional. Specify 0 or NULL to ignore.

volume

Address of a variable that receives the default volume for the sound. Result will be from 0.0 to 1.0. 0.0 = Silent, 1.0 = full volume. Default = 1.0. Optional. Specify 0 or NULL to ignore.

pan

Address of a variable that receives the default pan for the sound. Result will be from -1.0 to +1.0. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0. Optional. Specify 0 or NULL to ignore.

priority

Address of a variable that receives the default priority for the sound when played on a channel. Result will be from 0 to 256. 0 = most important, 256 = least important. Default = 128. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setDefaults
- System::createSound

- System::playSound

Sound::getFormat

Returns format information about the sound.?

Syntax

```
  FMOD_RESULT Sound::getFormat(
    FMOD_SOUND_TYPE *  type,
    FMOD_SOUND_FORMAT *  format,
  int *  channels,
  int *  bits
);
```

Parameters

type

Address of a variable that receives the type of sound. Optional. Specify 0 or NULL to ignore.

format

Address of a variable that receives the format of the sound. Optional. Specify 0 or NULL to ignore.

channels

Address of a variable that receives the number of channels for the sound. Optional. Specify 0 or NULL to ignore.

bits

Address of a variable that receives the number of bits per sample for the sound. This corresponds to FMOD_SOUND_FORMAT but is provided as an integer format for convenience. Hardware compressed formats such as VAG, XADPCM, GCADPCM that stay compressed in memory will return 0. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_SOUND_TYPE
- FMOD_SOUND_FORMAT

Sound::getLength

Retrieves the length of the sound using the specified time unit.?

Syntax

```
  FMOD_RESULT Sound::getLength(
  unsigned int * length,
   FMOD_TIMEUNIT lengthtype
);
```

Parameters

length

Address of a variable that receives the length of the sound.

lengthtype

Time unit retrieve into the length parameter. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Certain timeunits do not work depending on the file format. For example FMOD_TIMEUNIT_MODORDER will not work with an mp3 file.
A length of 0xFFFFFFFF usually means it is of unlimited length, such as an internet radio stream or MOD/S3M/XM/IT file which may loop forever.

Warning! Using a VBR source that does not have an associated length information in milliseconds or pcm samples (such as MP3 or MOD/S3M/XM/IT) may return inaccurate lengths specify FMOD_TIMEUNIT_MS or FMOD_TIMEUNIT_PCM.
If you want FMOD to retrieve an accurate length it will have to pre-scan the file first in this case. You will have to specify FMOD_ACCURATETIME when loading or opening the sound. This means there is a slight delay as FMOD scans the whole file when loading the sound to find the right length in millseconds or pcm samples, and this also creates a seek table as it does this for seeking purposes.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT

Sound::getLoopCount

Retrieves the current loop count value for the specified sound.?

Syntax

```
  FMOD_RESULT Sound::getLoopCount(
  int * loopcount
);
```

Parameters

loopcount

Address of a variable that receives the number of times a sound will loop by default before stopping. 0 = oneshot. 1 = loop once then stop. -1 = loop forever. Default = -1

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Unlike the channel loop count function, this function simply returns the value set with Sound::setLoopCount. It does not decrement as it plays (especially seeing as one sound can be played multiple times).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setLoopCount

Sound::getLoopPoints

Retrieves the loop points for a sound.?

Syntax

```
  FMOD_RESULT Sound::getLoopPoints(
  unsigned int *  loopstart,
   FMOD_TIMEUNIT  loopstarttype,
  unsigned int *  loopend,
   FMOD_TIMEUNIT  loopendtype
);
```

Parameters

loopstart

Address of a variable to receive the loop start point. This point in time is played, so it is inclusive. Optional. Specify 0 or NULL to ignore.

loopstarttype

The time format used for the returned loop start point. See FMOD_TIMEUNIT.

loopend

Address of a variable to receive the loop end point. This point in time is played, so it is inclusive. Optional. Specify 0 or NULL to ignore.

loopendtype

The time format used for the returned loop end point. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- Sound::setLoopPoints

Sound::getMode

Retrieves the mode bits set by the codec and the user when opening the sound.?

Syntax

```
  FMOD_RESULT Sound::getMode(
    FMOD_MODE *  mode
);
```

Parameters

mode

Address of a variable that receives the current mode for this sound.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setMode
- System::createSound
- Channel::setMode
- Channel::getMode

Sound::getName

Retrieves the name of a sound.?

Syntax

```
  FMOD_RESULT Sound::getName(
  char * name,
  int namelen
);
```

Parameters

name

Address of a variable that receives the name of the sound.

namelen

Length in bytes of the target buffer to receieve the string.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

if FMOD_LOWMEM has been specified in System::createSound, this function will return "(null)".

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound
- FMOD_MODE

Sound::getNumSubSounds

Retrieves the number of subsounds stored within a sound.?

Syntax

```
  FMOD_RESULT Sound::getNumSubSounds(
  int *  numsubsounds
);
```

Parameters

numsubsounds

Address of a variable that receives the number of subsounds stored within this sound.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A format that has subsounds is usually a container format, such as FSB, DLS, MOD, S3M, XM, IT.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getSubSound

Sound::getNumSyncPoints

Retrieves the number of sync points stored within a sound. These points can be user generated or can come from a wav file with embedded markers.?

Syntax

```
  FMOD_RESULT Sound::getNumSyncPoints(
  int *  numsyncpoints
);
```

Parameters

numsyncpoints

Address of a variable to receive the number of sync points within this sound.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In sound forge, a marker can be added a wave file by clicking on the timeline / ruler, and right clicking then selecting 'Insert Marker/Region'.
Riff wrapped mp3 files are also supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getSyncPoint
- Sound::getSyncPointInfo
- Sound::addSyncPoint
- Sound::deleteSyncPoint

Sound::getNumTags

Retrieves the number of tags belonging to a sound.?

Syntax

```
  FMOD_RESULT Sound::getNumTags(
  int *  numtags,
  int *  numtagsupdated
);
```

Parameters

numtags

Address of a variable that receives the number of tags in the sound. Optional. Specify 0 or NULL to ignore.

numtagsupdated

Address of a variable that receives the number of tags updated since this function was last called. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 'numtagsupdated' parameter can be used to check if any tags have been updated since last calling this function. This can be useful to update tag fields, for example from internet based streams, such as shoutcast or icecast where the name of the song might change.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getTag

Sound::getOpenState

Retrieves the state a sound is in after FMOD_NONBLOCKING has been used to open it, or the state of the streaming buffer.?

Syntax

```
  FMOD_RESULT Sound::getOpenState(
    FMOD_OPENSTATE * openstate,
  unsigned int * percentbuffered,
    bool * starving
);
```

Parameters

openstate

Address of a variable that receives the open state of a sound. Optional. Specify 0 or NULL to ignore.

percentbuffered

Address of a variable that receives the percentage of the file buffer filled progress of a stream. Optional. Specify 0 or NULL to ignore.

starving

Address of a variable that receives the starving state of a sound. If a stream has decoded more than the stream file buffer has ready for it, it will return TRUE. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
Note: The return value will be the result of the asynchronous sound create. Use this to determine what happened if a sound failed to open.
Note: Always check 'openstate' to determine the state of the sound. Do not assume that if this function returns FMOD_OK then the sound has finished loading.

Remarks

When a sound is opened with FMOD_NONBLOCKING, it is opened and prepared in the background, or asynchronously.
This allows the main application to execute without stalling on audio loads.
This function will describe the state of the asynchronous load routine i.e. whether it has succeeded, failed or is still in progress.

If 'starving' is true, then you will most likely hear a stuttering/repeating sound as the decode buffer loops on itself and replays old data.
Now that this variable exists, you can detect buffer underrun and use something like Channel::setMute to keep it quiet

until it is not starving any more.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_OPENSTATE
- FMOD_MODE
- Channel::setMute

Sound::getSoundGroup

Retrieves the sound's current soundgroup.?

Syntax

```
  FMOD_RESULT Sound::getSoundGroup(
    FMOD::SoundGroup ** soundgroup
);
```

Parameters

soundgroup

Address of a pointer to a SoundGroup to receive the sound's current soundgroup.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
By default a sound is located in the 'master sound group'. This can be retrieved with System::getMasterSoundGroup.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setSoundGroup
- System::getMasterSoundGroup

Sound::getSubSound

Retrieves a handle to a Sound object that is contained within the parent sound.?

Syntax

```
  FMOD_RESULT Sound::getSubSound(
  int  index,
   FMOD::Sound **  subsound
);
```

Parameters

index

Index of the subsound to retrieve within this sound.

subsound

Address of a variable that receives the sound object specified.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If the sound is a stream and FMOD_NONBLOCKING was not used, then this call will perform a blocking seek/flush to the specified subsound.

If FMOD_NONBLOCKING was used to open this sound and the sound is a stream, FMOD will do a non blocking seek/flush and set the state of the subsound to FMOD_OPENSTATE_SEEKING.
The sound won't be ready to be used in this case until the state of the sound becomes FMOD_OPENSTATE_READY (or FMOD_OPENSTATE_ERROR).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSubSounds
- Sound::setSubSound
- System::createSound

- FMOD_MODE
- FMOD_OPENSTATE

Sound::getSyncPoint

Retrieve a handle to a sync point. These points can be user generated or can come from a wav file with embedded markers.?

Syntax

```
  FMOD_RESULT Sound::getSyncPoint(
  int  index,
    FMOD_SYNCPOINT **  point
);
```

Parameters

index

Index of the sync point to retrieve. Use Sound::getNumSyncPoints to determine the number of syncpoints.

point

Address of a variable to receive a pointer to a sync point.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In sound forge, a marker can be added a wave file by clicking on the timeline / ruler, and right clicking then selecting 'Insert Marker/Region'.
Riff wrapped mp3 files are also supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSyncPoints
- Sound::getSyncPointInfo
- Sound::addSyncPoint
- Sound::deleteSyncPoint

Sound::getSyncPointInfo

Retrieves information on an embedded sync point. These points can be user generated or can come from a wav file with embedded markers.?

Syntax

```
FMOD_RESULT Sound::getSyncPointInfo(
  FMOD_SYNCPOINT *  point,
  char *  name,
  int  namelen,
  unsigned int *  offset,
  FMOD_TIMEUNIT  offsettype
);
```

Parameters

point

Pointer to a sync point. Use Sound::getSyncPoint to retrieve a syncpoint or Sound::addSyncPoint to create one.

name

Address of a variable to receive the name of the syncpoint. Optional. Specify 0 or NULL to ignore.

namelen

Size of buffer in bytes for name parameter. FMOD will only copy to this point if the string is bigger than the buffer passed in. Specify 0 to ignore name parameter.

offset

Address of a variable to receceive the offset of the syncpoint in a format determined by the offsettype parameter. Optional. Specify 0 or NULL to ignore.

offsettype

A timeunit parameter to determine a desired format for the offset parameter. For example the offset can be specified as pcm samples, or milliseconds.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

In sound forge, a marker can be added a wave file by clicking on the timeline / ruler, and right clicking then selecting 'Insert Marker/Region'.

Riff wrapped mp3 files are also supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSyncPoints
- Sound::getSyncPoint
- Sound::addSyncPoint
- Sound::deleteSyncPoint

Sound::getSystemObject

Retrieves the parent System object that was used to create this object.?

Syntax

```
  FMOD_RESULT Sound::getSystemObject(
    FMOD::System ** system
);
```

Parameters

system

Address of a pointer that receives the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound

Sound::getTag

Retrieves a descriptive tag stored by the sound, to describe things like the song name, author etc.?

Syntax

```
  FMOD_RESULT Sound::getTag(
  const char *  name,
  int  index,
   FMOD_TAG *  tag
);
```

Parameters

name

Optional. Name of a tag to retrieve. Used to specify a particular tag if the user requires it. To get all types of tags leave this parameter as 0 or NULL.

index

Index into the tag list. If the name parameter is null, then the index is the index into all tags present, from 0 up to but not including the numtags value returned by Sound::getNumTags.
If name is not null, then index is the index from 0 up to the number of tags with the same name. For example if there were 2 tags with the name "TITLE" then you could use 0 and 1 to reference them.
Specifying an index of -1 returns new or updated tags. This can be used to pull tags out as they are added or updated.

tag

Pointer to a tag structure. This will receive

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The number of tags available can be found with Sound::getNumTags. The way to display or retrieve tags can be done in 3 different ways.
All tags can be continuously retrieved by looping from 0 to the numtags value in Sound::getNumTags - 1. Updated tags will refresh automatically, and the 'updated' member of the FMOD_TAG structure will be set to true if a tag has been updated, due to something like a netstream changing the song name for example.
Tags could also be retrieved by specifying -1 as the index and only updating tags that are returned. If all tags are retrieved and this function is called the function will return an error of FMOD_ERR_TAGNOTFOUND.
Specific tags can be retrieved by specifying a name parameter. The index can be 0 based or -1 in the same fashion as described previously.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumTags
- FMOD_TAG

Sound::getUserData

Retrieves the user value that that was set by calling the Sound::setUserData function.?

Syntax

```
  FMOD_RESULT Sound::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the Sound::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setUserData

Sound::getVariations

Retrieves the current playback behaviour variations of a sound.?

Syntax

```
  FMOD_RESULT Sound::getVariations(
    float *  frequencyvar,
    float *  volumevar,
    float *  panvar
);
```

Parameters

frequencyvar

Address of a variable to receive the frequency variation in hz. Frequency will play at its default frequency, plus or minus a random value within this range. Default = 0.0. Specify 0 or NULL to ignore.

volumevar

Address of a variable to receive the volume variation. 0.0 to 1.0. Sound will play at its default volume, plus or minus a random value within this range. Default = 0.0. Specify 0 or NULL to ignore.

panvar

Address of a variable to receive the pan variation. 0.0 to 2.0. Sound will play at its default pan, plus or minus a random value within this range. Pan is from -1.0 to +1.0 normally so the range can be a maximum of 2.0 in this case. Default = 0. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::setVariations

Sound::lock

Returns a pointer to the beginning of the sample data for a sound.
?

Syntax

```
FMOD_RESULT Sound::lock(
  unsigned int offset,
  unsigned int length,
  void ** ptr1,
  void ** ptr2,
  unsigned int * len1,
  unsigned int * len2
);
```

Parameters

offset

Offset in *bytes* to the position you want to lock in the sample buffer.

length

Number of *bytes* you want to lock in the sample buffer.

ptr1

Address of a pointer that will point to the first part of the locked data.

ptr2

Address of a pointer that will point to the second part of the locked data. This will be null if the data locked hasn't wrapped at the end of the buffer.

len1

Length of data in *bytes* that was locked for ptr1

len2

Length of data in *bytes* that was locked for ptr2. This will be 0 if the data locked hasn't wrapped at the end of the buffer.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

You must always unlock the data again after you have finished with it, using Sound::unlock.
With this function you get access to the RAW audio data, for example 8, 16, 24 or 32bit PCM data, mono or stereo data, and on consoles, vag, xadpcm or gcadpcm compressed data. You must take this into consideration when processing the data within the pointer.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::unlock
- System::createSound

Sound::readData

Reads data from an opened sound to a specified pointer, using the FMOD codec created internally.
?This can be used for decoding data offline in small pieces (or big pieces), rather than playing and capturing it, or loading the whole file at once and having to lock / unlock the data.?

Syntax

```
  FMOD_RESULT Sound::readData(
   void * buffer,
  unsigned int lenbytes,
  unsigned int * read
);
```

Parameters

buffer

Address of a buffer that receives the decoded data from the sound.

lenbytes

Number of bytes to read into the buffer.

read

Number of bytes actually read.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If too much data is read, it is possible FMOD_ERR_FILE_EOF will be returned, meaning it is out of data. The 'read' parameter will reflect this by returning a smaller number of bytes read than was requested.
As a sound already reads the whole file then closes it upon calling System::createSound (unless System::createStream or FMOD_CREATESTREAM is used), this function will not work because the file is no longer open.
Note that opening a stream makes it read a chunk of data and this will advance the read cursor. You need to either use FMOD_OPENONLY to stop the stream pre-buffering or call Sound::seekData to reset the read cursor.
If FMOD_OPENONLY flag is used when opening a sound, it will leave the file handle open, and FMOD will not read any data internally, so the read cursor will be at position 0. This will allow the user to read the data from the start.
As noted previously, if a sound is opened as a stream and this function is called to read some data, then you will 'miss the start' of the sound.
Channel::setPosition will have the same result. These function will flush the stream buffer and read in a chunk of audio internally. This is why if you want to read from an absolute position you should use Sound::seekData and not the previously mentioned functions.

Remember if you are calling readData and seekData on a stream it is up to you to cope with the side effects that may occur. Information functions such as Channel::getPosition may give misleading results. Calling Channel::setPosition will reset and flush the stream, leading to the time values returning to their correct position.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::seekData
- FMOD_MODE
- Channel::setPosition
- System::createSound
- System::createStream

Sound::release

Frees a sound object.?

Syntax

```
FMOD_RESULT Sound::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This will free the sound object and everything created under it.

If this is a stream that is playing as a subsound of another parent stream, then if this is the currently playing subsound (be it a normal subsound playback, or as part of a sentence), the whole stream will stop.
Note - This function will block if it was opened with FMOD_NONBLOCKING and hasn't finished opening yet.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound
- Sound::getSubSound

Sound::seekData

Seeks a sound for use with data reading. This is not a function to 'seek a sound' for normal use. This is for use in conjunction with Sound::readData.?

Syntax

```
  FMOD_RESULT Sound::seekData(
  unsigned int  pcm
);
```

Parameters

pcm

Offset to seek to in PCM samples.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note. If a stream is opened and this function is called to read some data, then it will advance the internal file pointer, so data will be skipped if you play the stream. Also calling position / time information functions will lead to misleading results.
A stream can be reset before playing by setting the position of the channel (ie using Channel::setPosition), which will make it seek, reset and flush the stream buffer. This will make it sound correct again.
Remember if you are calling readData and seekData on a stream it is up to you to cope with the side effects that may occur.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::readData
- Channel::setPosition

Sound::set3DConeSettings

Sets the inside and outside angles of the sound projection cone, as well as the volume of the sound outside the outside angle of the sound projection cone.?

Syntax

```
FMOD_RESULT Sound::set3DConeSettings(
  float insideconeangle,
  float outsideconeangle,
  float outsidevolume
);
```

Parameters

insideconeangle

Inside cone angle, in degrees, from 0 to 360. This is the angle within which the sound is at its normal volume. Must not be greater than outsideconeangle. Default = 360.

outsideconeangle

Outside cone angle, in degrees, from 0 to 360. This is the angle outside of which the sound is at its outside volume. Must not be less than insideconeangle. Default = 360.

outsidevolume

Cone outside volume, from 0 to 1.0. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::get3DConeSettings
- Channel::set3DConeSettings

Sound::set3DCustomRolloff

Point a sound to use a custom rolloff curve. Must be used in conjunction with FMOD_3D_CUSTOMROLLOFF flag to be activated.?

Syntax

```
FMOD_RESULT Sound::set3DCustomRolloff(
  FMOD_VECTOR *  points,
  int  numpoints
);
```

Parameters

points

An array of FMOD_VECTOR structures where x = distance and y = volume from 0.0 to 1.0. z should be set to 0.

numpoints

The number of points in the array.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! This function does not duplicate the memory for the points internally. The pointer you pass to FMOD must remain valid until there is no more use for it.
Do not free the memory while in use, or use a local variable that goes out of scope while in use.

Points must be sorted by distance! Passing an unsorted list to FMOD will result in an error.

Set the points parameter to 0 or NULL to disable the points. If FMOD_3D_CUSTOMROLLOFF is set and the rolloff curve is 0, FMOD will revert to logarithmic curve rolloff.

Min and maxdistance are meaningless when FMOD_3D_CUSTOMROLLOFF is used and the values are ignored.

Here is an example of a custom array of points.

```
FMOD_VECTOR curve[3] =
{


      { 0.0f, 1.0f, 0.0f },
{  2.0f,  0.2f, 0.0f },
{  20.0f, 0.0f, 0.0f }
};
```
x represents the distance, y represents the volume. z is always 0.
Distances between points are linearly interpolated.
Note that after the highest distance specified, the volume in the last entry is used from that distance onwards.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- FMOD_VECTOR
- Sound::get3DCustomRolloff
- Channel::set3DCustomRolloff
- Channel::get3DCustomRolloff

Sound::set3DMinMaxDistance

Sets the minimum and maximum audible distance for a sound.
?
?MinDistance is the minimum distance that the sound emitter will cease to continue growing louder at (as it approaches the listener).
?Within the mindistance it stays at the constant loudest volume possible. Outside of this mindistance it begins to attenuate.
?MaxDistance is the distance a sound stops attenuating at. Beyond this point it will stay at the volume it would be at maxdistance units from the listener and will not attenuate any more.
?MinDistance is useful to give the impression that the sound is loud or soft in 3d space. An example of this is a small quiet object, such as a bumblebee, which you could set a mindistance of to 0.1 for example, which would cause it to attenuate quickly and dissapear when only a few meters away from the listener.
?Another example is a jumbo jet, which you could set to a mindistance of 100.0, which would keep the sound volume at max until the listener was 100 meters away, then it would be hundreds of meters more before it would fade out.
?
?In summary, increase the mindistance of a sound to make it 'louder' in a 3d world, and decrease it to make it 'quieter' in a 3d world.
?Maxdistance is effectively obsolete unless you need the sound to stop fading out at a certain point. Do not adjust this from the default if you dont need to.
?Some people have the confusion that maxdistance is the point the sound will fade out to, this is not the case.
?

Syntax

```
FMOD_RESULT Sound::set3DMinMaxDistance(
  float  min,
  float  max
);
```

Parameters

min

The sound's minimum volume distance in "units". See remarks for more on units.

max

The sound's maximum volume distance in "units". See remarks for more on units.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.
See System::set3DSettings for more on this.

The default units for minimum and maximum distances are 1.0 and 10,000.0f.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::get3DMinMaxDistance
- Channel::set3DMinMaxDistance
- Channel::get3DMinMaxDistance
- System::set3DSettings

Sound::setDefaults

Sets a sounds's default attributes, so when it is played it uses these values without having to specify them later for each channel each time the sound is played.?

Syntax

```
FMOD_RESULT Sound::setDefaults(
  float  frequency,
  float  volume,
  float  pan,
  int  priority
);
```

Parameters

frequency

Default playback frequency for the sound, in hz. (ie 44100hz).

volume

Default volume for the sound. 0.0 to 1.0. 0.0 = Silent, 1.0 = full volume. Default = 1.0.

pan

Default pan for the sound. -1.0 to +1.0. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0.

priority

Default priority for the sound when played on a channel. 0 to 256. 0 = most important, 256 = least important. Default = 128.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

There are no 'ignore' values for these parameters. Use Sound::getDefaults if you want to change only 1 and leave others unaltered.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getDefaults
- System::playSound
- System::createSound

Sound::setLoopCount

Sets a sound, by default, to loop a specified number of times before stopping if its mode is set to FMOD_LOOP_NORMAL or FMOD_LOOP_BIDI.?

Syntax

```
  FMOD_RESULT Sound::setLoopCount(
  int  loopcount
);
```

Parameters

loopcount

Number of times to loop before stopping. 0 = oneshot. 1 = loop once then stop. -1 = loop forever. Default = -1

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not affect FMOD_HARDWARE based sounds that are not streamable.
FMOD_SOFTWARE based sounds or any type of sound created with System::CreateStream or FMOD_CREATESTREAM will support this function.

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM). When changing the loop count, sounds created with System::createStream or FMOD_CREATESTREAM may already have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size. Otherwise you will not normally encounter any problems.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getLoopCount

- System::setStreamBufferSize
- FMOD_CREATESOUNDEXINFO

Sound::setLoopPoints

Sets the loop points within a sound.?

Syntax

```
FMOD_RESULT Sound::setLoopPoints(
  unsigned int loopstart,
   FMOD_TIMEUNIT loopstarttype,
  unsigned int loopend,
   FMOD_TIMEUNIT loopendtype
);
```

Parameters

loopstart

The loop start point. This point in time is played, so it is inclusive.

loopstarttype

The time format used for the loop start point. See FMOD_TIMEUNIT.

loopend

The loop end point. This point in time is played, so it is inclusive.

loopendtype

The time format used for the loop end point. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Not supported by static sounds created with FMOD_HARDWARE.
Supported by sounds created with FMOD_SOFTWARE, or sounds of any type (hardware or software) created with System::createStream or FMOD_CREATESTREAM.
If a sound was 1000ms long and you wanted to loop the whole sound, loopstart would be 0, and loopend would be 999,
not 1000.
If loop end is smaller or equal to loop start, it will result in an error.
If loop start or loop end is larger than the length of the sound, it will result in an error.

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM).
When changing the loop points, sounds created with System::createStream or FMOD_CREATESTREAM may

already have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size. Otherwise you will not normally encounter any problems.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- FMOD_MODE
- Sound::getLoopPoints
- Sound::setLoopCount
- System::createStream
- System::setStreamBufferSize
- Channel::setPosition
- FMOD_CREATESOUNDEXINFO

Sound::setMode

Sets or alters the mode of a sound.?

Syntax

```
  FMOD_RESULT Sound::setMode(
    FMOD_MODE  mode
);
```

Parameters

mode

Mode bits to set.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When calling this function, note that it will only take effect when the sound is played again with System::playSound. Consider this mode the 'default mode' for when the sound plays, not a mode that will suddenly change all currently playing instances of this sound.
Flags supported:
FMOD_LOOP_OFF
FMOD_LOOP_NORMAL
FMOD_LOOP_BIDI (only works with sounds created with FMOD_SOFTWARE. Otherwise it will behave as FMOD_LOOP_NORMAL)
FMOD_3D_HEADRELATIVE
FMOD_3D_WORLDRELATIVE
FMOD_2D (see notes for win32 hardware voices)
FMOD_3D (see notes for win32 hardware voices)
FMOD_3D_LOGROLLOFF
FMOD_3D_LINEARROLLOFF
FMOD_3D_CUSTOMROLLOFF
FMOD_3D_IGNOREGEOMETRY
FMOD_DONTRESTOREVIRTUAL

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM). When changing the loop mode, sounds created with System::createStream or FMOD_CREATESTREAM may already have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size.

Otherwise you will not normally encounter any problems.

Win32 FMOD_HARDWARE note. Under DirectSound, you cannot change the mode of a sound between FMOD_2D and FMOD_3D. If this is a problem create the sound as FMOD_3D initially, and use FMOD_3D_HEADRELATIVE and FMOD_3D_WORLDRELATIVE. Alternatively just use FMOD_SOFTWARE.

If FMOD_3D_IGNOREGEOMETRY is not specified, the flag will be cleared if it was specified previously.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- Sound::getMode
- System::setStreamBufferSize
- System::playSound
- System::createStream
- Channel::setPosition
- FMOD_CREATESOUNDEXINFO

Sound::setSoundGroup

Moves the sound from its existing SoundGroup to the specified sound group.?

Syntax

```
  FMOD_RESULT Sound::setSoundGroup(
    FMOD::SoundGroup * soundgroup
);
```

Parameters

soundgroup

Address of a SoundGroup object to move the sound to.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default a sound is located in the 'master sound group'. This can be retrieved with System::getMasterSoundGroup. Putting a sound in a sound group (or just using the master sound group) allows for functionality like limiting a group of sounds to a certain number of playbacks (see SoundGroup::setMaxAudible).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getSoundGroup
- System::getMasterSoundGroup
- System::createSoundGroup
- SoundGroup::setMaxAudible

Sound::setSubSound

Assigns a sound as a 'subsound' of another sound. A sound can contain other sounds. The sound object that is issuing the command will be the 'parent' sound.?

Syntax

```
 FMOD_RESULT Sound::setSubSound(
 int index,
  FMOD::Sound * subsound
);
```

Parameters

index

Index within the sound to set the new sound to as a 'subsound'.

subsound

Sound object to set as a subsound within this sound.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getNumSubSounds
- Sound::getSubSound

Sound::setSubSoundSentence

For any sound that has subsounds, this function will determine the order of playback of these subsounds, and it will play / stitch together the subsounds without gaps.
?This is a very useful feature for those users wanting to do seamless / gapless stream playback. (ie sports commentary, gapless playback media players etc).?

Syntax

```
  FMOD_RESULT Sound::setSubSoundSentence(
  int *  subsoundlist,
  int  numsubsounds
);
```

Parameters

subsoundlist

Pointer to an array of indicies which are the subsounds to play. One subsound can be included in this list multiple times if required.

numsubsounds

Number of indicies inside the subsoundlist array.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! Only streams can be sentenced. Static samples are not stitchable because most hardware api's don't have a way to gaplessly play 2 sounds after one another.

By default subsounds are stitched automatically from index 0 to the last index. For example a CD that is opened as a sound (and the cd tracks are its subsounds) will play all CD tracks from start to end without gaps if the parent sound is played with System::playSound.
A user can swap subsounds that arent playing at the time to do dynamic stitching/sentencing of sounds.

The currently playing subsound in a sentence can be found with Channel::getPosition and the timeunit FMOD_TIMEUNIT_SENTENCE_SUBSOUND. This is useful for displaying the currently playing track of a cd in a whole CD sentence for example.
For realtime stitching purposes, it is better to know the buffered ahead of time subsound index. This can be done by adding the flag (using bitwise OR) FMOD_TIMEUNIT_BUFFERED.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- Sound::getSubSound
- Channel::getPosition
- FMOD_TIMEUNIT

Sound::setUserData

Sets a user value that the Sound object will store internally. Can be retrieved with Sound::getUserData.?

Syntax

```
  FMOD_RESULT Sound::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the Sound object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using Sound::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getUserData

Sound::setVariations

Changes the playback behaviour of a sound by allowing random variations to playback parameters to be set.?

Syntax

```
 FMOD_RESULT Sound::setVariations(
  float  frequencyvar,
  float  volumevar,
  float  panvar
);
```

Parameters

frequencyvar

Frequency variation in hz. Frequency will play at its default frequency, plus or minus a random value within this range. Default = 0.0.

volumevar

Volume variation. 0.0 to 1.0. Sound will play at its default volume, plus or minus a random value within this range. Default = 0.0.

panvar

Pan variation. 0.0 to 2.0. Sound will play at its default pan, plus or minus a random value within this range. Pan is from -1.0 to +1.0 normally so the range can be a maximum of 2.0 in this case. Default = 0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getVariations

Sound::unlock

Releases previous sample data lock from Sound::lock.?

Syntax

```
  FMOD_RESULT Sound::unlock(
    void *  ptr1,
    void *  ptr2,
  unsigned int  len1,
  unsigned int  len2
);
```

Parameters

ptr1

Pointer to the 1st locked portion of sample data, from Sound::lock.

ptr2

Pointer to the 2nd locked portion of sample data, from Sound::lock.

len1

Length of data in *bytes* that was locked for ptr1

len2

Length of data in *bytes* that was locked for ptr2. This will be 0 if the data locked hasn't wrapped at the end of the buffer.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
Call this function after data has been read/written to from Sound::lock. This function will do any post processing necessary and if needed, send it to sound ram.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::lock
- System::createSound

Version 4.12.03 Built on Feb 18, 2008

Channel Interface

Channel::setDelay
Channel::setFrequency
Channel::setInputChannelMix
Channel::setLoopCount
Channel::setLoopPoints
Channel::setMode
Channel::setMute
Channel::setPan
Channel::setPaused
Channel::setPosition
Channel::setPriority
Channel::setReverbProperties
Channel::setSpeakerLevels
Channel::setSpeakerMix
Channel::setUserData
Channel::setVolume
Channel::stop

Firelight Technologies FMOD Ex

Channel::addDSP

This function adds a pre-created DSP unit or effect to the head of the Channel DSP chain.?

Syntax

```
  FMOD_RESULT Channel::addDSP(
    FMOD::DSP *  dsp
);
```

Parameters

dsp

A pointer to a pre-created DSP unit to be inserted at the head of the Channel DSP chain.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is a wrapper function to insert a DSP unit at the top of the Channel DSP chain. It disconnects the head unit from its input, then inserts the unit at the head and reconnects the previously disconnected input back as as an input to the new unit. It is effectively the following code.

```
int numinputs;
channel->getDSPHead(?
 dsphead->getNumInputs(?
if (numinputs > 1)
{
        return FMOD_ERR_DSP_TOOMANYCONNECTIONS;

}
 dsphead->getInput(0, ?
 dsphead->disconnectFrom(next);
 dsphead->addInput(dsp);
 dsp->addInput(next);
 dsp->setActive(true);
```

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getDSPHead
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- System::addDSP
- ChannelGroup::addDSP
- DSP::remove

Channel::get3DAttributes

Retrieves the position and velocity of a 3d channel.?

Syntax

```
  FMOD_RESULT Channel::get3DAttributes(
    FMOD_VECTOR *  pos,
    FMOD_VECTOR *  vel
);
```

Parameters

pos

Address of a variable that receives the position in 3D space of the channel. Optional. Specify 0 or NULL to ignore.

vel

Address of a variable that receives the velocity in 'distance units per second' in 3D space of the channel. See remarks. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DAttributes
- FMOD_VECTOR
- System::set3DSettings

Channel::get3DConeOrientation

Retrieves the orientation of the sound projection cone for this channel.?

Syntax

```
  FMOD_RESULT Channel::get3DConeOrientation(
    FMOD_VECTOR * orientation
);
```

Parameters

orientation

Address of a variable that receives the orientation of the sound projection cone. The vector information represents the center of the sound cone.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DConeOrientation

Channel::get3DConeSettings

Retrieves the inside and outside angles of the sound projection cone.?

Syntax

```
 FMOD_RESULT Channel::get3DConeSettings(
  float * insideconeangle,
  float * outsideconeangle,
  float * outsidevolume
);
```

Parameters

insideconeangle

Address of a variable that receives the inside angle of the sound projection cone, in degrees. This is the angle within which the sound is at its normal volume. Optional. Specify 0 or NULL to ignore.

outsideconeangle

Address of a variable that receives the outside angle of the sound projection cone, in degrees. This is the angle outside of which the sound is at its outside volume. Optional. Specify 0 or NULL to ignore.

outsidevolume

Address of a variable that receives the cone outside volume for this channel. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DConeSettings
- Sound::get3DConeSettings

Channel::get3DCustomRolloff

Retrieves a pointer to the sound's current custom rolloff curve.?

Syntax

```
  FMOD_RESULT Channel::get3DCustomRolloff(
   FMOD_VECTOR **  points,
  int *  numpoints
);
```

Parameters

points

Address of a variable to receive the pointer to the current custom rolloff point list. Optional. Specify 0 or NULL to ignore.

numpoints

Address of a variable to receive the number of points int he current custom rolloff point list. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_VECTOR
- Channel::set3DCustomRolloff
- Sound::set3DCustomRolloff
- Sound::get3DCustomRolloff

Channel::get3DDopplerLevel

Retrieves the current 3D doppler level for the channel set by Channel::set3DDopplerLevel.?

Syntax

```
  FMOD_RESULT Channel::get3DDopplerLevel(
    float * level
);
```

Parameters

level

Address of a variable to receives the current doppler scale for this channel. 0 = No doppler. 1 = Normal doppler. 5 = max. Default = 1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DDopplerLevel

Channel::get3DMinMaxDistance

Retrieves the current minimum and maximum audible distance for a channel.?

Syntax

```
  FMOD_RESULT Channel::get3DMinMaxDistance(
    float *  mindistance,
    float *  maxdistance
);
```

Parameters

mindistance

Pointer to a floating point value to store mindistance. Optional. Specify 0 or NULL to ignore.

maxdistance

Pointer to a floating point value to store maxdistance. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DMinMaxDistance
- System::set3DSettings
- Sound::set3DMinMaxDistance

Channel::get3DOcclusion

Retrieves the the EAX or software based occlusion factors for a channel.?

Syntax

```
  FMOD_RESULT Channel::get3DOcclusion(
    float *  directocclusion,
    float *  reverbocclusion
);
```

Parameters

directocclusion

Address of a variable that receives the occlusion factor for a voice for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

reverbocclusion

Address of a variable that receives the occlusion factor for a voice for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DOcclusion
- ChannelGroup::get3DOcclusion

Channel::get3DPanLevel

Retrieves the current 3D mix level for the channel set by Channel::set3DPanLevel.?

Syntax

```
  FMOD_RESULT Channel::get3DPanLevel(
    float * level
);
```

Parameters

level

0 = Sound pans according to Channel::setSpeakerMix. 1 = Sound pans according to 3d position. Default = 1 (all by 3d position).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DPanLevel
- Channel::setSpeakerMix

Channel::get3DSpread

Retrieves the stereo (and above) spread angle specified by Channel::set3DSpread.?

Syntax

```
  FMOD_RESULT Channel::get3DSpread(
    float * angle
);
```

Parameters

angle

Address of a variable that receives the spread angle for subchannels. 0 = all subchannels are located at the same position. 360 = all subchannels are located at the opposite position.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DSpread

Channel::getAudibility

Returns the combined volume of the channel after 3d sound, volume, channel group volume and geometry occlusion calculations have been performed on it.?

Syntax

```
FMOD_RESULT Channel::getAudibility(
  float * audibility
);
```

Parameters

audibility

Address of a variable that receives the channel audibility value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This does not represent the waveform, just the calculated volume based on 3d distance, occlusion, volume and channel group volume. This value is used by the FMOD Ex virtual channel system to order its channels between real and virtual.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setVolume
- Channel::getVolume
- ChannelGroup::setVolume
- ChannelGroup::getVolume
- Channel::set3DOcclusion
- Channel::get3DOcclusion
- Channel::set3DAttributes
- Channel::get3DAttributes

Channel::getChannelGroup

Retrieves the currently assigned channel group for the channel.?

Syntax

```
  FMOD_RESULT Channel::getChannelGroup(
    FMOD::ChannelGroup ** channelgroup
);
```

Parameters

channelgroup

Address of a variable to receive a pointer to the currently assigned channel group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setChannelGroup

Channel::getCurrentSound

Returns the currently playing sound for this channel.?

Syntax

```
  FMOD_RESULT Channel::getCurrentSound(
    FMOD::Sound ** sound
);
```

Parameters

sound

Address of a variable that receives the pointer to the currently playing sound for this channel.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a sound is not playing the returned pointer will be 0 or NULL.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::playDSP

Channel::getDSPHead

Returns a pointer to the DSP unit head node that handles software mixing for this channel.
?Only applicable to channels playing sounds created with FMOD_SOFTWARE.?

Syntax

```
FMOD_RESULT Channel::getDSPHead(
  FMOD::DSP ** dsp
);
```

Parameters

dsp

Address of a variable that receives pointer to the current head DSP unit for this channel.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default a channel DSP unit usually contains 1 input, which is the wavetable input.
If System::playDSP has been used then the input to the channel head unit will be the unit that was specified in the call.
See the tutorials for more information on DSP networks and how to manipulate them.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createDSP
- System::createDSPByType
- System::playDSP

Channel::getDelay

Sets a delay before the sound is audible and after the sound ends.?

Syntax

```
  FMOD_RESULT Channel::getDelay(
  unsigned int * startdelay,
  unsigned int * enddelay
);
```

Parameters

startdelay

Address of a variable that receives the current channel delay in milliseconds for before the sound starts. Optional. Specify 0 or NULL to ignore.

enddelay

Address of a variable that receives the current channel delay in milliseconds for for after the sound stops. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setDelay

Channel::getFrequency

Returns the frequency in HZ of the channel.?

Syntax

```
  FMOD_RESULT Channel::getFrequency(
    float *  frequency
);
```

Parameters

frequency

Address of a variable that receives the current frequency of the channel in HZ.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setFrequency

Channel::getIndex

Retrieves the internal channel index for a channel.?

Syntax

```
  FMOD_RESULT Channel::getIndex(
  int * index
);
```

Parameters

index

Address of a variable to receive the channel index. This will be from 0 to the value specified in System::init minus 1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that working with channel indicies directly is not recommended. It is recommended that you use FMOD_CHANNEL_FREE for the index in System::playSound to use FMOD's channel manager.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::init

Channel::getInputChannelMix

Retrieves the incoming levels for a channel in a sound.
?A mono sound has 1 input channel, a stereo has 2, etc. It depends on what type of sound is playing on the channel at the time.?

Syntax

```
  FMOD_RESULT Channel::getInputChannelMix(
   float *  levels,
  int  numlevels
);
```

Parameters

levels

Address of an array of float volume levels, from 0.0 to 1.0. These represent the incoming channels for the sound playing on the channel at the time.

numlevels

Number of floats to receive into the array. Maximum = the maximum number of input channels specified in System::setSoftwareFormat.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This does not affect which speakers the sound is routed to. This can be used in conjunction with functions like Channel::setPan, Channel::setSpeakerMix, Channel::setSpeakerLevels.
This function only scales the input channels from the sound.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setInputChannelMix
- Channel::setPan
- Channel::setSpeakerMix

- Channel::setSpeakerLevels
- System::setSoftwareFormat

Channel::getLoopCount

Retrieves the current loop count for the specified channel.?

Syntax

```
  FMOD_RESULT Channel::getLoopCount(
  int *  loopcount
);
```

Parameters

loopcount

Address of a variable that receives the number of times a channel will loop before stopping. 0 = oneshot. 1 = loop once then stop. -1 = loop forever. Default = -1

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function retrieves the **current** loop countdown value for the channel being played.
This means it will decrement until reaching 0, as it plays. To reset the value, use Channel::setLoopCount.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setLoopCount

Channel::getLoopPoints

Retrieves the loop points for a channel.?

Syntax

```
  FMOD_RESULT Channel::getLoopPoints(
  unsigned int *  loopstart,
   FMOD_TIMEUNIT  loopstarttype,
  unsigned int *  loopend,
   FMOD_TIMEUNIT  loopendtype
);
```

Parameters

loopstart

Address of a variable to receive the loop start point. This point in time is played, so it is inclusive. Optional. Specify 0 or NULL to ignore.

loopstarttype

The time format used for the returned loop start point. See FMOD_TIMEUNIT.

loopend

Address of a variable to receive the loop end point. This point in time is played, so it is inclusive. Optional. Specify 0 or NULL to ignore.

loopendtype

The time format used for the returned loop end point. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- Channel::setLoopPoints

Channel::getMode

Retrieves the current mode bit flags for the current channel.?

Syntax

```
  FMOD_RESULT Channel::getMode(
    FMOD_MODE * mode
);
```

Parameters

mode

Address of a an FMOD_MODE variable that receives the current mode for this channel.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setMode

Channel::getMute

Returns the current mute status of the channel.?

Syntax

```
  FMOD_RESULT Channel::getMute(
    bool * mute
);
```

Parameters

mute

true = channel is muted (silent), false = channel is at normal volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setMute

Channel::getPan

Returns the pan position of the channel.?

Syntax

```
  FMOD_RESULT Channel::getPan(
    float *  pan
);
```

Parameters

pan

Address of a variable to receive the left/right pan level for the channel, from -1.0 to 1.0 inclusive. -1.0 = Full left, 1.0 = full right. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setPan

Channel::getPaused

Retrieves the paused state of the channel.?

Syntax

```
  FMOD_RESULT Channel::getPaused(
    bool *  paused
);
```

Parameters

paused

Address of a variable that receives the current paused state. true = the sound is paused. false = the sound is not paused.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setPaused

Channel::getPosition

Returns the current PCM offset or playback position for the specified channel.?

Syntax

```
  FMOD_RESULT Channel::getPosition(
  unsigned int *  position,
   FMOD_TIMEUNIT  postype
);
```

Parameters

position

Address of a variable that receives the position of the sound.

postype

Time unit to retrieve into the position parameter. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Certain timeunits do not work depending on the file format. For example FMOD_TIMEUNIT_MODORDER will not work with an mp3 file.
A PCM sample is a unit of measurement in audio that contains the data for one audible element of sound. 1 sample might be 16bit stereo, so 1 sample contains 4 bytes. 44,100 samples of a 44khz sound would represent 1 second of data.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setPosition
- FMOD_TIMEUNIT
- Sound::getLength

Channel::getPriority

Retrieves the current priority for this channel.?

Syntax

```
  FMOD_RESULT Channel::getPriority(
  int *  priority
);
```

Parameters

priority

Address of a variable that receives the current channel priority. 0 to 256 inclusive. 0 = most important. 256 = least important. Default = 128.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setPriority

Channel::getReverbProperties

Retrieves the current reverb properties for this channel.?

Syntax

```
  FMOD_RESULT Channel::getReverbProperties(
    FMOD_REVERB_CHANNELPROPERTIES *  prop
);
```

Parameters

prop

Address of a variable to receive the FMOD_REVERB_CHANNELPROPERTIES information.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setReverbProperties
- FMOD_REVERB_CHANNELPROPERTIES

Channel::getSpeakerLevels

Retrieves the current level settings from Channel::setSpeakerLevels.?

Syntax

```
  FMOD_RESULT Channel::getSpeakerLevels(
   FMOD_SPEAKER speaker,
   float * levels,
  int numlevels
);
```

Parameters

speaker

The speaker id to get the levels for. This can be cast to an integer if you are using a device with more than the pre-defined speaker range.

levels

Address of a variable that receives the current levels for the channel. This is an array of floating point values. The destination array size can be specified with the numlevels parameter.

numlevels

Number of floats in the destination array.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not return level values reflecting Channel::setPan or Channel::setVolume.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setSpeakerLevels
- Channel::setPan

- Channel::setVolume

Channel::getSpeakerMix

Sets the channel's speaker volume levels for each speaker individually.?

Syntax

```
  FMOD_RESULT Channel::getSpeakerMix(
    float *  frontleft,
    float *  frontright,
    float * center,
    float *  lfe,
    float *  backleft,
    float *  backright,
    float * sideleft,
    float * sideright
);
```

Parameters

frontleft

Address of a variable to receive the current volume level for this channel in the front left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

frontright

Address of a variable to receive the current volume level for this channel in the front right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

center

Address of a variable to receive the current volume level for this channel in the center speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

lfe

Address of a variable to receive the current volume level for this channel in the subwoofer speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

backleft

Address of a variable to receive the current volume level for this channel in the back left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

backright

Address of a variable to receive the current volume level for this channel in the back right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

sideleft

Address of a variable to receive the current volume level for this channel in the side left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

sideright

Address of a variable to receive the current volume level for this channel in the side right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume, up to 5.0 = 5x amplification.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

For 3D sound, the values set here are not representative of the 3d mix. For 3D sound this function is mainly for retrieving the LFE value if it was set by the user.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setSpeakerMix

Channel::getSpectrum

Retrieves the spectrum from the currently playing output signal for the current channel only.?

Syntax

```
  FMOD_RESULT Channel::getSpectrum(
    float * spectrumarray,
  int  numvalues,
  int channeloffset,
    FMOD_DSP_FFT_WINDOW  windowtype
);
```

Parameters

spectrumarray

Address of a variable that receives the spectrum data. This is an array of floating point values. Data will range is 0.0 to 1.0. Decibels = 10.0f * (float)log10(val) * 2.0f; See remarks for what the data represents.

numvalues

Size of array in floating point values being passed to the function. Must be a power of 2. (ie 128/256/512 etc). Min = 64. Max = 8192.

channeloffset

Channel of the signal to analyze. If the signal is multichannel (such as a stereo output), then this value represents which channel to analyze. On a stereo signal 0 = left, 1 = right.

windowtype

"Pre-FFT" window method. This filters the PCM data before entering the spectrum analyzer to reduce transient frequency error for more accurate results. See FMOD_DSP_FFT_WINDOW for different types of fft window techniques possible and for a more detailed explanation.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The larger the numvalues, the more CPU the FFT will take. Choose the right value to trade off between accuracy / speed.
The larger the numvalues, the more 'lag' the spectrum will seem to inherit. This is because the FFT window size stretches the analysis back in time to what was already played. For example if the numvalues size happened to be 44100 and the output rate was 44100 it would be analyzing the past second of data, and giving you the average spectrum over that time period.

If you are not displaying the result in dB, then the data may seem smaller than it should be. To display it you may want to normalize the data - that is, find the maximum value in the resulting spectrum, and scale all values in the array by 1 / max. (ie if the max was 0.5f, then it would become 1).
To get the spectrum for both channels of a stereo signal, call this function twice, once with channeloffset = 0, and again with channeloffset = 1. Then add the spectrums together and divide by 2 to get the average spectrum for both channels.

<u>What the data represents.</u>
To work out what each entry in the array represents, use this formula

```
entry_hz = (output_rate / 2) / numvalues
```

The array represents amplitudes of each frequency band from 0hz to the nyquist rate. The nyquist rate is equal to the output rate divided by 2.
For example when FMOD is set to 44100hz output, the range of represented frequencies will be 0hz to 22049hz, a total of 22050hz represented.
If in the same example, 1024 was passed to this function as the numvalues, each entry's contribution would be as follows.

```
entry_hz = (44100 / 2) / 1024
entry_hz = 21.53 hz
```

Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_FFT_WINDOW
- System::getSpectrum
- ChannelGroup::getSpectrum
- System::getWaveData

Channel::getSystemObject

Retrieves the parent System object that was used to create this object.?

Syntax

```
  FMOD_RESULT Channel::getSystemObject(
   FMOD::System ** system
);
```

Parameters

system

Address of a variable that receives the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound

Channel::getUserData

Retrieves the user value that that was set by calling the Channel::setUserData function.?

Syntax

```
  FMOD_RESULT Channel::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the Channel::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

NOTE: If this channel was spawned by the event system then its user data field will be set, by the event system, to the event instance handle that spawned it. Use this function to go from an arbitrary channel back up to the event that owns it.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setUserData

Channel::getVolume

Retrieves the volume level for the channel.?

Syntax

```
  FMOD_RESULT Channel::getVolume(
    float *  volume
);
```

Parameters

volume

Address of a variable to receive the channel volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setVolume

Channel::getWaveData

Retrieves a pointer to a block of PCM data that represents the currently playing waveform on this channel. ?This function is useful for a very easy way to plot an oscilliscope.?

Syntax

```
  FMOD_RESULT Channel::getWaveData(
   float * wavearray,
  int  numvalues,
  int channeloffset
);
```

Parameters

wavearray

Address of a variable that receives the currently playing waveform data. This is an array of floating point values.

numvalues

Number of floats to write to the array. Maximum value = 16384.

channeloffset

Offset into multichannel data. Mono channels use 0. Stereo channels use 0 = left, 1 = right. More than stereo use the appropriate index.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This is the actual resampled pcm data window at the time the function is called.

Do not use this function to try and display the whole waveform of the sound, as this is more of a 'snapshot' of the current waveform at the time it is called, and could return the same data if it is called very quickly in succession. See the DSP API to capture a continual stream of wave data as it plays, or see Sound::lock / Sound::unlock if you want to simply display the waveform of a sound.

This function allows retrieval of left and right data for a stereo sound individually. To combine them into one signal, simply add the entries of each seperate buffer together and then divide them by 2.
Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getSpectrum
- ChannelGroup::getWaveData
- System::getWaveData
- Sound::lock
- Sound::unlock

Channel::isPlaying

Returns the playing state for the current channel.?

Syntax

```
  FMOD_RESULT Channel::isPlaying(
    bool * isplaying
);
```

Parameters

isplaying

Address of a variable that receives the current channel's playing status. true = the channel is currently playing a sound. false = the channel is not playing a sound.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::playDSP

Channel::isVirtual

Returns the current channel's status of whether it is virtual (emulated) or not due to FMOD Ex's virtual channel management system.?

Syntax

```
  FMOD_RESULT Channel::isVirtual(
   bool * isvirtual
);
```

Parameters

isvirtual

Address of a variable that receives the current channel's virtual status. true = the channel is inaudible and currently being emulated at no cpu cost. false = the channel is a real hardware or software voice and should be audible.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Virtual channels are not audible, because there are no more real hardware or software channels available.
If you are plotting virtual voices vs real voices graphically, and wondering why FMOD sometimes chooses seemingly random channels to be virtual that are usually far away, that is because they are probably silent. It doesn't matter which are virtual and which are not if they are silent. Virtual voices are not calculation on 'closest to listener' calculation, they are based on audibility. See the tutorial in the FMOD Ex documentation for more information on virtual channels.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound

Channel::set3DAttributes

Sets the position and velocity of a 3d channel.?

Syntax

```
  FMOD_RESULT Channel::set3DAttributes(
  const FMOD_VECTOR *  pos,
  const FMOD_VECTOR *  vel
);
```

Parameters

pos

Position in 3D space of the channel. Specifying 0 / null will ignore this parameter.

vel

Velocity in 'distance units per second' in 3D space of the channel. See remarks. Specifying 0 / null will ignore this parameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.

For a stereo 3d sound, you can set the spread of the left/right parts in speaker space by using Channel::set3DSpread.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DAttributes
- FMOD_VECTOR
- System::set3DSettings
- Channel::set3DSpread

Channel::set3DConeOrientation

Sets the orientation of the sound projection cone.?

Syntax

```
  FMOD_RESULT Channel::set3DConeOrientation(
    FMOD_VECTOR * orientation
);
```

Parameters

orientation

Pointer to an FMOD_VECTOR defining the coordinates of the sound cone orientation vector.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function has no effect unless the cone angle and cone outside volume have also been set to values other than the default.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DConeOrientation
- Channel::set3DConeSettings
- Sound::set3DConeSettings
- FMOD_VECTOR

Channel::set3DConeSettings

Sets the inside and outside angles of the sound projection cone, as well as the volume of the sound outside the outside angle of the sound projection cone.?

Syntax

```
FMOD_RESULT Channel::set3DConeSettings(
  float insideconeangle,
  float outsideconeangle,
  float outsidevolume
);
```

Parameters

insideconeangle

Inside cone angle, in degrees. This is the angle within which the sound is at its normal volume. Must not be greater than outsideconeangle. Default = 360.

outsideconeangle

Outside cone angle, in degrees. This is the angle outside of which the sound is at its outside volume. Must not be less than insideconeangle. Default = 360.

outsidevolume

Cone outside volume, from 0 to 1.0. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DConeSettings
- Channel::set3DConeOrientation
- Sound::set3DConeSettings

Channel::set3DCustomRolloff

Point a channel to use a custom rolloff curve. Must be used in conjunction with FMOD_3D_CUSTOMROLLOFF flag to be activated.?

Syntax

```
  FMOD_RESULT Channel::set3DCustomRolloff(
   FMOD_VECTOR *  points,
  int  numpoints
);
```

Parameters

points

An array of FMOD_VECTOR structures where x = distance and y = volume from 0.0 to 1.0. z should be set to 0.

numpoints

The number of points in the array.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! This function does not duplicate the memory for the points internally. The pointer you pass to FMOD must remain valid until there is no more use for it.
Do not free the memory while in use, or use a local variable that goes out of scope while in use.

Points must be sorted by distance! Passing an unsorted list to FMOD will result in an error.

Set the points parameter to 0 or NULL to disable the points. If FMOD_3D_CUSTOMROLLOFF is set and the rolloff curve is 0, FMOD will revert to logarithmic curve rolloff.

Min and maxdistance are meaningless when FMOD_3D_CUSTOMROLLOFF is used and the values are ignored.

Here is an example of a custom array of points.

```
 FMOD_VECTOR curve[3]  =
{


      {  0.0f,  1.0f,  0.0f} ,
{   2.0f,   0.2f,  0.0f} ,
{   20.0f,  0.0f,  0.0f}
};
```
x represents the distance, y represents the volume. z is always 0.
Distances between points are linearly interpolated.
Note that after the highest distance specified, the volume in the last entry is used from that distance onwards.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- FMOD_VECTOR
- Channel::get3DCustomRolloff
- Sound::set3DCustomRolloff
- Sound::get3DCustomRolloff

Channel::set3DDopplerLevel

Sets the channel specific doppler scale for the channel.?

Syntax

```
  FMOD_RESULT Channel::set3DDopplerLevel(
    float  level
);
```

Parameters

level

0 = No doppler. 1 = Normal doppler. 5 = max. Default = 1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DDopplerLevel

Channel::set3DMinMaxDistance

Sets the minimum and maximum audible distance for a channel.
?

Syntax

```
FMOD_RESULT Channel::set3DMinMaxDistance(
  float mindistance,
  float maxdistance
);
```

Parameters

mindistance

The channel's minimum volume distance in "units". See remarks for more on units.

maxdistance

The channel's maximum volume distance in "units". See remarks for more on units.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

MinDistance is the minimum distance that the sound emitter will cease to continue growing louder at (as it approaches the listener).
Within the mindistance it stays at the constant loudest volume possible. Outside of this mindistance it begins to attenuate.
MaxDistance is the distance a sound stops attenuating at. Beyond this point it will stay at the volume it would be at maxdistance units from the listener and will not attenuate any more.
MinDistance is useful to give the impression that the sound is loud or soft in 3d space. An example of this is a small quiet object, such as a bumblebee, which you could set a mindistance of to 0.1 for example, which would cause it to attenuate quickly and dissapear when only a few meters away from the listener.
Another example is a jumbo jet, which you could set to a mindistance of 100.0, which would keep the sound volume at max until the listener was 100 meters away, then it would be hundreds of meters more before it would fade out.

In summary, increase the mindistance of a sound to make it 'louder' in a 3d world, and decrease it to make it 'quieter' in a 3d world.
maxdistance is effectively obsolete unless you need the sound to stop fading out at a certain point. Do not adjust this from the default if you dont need to.
Some people have the confusion that maxdistance is the point the sound will fade out to, this is not the case.

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.

The default units for minimum and maximum distances are 1.0 and 10000.0f.
Volume drops off at mindistance / distance.
To define the min and max distance per sound and not per channel use Sound::set3DMinMaxDistance.

If FMOD_3D_CUSTOMROLLOFF is used, then these values are stored, but ignored in 3d processing.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DMinMaxDistance
- System::set3DSettings
- Sound::set3DMinMaxDistance

Channel::set3DOcclusion

Sets the EAX or software based occlusion factors for a channel. If the FMOD geometry engine is not being used, this function can be called to produce the same audible effects, just without the built in polygon processing. FMOD's internal geometry engine calls this function.?

Syntax

```
  FMOD_RESULT Channel::set3DOcclusion(
    float  directocclusion,
    float  reverbocclusion
);
```

Parameters

directocclusion

Occlusion factor for a voice for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

reverbocclusion

Occlusion factor for a voice for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

With EAX based sound cards and FMOD_HARDWARE based sounds, this will attenuate the sound using frequency filtering.
With non EAX sounds, then the volume is simply attenuated by the directOcclusion factor.
If FMOD_INIT_OCCLUSION_LOWPASS is specified, FMOD_SOFTWARE based sounds will also use frequency filtering, with a small CPU hit.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DOcclusion
- ChannelGroup::set3DOcclusion

Channel::set3DPanLevel

Sets how much the 3d engine has an effect on the channel, versus that set by Channel::setPan, Channel::setSpeakerMix, Channel::setSpeakerLevels.
?

Syntax

```
  FMOD_RESULT Channel::set3DPanLevel(
    float  level
);
```

Parameters

level

1 = Sound pans and attenuates according to 3d position. 0 = Attenuation is ignored and pan/speaker levels are defined by Channel::setPan, Channel::setSpeakerMix, Channel::setSpeakerLevels. Default = 1 (all by 3D position).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Only affects sounds created with FMOD_SOFTWARE and FMOD_3D.

Useful for morhping a sound between 3D and 2D. This is most common in volumetric sound, when the sound goes from directional, to 'all around you' (and doesn't pan according to listener position/direction).
FMOD_INIT_SOFTWARE_HRTF is also interpolated to be 'off' if level = 0, so that you do not get a muffling effect based on location when the sound is supposed to be effectively 2D.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DPanLevel
- Channel::setSpeakerMix
- Channel::setPan
- Channel::setSpeakerLevels

Channel::set3DSpread

Sets the spread of a 3d stereo or multichannel sound in speaker space.
?Normally a 3d sound is aimed at one position in a speaker array depending on the 3d position, to give it direction. Left and right parts of a stereo sound for example are consequently summed together and become 'mono'.
?When increasing the 'spread' of a sound, the left and right parts of a stereo sound rotate away from their original position, to give it more 'stereoness'. The rotation of the sound channels are done in 'speaker space'.?

Syntax

```
  FMOD_RESULT Channel::set3DSpread(
    float angle
);
```

Parameters

angle

Spread angle for stereo sounds and above. 0 = all sound channels are located at the same speaker location and is 'mono'. 360 = all subchannels are located at the opposite speaker location to the speaker location that it should be according to 3D position. Default = 0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Only affects sounds created with FMOD_SOFTWARE.

By default, if a stereo sound was played in 3d, and it was directly in front of you, the left and right part of the stereo sound would be summed into the center speaker (on a 5.1 setup), making it sound mono.
This function lets you control the speaker spread of a stereo (and above) sound within the speaker array, to separate the left right part of a stereo sound for example.
In the above case, in a 5.1 setup, specifying a spread of 90 degrees would put the left part of the sound in the front left speaker, and the right part of the sound in the front right speaker. This stereo separation remains in tact as the listener rotates and the sound moves around the speakers.
To summarize (for a stereo sound).
1. A spread angle of 0 makes the stereo sound mono at the point of the 3d emitter.
2. A spread angle of 90 makes the left part of the stereo sound place itself at 45 degrees to the left and the right part 45 degrees to the right.
3. A spread angle of 180 makes the left part of the stero sound place itself at 90 degrees to the left and the right part 90 degrees to the right.
4. A spread angle of 360 makes the stereo sound mono at the opposite speaker location to where the 3d emitter should be located (by moving the left part 180 degrees left and the right part 180 degrees right). So in this case, behind you when the sound should be in front of you!

Multichannel sounds with channel counts greater than stereo have their sub-channels spread evently through the specified angle. For example a 6 channel sound over a 90 degree spread has each subchannel located 15 degrees apart from each other in the speaker array.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::get3DSpread
- FMOD_MODE

Channel::setCallback

Sets a callback for a channel for a specific event.?

Syntax

```
FMOD_RESULT Channel::setCallback(
  FMOD_CHANNEL_CALLBACKTYPE  type,
  FMOD_CHANNEL_CALLBACK  callback,
  int  command
);
```

Parameters

type

The callback type, for example an 'end of sound' callback.

callback

Pointer to a callback to receive the event when it happens.

command

The callback parameter. This has a different meaning for each type of callback type.
FMOD_CHANNEL_CALLBACK_END - This parameter has no effect.
FMOD_CHANNEL_CALLBACK_VIRTUALVOICE - This parameter has no effect.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Currently callbacks are driven by System::update and will only occur when this function is called. This has the main advantage of far less complication due to thread issues, and allows all FMOD commands, including loading sounds and playing new sounds from the callback.
It also allows any type of sound to have an end callback, no matter what it is. The only disadvantage is that callbacks are not asynchronous and are bound by the latency caused by the rate the user calls the update command.
Callbacks are stdcall. Use F_CALLBACK inbetween your return type and function name.
Example:

```
 FMOD_RESULT F_CALLBACK mycallback(FMOD_CHANNEL *channel, FMOD_CHANNEL_CALLBACKTYPE type,
int command, unsigned int commanddata1, unsigned int commanddata2)
{


    FMOD::Channel *cppchannel = (FMOD::Channel *)channel;
```

```
        // More code goes here.

        return FMOD_OK;

}
```

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::update
- FMOD_CHANNEL_CALLBACK
- FMOD_CHANNEL_CALLBACKTYPE

Channel::setChannelGroup

Sets a channel to belong to a specified channel group. A channelgroup can contain many channels.
?

Syntax

```
  FMOD_RESULT Channel::setChannelGroup(
    FMOD::ChannelGroup * channelgroup
);
```

Parameters

channelgroup

Pointer to a ChannelGroup object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Setting a channel to a channel group removes it from any previous group, it does not allow sharing of channel groups.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getChannelGroup

Channel::setDelay

Sets a delay before the sound is audible and after the sound ends.?

Syntax

```
  FMOD_RESULT Channel::setDelay(
  unsigned int startdelay,
  unsigned int enddelay
);
```

Parameters

startdelay

The delay in milliseconds before the sound starts. Currently not implemented yet.

enddelay

The delay in milliseconds after the sound stops before the channel actually stops processing. Channel::isPlaying will remain true until this delay has passed even though the sound itself has stopped playing.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Setting a delay after a sound ends is sometimes useful to prolong the sound, even though it has stopped, so that DSP effects can trail out, or render the last of their tails. (for example an echo or reverb effect).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getDelay
- Channel::isPlaying

Channel::setFrequency

Sets the channel's frequency or playback rate, in HZ.?

Syntax

```
  FMOD_RESULT Channel::setFrequency(
    float  frequency
);
```

Parameters

frequency

A frequency value in HZ. This value can also be negative to play the sound backwards (negative frequencies allowed with FMOD_SOFTWARE based non-stream sounds only). DirectSound hardware voices have limited frequency range on some soundcards. Please see remarks for more on this.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a sound is played, it plays at the default frequency of the sound which can be set by Sound::setDefaults.
For most file formats, the volume is determined by the audio format.

Frequency limitations for sounds created with FMOD_HARDWARE in DirectSound.
Every hardware device has a minimum and maximum frequency. This means setting the frequency above the maximum and below the minimum will have no effect.
FMOD clamps frequencies to these values when playing back on hardware, so if you are setting the frequency outside of this range, the frequency will stay at either the minimum or maximum.
Note that FMOD_SOFTWARE based sounds do not have this limitation.
To find out the minimum and maximum value before initializing FMOD (maybe to decide whether to use a different soundcard, output mode, or drop back fully to software mixing), you can use the System::getDriverCaps function.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getFrequency

- System::getDriverCaps
- Sound::setDefaults

Channel::setInputChannelMix

Sets the incoming levels in a sound. This means if you have a multichannel sound you can turn channels on and off. ?A mono sound has 1 input channel, a stereo has 2, etc. It depends on what type of sound is playing on the channel at the time.?

Syntax

```
  FMOD_RESULT Channel::setInputChannelMix(
   float * levels,
  int  numlevels
);
```

Parameters

levels

Array of float volume levels, from 0.0 to 1.0. These represent the incoming channels for the sound playing on the channel at the time.

numlevels

Number of floats in the array. Maximum = the maximum number of input channels specified in System::setSoftwareFormat.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getInputChannelMix
- Channel::setPan
- Channel::setSpeakerMix
- Channel::setSpeakerLevels
- System::setSoftwareFormat

Channel::setLoopCount

Sets a channel to loop a specified number of times before stopping.?

Syntax

```
  FMOD_RESULT Channel::setLoopCount(
  int  loopcount
);
```

Parameters

loopcount

Number of times to loop before stopping. 0 = oneshot. 1 = loop once then stop. -1 = loop forever. Default = -1

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not affect FMOD_HARDWARE based sounds that are not streamable.
FMOD_SOFTWARE based sounds or any type of sound created with System::CreateStream or FMOD_CREATESTREAM will support this function.

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM). When changing the loop count, sounds created with System::createStream or FMOD_CREATESTREAM may already have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size. Otherwise you will not normally encounter any problems.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getLoopCount

- Channel::setPosition
- System::createStream
- FMOD_CREATESOUNDEXINFO
- FMOD_MODE

Channel::setLoopPoints

Sets the loop points within a channel.?

Syntax

```
  FMOD_RESULT Channel::setLoopPoints(
  unsigned int  loopstart,
   FMOD_TIMEUNIT  loopstarttype,
  unsigned int  loopend,
   FMOD_TIMEUNIT  loopendtype
);
```

Parameters

loopstart

The loop start point. This point in time is played, so it is inclusive.

loopstarttype

The time format used for the loop start point. See FMOD_TIMEUNIT.

loopend

The loop end point. This point in time is played, so it is inclusive.

loopendtype

The time format used for the loop end point. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Not supported by static sounds created with FMOD_HARDWARE.
Supported by sounds created with FMOD_SOFTWARE, or sounds of any type (hardware or software) created with System::createStream or FMOD_CREATESTREAM.
If a sound was 1000ms long and you wanted to loop the whole sound, loopstart would be 0, and loopend would be 999,
not 1000.
If loop end is smaller or equal to loop start, it will result in an error.
If loop start or loop end is larger than the length of the sound, it will result in an error.

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM). When changing the loop points, sounds created with System::createStream or FMOD_CREATESTREAM may already

have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size. Otherwise you will not normally encounter any problems.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_TIMEUNIT
- FMOD_MODE
- Channel::getLoopPoints
- Channel::setLoopCount
- System::createStream
- System::setStreamBufferSize
- FMOD_CREATESOUNDEXINFO

Channel::setMode

Changes some attributes for a channel based on the mode passed in.?

Syntax

```
  FMOD_RESULT Channel::setMode(
    FMOD_MODE  mode
);
```

Parameters

mode

Mode bits to set.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Flags supported:
FMOD_LOOP_OFF
FMOD_LOOP_NORMAL
FMOD_LOOP_BIDI (only works with sounds created with FMOD_SOFTWARE. Otherwise it will behave as FMOD_LOOP_NORMAL)
FMOD_3D_HEADRELATIVE
FMOD_3D_WORLDRELATIVE
FMOD_2D (see notes for win32 hardware voices)
FMOD_3D (see notes for win32 hardware voices)
FMOD_3D_LOGROLLOFF
FMOD_3D_LINEARROLLOFF
FMOD_3D_CUSTOMROLLOFF
FMOD_3D_IGNOREGEOMETRY
FMOD_DONTRESTOREVIRTUAL

Issues with streamed audio. (Sounds created with with System::createStream or FMOD_CREATESTREAM). When changing the loop mode, sounds created with System::createStream or FMOD_CREATESTREAM may already have been pre-buffered and executed their loop logic ahead of time, before this call was even made.
This is dependant on the size of the sound versus the size of the stream *decode* buffer. See FMOD_CREATESOUNDEXINFO.
If this happens, you may need to reflush the stream buffer. To do this, you can call Channel::setPosition which forces a reflush of the stream buffer.
Note this will usually only happen if you have sounds or looppoints that are smaller than the stream decode buffer size. Otherwise you will not normally encounter any problems.

Win32 FMOD_HARDWARE note. Under DirectSound, you cannot change the loop mode of a channel while it is

playing. You must use Sound::setMode or pause the channel to get this to work.
Win32 FMOD_HARDWARE note. Under DirectSound, you cannot change the mode of a channel between FMOD_2D and FMOD_3D. If this is a problem create the sound as FMOD_3D initially, and use FMOD_3D_HEADRELATIVE and FMOD_3D_WORLDRELATIVE. Alternatively just use FMOD_SOFTWARE.

If FMOD_3D_IGNOREGEOMETRY is not specified, the flag will be cleared if it was specified previously.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MODE
- Channel::getMode
- Channel::setPosition
- Sound::setMode
- System::createStream
- System::setStreamBufferSize
- FMOD_CREATESOUNDEXINFO

Channel::setMute

Mutes / un-mutes a channel, effectively silencing it or returning it to its normal volume.?

Syntax

```
  FMOD_RESULT Channel::setMute(
    bool  mute
);
```

Parameters

mute

true = channel becomes muted (silent), false = channel returns to normal volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a channel belongs to a muted channelgroup, it will stay muted regardless of the channel mute state. The channel mute state will still be reflected internally though, ie Channel::getMute will still return the value you set. If the channelgroup has mute set to false, this function will become effective again.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getMute
- ChannelGroup::setMute

Channel::setPan

Sets a channels pan position linearly.?

Syntax

```
  FMOD_RESULT Channel::setPan(
    float  pan
);
```

Parameters

pan

A left/right pan level, from -1.0 to 1.0 inclusive. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function only works on sounds created with FMOD_2D. 3D sounds are not pannable and will return FMOD_ERR_NEEDS2D.

Only sounds that are mono or stereo can be panned. Multichannel sounds (ie >2 channels) cannot be panned.
Mono sounds are panned from left to right using constant power panning (non linear fade). This means when pan = 0.0, the balance for the sound in each speaker is 71% left and 71% right, not 50% left and 50% right. This gives (audibly) smoother pans.
Stereo sounds heave each left/right value faded up and down according to the specified pan position. This means when pan = 0.0, the balance for the sound in each speaker is 100% left and 100% right. When pan = -1.0, only the left channel of the stereo sound is audible, when pan = 1.0, only the right channel of the stereo sound is audible.

Panning does not work if the speaker mode is FMOD_SPEAKERMODE_RAW.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getPan
- FMOD_SPEAKERMODE

Version 4.12.03 Built on Feb 18, 2008

Channel::setPaused

Sets the paused state of the channel.?

Syntax

```
  FMOD_RESULT Channel::setPaused(
   bool  paused
);
```

Parameters

paused

Paused state to set. true = channel is paused. false = channel is unpaused.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a channel belongs to a paused channelgroup, it will stay paused regardless of the channel pause state. The channel pause state will still be reflected internally though, ie Channel::getPaused will still return the value you set. If the channelgroup has paused set to false, this function will become effective again.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getPaused
- ChannelGroup::setPaused

Channel::setPosition

Sets the current playback position for the currently playing sound to the specified PCM offset.?

Syntax

```
FMOD_RESULT Channel::setPosition(
  unsigned int  position,
  FMOD_TIMEUNIT  postype
);
```

Parameters

position

Position of the channel to set in units specified in the postype parameter.

postype

Time unit to set the channel position by. See FMOD_TIMEUNIT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Certain timeunits do not work depending on the file format. For example FMOD_TIMEUNIT_MODORDER will not work with an mp3 file.

Note that if you are calling this function on a stream, it has to possibly reflush its buffer to get zero latency playback when it resumes playing, therefore it could potentially cause a stall or take a small amount of time to do this.
Warning! Using a VBR source that does not have an associated seek table or seek information (such as MP3 or MOD/S3M/XM/IT) may cause inaccurate seeking if you specify FMOD_TIMEUNIT_MS or FMOD_TIMEUNIT_PCM.
If you want FMOD to create a pcm vs bytes seek table so that seeking is accurate, you will have to specify FMOD_ACCURATETIME when loading or opening the sound. This means there is a slight delay as FMOD scans the whole file when loading the sound to create this table.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getPosition
- FMOD_TIMEUNIT
- FMOD_MODE
- Sound::getLength

Channel::setPriority

Sets the priority for a channel after it has been played. A sound with a higher priority than another sound will not be stolen or made virtual by that sound.?

Syntax

```
FMOD_RESULT Channel::setPriority(
  int  priority
);
```

Parameters

priority

priority for the channel. 0 to 256 inclusive. 0 = most important. 256 = least important. Default = 128.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Priority will make a channel more important or less important than its counterparts. When virtual channels are in place, by default the importance of the sound (whether it is audible or not when more channels are playing than exist) is based on the volume, or audiblity of the sound. This is determined by distance from the listener in 3d, the volume set with Channel::setVolume, channel group volume, and geometry occlusion factors. To make a quiet sound more important, so that it isn't made virtual by louder sounds, you can use this function to increase its importance, and keep it audible.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getPriority
- Channel::setVolume

Channel::setReverbProperties

Sets the channel specific reverb properties, including things like wet/dry mix.?

Syntax

```
  FMOD_RESULT Channel::setReverbProperties(
  const FMOD_REVERB_CHANNELPROPERTIES *  prop
);
```

Parameters

prop

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

With FMOD_HARDWARE on Windows using EAX, the reverb will only work on FMOD_3D based sounds. FMOD_SOFTWARE does not have this problem and works on FMOD_2D and FMOD_3D based sounds.

On PlayStation 2, the 'Room' parameter is the only parameter supported. The hardware only allows 'on' or 'off', so the reverb will be off when 'Room' is -10000 and on for every other value.

On Xbox, it is possible to apply reverb to FMOD_2D and FMOD_HARDWARE based voices using this function. By default reverb is turned off for FMOD_2D hardware based voices.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getReverbProperties
- FMOD_REVERB_CHANNELPROPERTIES

Channel::setSpeakerLevels

Sets the incoming sound levels for a particular speaker.?

Syntax

```
  FMOD_RESULT Channel::setSpeakerLevels(
   FMOD_SPEAKER speaker,
   float * levels,
  int numlevels
);
```

Parameters

speaker

The target speaker to modify the levels for. This can be cast to an integer if you are using FMOD_SPEAKERMODE_RAW and want to access up to 15 speakers (output channels).

levels

An array of floating point numbers from 0.0 to 1.0 representing the volume of each input channel of a sound. See remarks for more.

numlevels

The number of floats within the levels parameter being passed to this function. In the case of the above mono or stereo sound, 1 or 2 could be used respectively. If the sound being played was an 8 channel multichannel sound then 8 levels would be used.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

As an example of usage of this function, if the sound played on this speaker was mono, only 1 level would be needed.
If the sound played on this channel was stereo, then an array of 2 floats could be specified. For example { 0, 1 } on a channel playing a stereo sound would mute the left part of the stereo sound when it is played on this speaker.

Note! In FMOD_SPEAKERMODE_MONO it is preferable to use the alias FMOD_SPEAKER_MONO.

Only speakers that are usable with the current speaker mode will be accepted. Anything else will return FMOD_ERR_INVALID_SPEAKER.

Under FMOD_SPEAKERMODE_RAW, the 'speaker' parameter can be cast to an integer and used as a raw speaker index, disregarding FMOD's speaker mappings.

Warning. This function will allocate memory for the speaker level matrix and attach it to the channel. If you prefer not to have a dynamic memory allocation done at this point use Channel::setSpeakerMix instead.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getSpeakerLevels
- Channel::setSpeakerMix
- FMOD_SPEAKERMODE
- FMOD_SPEAKER

Channel::setSpeakerMix

Sets the channel's speaker volume levels for each speaker individually.?

Syntax

```
FMOD_RESULT Channel::setSpeakerMix(
  float  frontleft,
  float  frontright,
  float  center,
  float  lfe,
  float  backleft,
  float  backright,
  float  sideleft,
  float  sideright
);
```

Parameters

frontleft

Volume level for this channel in the front left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

frontright

Volume level for this channel in the front right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, up to 5.0 = 5x amplification.

center

Volume level for this channel in the center speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

lfe

Volume level for this channel in the subwoofer speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

backleft

Volume level for this channel in the back left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

backright

Volume level for this channel in the back right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

sideleft

Volume level for this channel in the side left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

sideright

Volume level for this channel in the side right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = normal volume, 5.0 = 5x amplification.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function only fully works on sounds created with FMOD_2D and FMOD_SOFTWARE. FMOD_3D based sounds only allow setting of LFE channel, as all other speaker levels are calculated by FMOD's 3D engine.

Speakers specified that don't exist will simply be ignored.

For more advanced speaker control, including sending the different channels of a stereo sound to arbitrary speakers, see Channel::setSpeakerLevels.

This function allows amplification! You can go up to 5 times the volume of a normal sound, but warning this may cause clipping/distortion! Useful for LFE boosting.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getSpeakerMix
- Channel::setSpeakerLevels
- FMOD_SPEAKERMODE

Channel::setUserData

Sets a user value that the Channel object will store internally. Can be retrieved with Channel::getUserData.?

Syntax

```
  FMOD_RESULT Channel::setUserData(
   void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the Channel object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using Channel::getUserData would help in the identification of the object.

NOTE: If this channel was spawned by the event system then its user data field will be set, by the event system, to the event instance handle that spawned it and this function should NOT be called.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getUserData

Channel::setVolume

Sets the volume for the channel linearly.?

Syntax

```
  FMOD_RESULT Channel::setVolume(
    float  volume
);
```

Parameters

volume

A linear volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a sound is played, it plays at the default volume of the sound which can be set by Sound::setDefaults.
For most file formats, the volume is determined by the audio format.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::getVolume
- ChannelGroup::setVolume
- Sound::setDefaults

Channel::stop

Stops the channel from playing. Makes it available for re-use by the priority system.?

Syntax

```
FMOD_RESULT Channel::stop();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound

ChannelGroup Interface

ChannelGroup::addDSP

Adds a DSP effect to this channelgroup, affecting all channels that belong to it. Because it is a submix, only one instance of the effect is added, and all subsequent channels are affected.?

Syntax

```
FMOD_RESULT ChannelGroup::addDSP(
  FMOD::DSP *  dsp
);
```

Parameters

dsp

Pointer to the dsp effect to add. This can be created with System::createDSP, System::createDSPByType, System::createDSPByIndex.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is a wrapper function to insert a DSP unit at the top of the channel group DSP chain.
It disconnects the head unit from its input, then inserts the unit at the head and reconnects the previously disconnected input back as as an input to the new unit.
It is effectively the following code.

```
int numinputs;
channelgroup->getDSPHead(?
 dsphead->getNumInputs(?
if (numinputs > 1)
{
       return FMOD_ERR_DSP_TOOMANYCONNECTIONS;

}
 dsphead->getInput(0, ?
 dsphead->disconnectFrom(next);
 dsphead->addInput(dsp);
 dsp->addInput(next);
 dsp->setActive(true);
```

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getDSPHead
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- System::getMasterChannelGroup
- System::createChannelGroup
- System::addDSP
- Channel::addDSP
- DSP::remove

ChannelGroup::addGroup

Adds a channel group as a child of the current channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::addGroup(
    FMOD::ChannelGroup * group
);
```

Parameters

group

channel group to add as a child.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getNumGroups
- ChannelGroup::getGroup

ChannelGroup::get3DOcclusion

Retrieves the master occlusion factors for the channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::get3DOcclusion(
    float *  directocclusion,
    float *  reverbocclusion
);
```

Parameters

directocclusion

Address of a variable that receives the occlusion factor for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

reverbocclusion

Address of a variable that receives the occlusion factor for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::set3DOcclusion
- Channel::set3DOcclusion
- Channel::get3DOcclusion
- System::getMasterChannelGroup

ChannelGroup::getChannel

Retrieves the a handle to a channel from the current channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getChannel(
  int index,
   FMOD::Channel ** channel
);
```

Parameters

index

Index of the channel inside the channel group, from 0 to the number of channels returned by ChannelGroup::getNumChannels.

channel

Address of a variable to receieve a pointer to a Channel object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getNumChannels
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getDSPHead

Retrieves the DSP unit responsible for this channel group. When channels are submixed to this channel group, this is the DSP unit they target.?

Syntax

```
FMOD_RESULT ChannelGroup::getDSPHead(
  FMOD::DSP **  dsp
);
```

Parameters

dsp

Address of a variable to receive the pointer to the head DSP unit for this channel group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this unit if you wish to connect custom DSP units to the channelgroup or filter the channels in the channel group by inserting filter units between this one and the incoming channel mixer unit.
Read the tutorial on DSP if you wish to know more about this. It is not recommended using this if you do not understand how the FMOD Ex DSP network is connected.
Alternatively you can simply add effects by using ChannelGroup::addDSP which does the connection / disconnection work for you.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::addDSP
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getGroup

Retrieves a handle to a specified sub channelgroup.?

Syntax

```
 FMOD_RESULT ChannelGroup::getGroup(
 int index,
  FMOD::ChannelGroup ** group
);
```

Parameters

index

Index to specify which sub channelgroup to receieve.

group

Address of a variable to receieve a pointer to a channelgroup.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getNumGroups
- ChannelGroup::getParentGroup
- ChannelGroup::addGroup

ChannelGroup::getMute

Retrieves the mute state of a ChannelGroup.?

Syntax

```
  FMOD_RESULT ChannelGroup::getMute(
    bool * mute
);
```

Parameters

mute

Address of a variable to receive the pause state of the channelgroup.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setMute
- Channel::setMute
- Channel::getMute
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getName

Retrieves the name of the channelgroup. The name is set when the group is created.?

Syntax

```
  FMOD_RESULT ChannelGroup::getName(
  char * name,
  int namelen
);
```

Parameters

name

Address of a variable that receives the name of the channel group.

namelen

Length in bytes of the target buffer to receieve the string.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getNumChannels

Retrieves the current number of assigned channels to this channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getNumChannels(
  int *  numchannels
);
```

Parameters

numchannels

Address of a variable to receive the current number of assigned channels in this channel group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this function to enumerate the channels within the channel group. You can then use ChannelGroup::getChannel to retrieve each individual channel.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getChannel
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getNumGroups

Retrieves the number of sub groups under this channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getNumGroups(
  int *  numgroups
);
```

Parameters

numgroups

Address of a variable to receive the number of channel groups within this channel group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getGroup
- ChannelGroup::addGroup

ChannelGroup::getParentGroup

Retrieves a handle to this channelgroup's parent channelgroup.?

Syntax

```
  FMOD_RESULT ChannelGroup::getParentGroup(
    FMOD::ChannelGroup ** group
);
```

Parameters

group

Address of a variable to recieve a pointer to a channelgroup.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getNumGroups
- ChannelGroup::getGroup

ChannelGroup::getPaused

Retrieves the pause state of a ChannelGroup.?

Syntax

```
  FMOD_RESULT ChannelGroup::getPaused(
    bool *  paused
);
```

Parameters

paused

Address of a variable to receive the pause state of the channelgroup.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setPaused
- Channel::setPaused
- Channel::getPaused
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getPitch

Retrieves the master pitch level for the channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getPitch(
   float * pitch
);
```

Parameters

pitch

Address of a variable to receive the channel group pitch value, from 0.0 to 10.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setPitch
- ChannelGroup::overrideFrequency
- System::getMasterChannelGroup

ChannelGroup::getSpectrum

Retrieves the spectrum from the currently playing channels assigned to this channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getSpectrum(
    float * spectrumarray,
  int  numvalues,
  int channeloffset,
    FMOD_DSP_FFT_WINDOW  windowtype
);
```

Parameters

spectrumarray

Address of a variable that receives the spectrum data. This is an array of floating point values. Data will range is 0.0 to 1.0. Decibels = 10.0f * (float)log10(val) * 2.0f; See remarks for what the data represents.

numvalues

Size of array in floating point values being passed to the function. Must be a power of 2. (ie 128/256/512 etc). Min = 64. Max = 8192.

channeloffset

Channel of the signal to analyze. If the signal is multichannel (such as a stereo output), then this value represents which channel to analyze. On a stereo signal 0 = left, 1 = right.

windowtype

"Pre-FFT" window method. This filters the PCM data before entering the spectrum analyzer to reduce transient frequency error for more accurate results. See FMOD_DSP_FFT_WINDOW for different types of fft window techniques possible and for a more detailed explanation.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The larger the numvalues, the more CPU the FFT will take. Choose the right value to trade off between accuracy / speed.
The larger the numvalues, the more 'lag' the spectrum will seem to inherit. This is because the FFT window size stretches the analysis back in time to what was already played. For example if the window size happened to be 44100 and the output rate was 44100 it would be analyzing the past second of data, and giving you the average spectrum over that time period.

If you are not displaying the result in dB, then the data may seem smaller than it should be. To display it you may want to normalize the data - that is, find the maximum value in the resulting spectrum, and scale all values in the array by 1 / max. (ie if the max was 0.5f, then it would become 1).
To get the spectrum for both channels of a stereo signal, call this function twice, once with channeloffset = 0, and again with channeloffset = 1. Then add the spectrums together and divide by 2 to get the average spectrum for both channels.

What the data represents.
To work out what each entry in the array represents, use this formula

```
entry_hz = (output_rate / 2) / numvalues
```

The array represents amplitudes of each frequency band from 0hz to the nyquist rate. The nyquist rate is equal to the output rate divided by 2.
For example when FMOD is set to 44100hz output, the range of represented frequencies will be 0hz to 22049hz, a total of 22050hz represented.
If in the same example, 1024 was passed to this function as the numvalues, each entry's contribution would be as follows.

```
entry_hz = (44100 / 2) / 1024
entry_hz = 21.53 hz
```

Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_FFT_WINDOW
- System::getSpectrum
- Channel::getSpectrum
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getSystemObject

Retrieves the parent System object that created this channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getSystemObject(
    FMOD::System ** system
);
```

Parameters

system

Address of a variable that receives the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createChannelGroup
- System::getMasterChannelGroup

ChannelGroup::getUserData

Retrieves the user value that that was set by calling the ChannelGroup::setUserData function.?

Syntax

```
  FMOD_RESULT ChannelGroup::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the to user data specified with the ChannelGroup::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setUserData

ChannelGroup::getVolume

Retrieves the master volume level for the channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::getVolume(
    float *  volume
);
```

Parameters

volume

Address of a variable to receive the channel group volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setVolume
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::getWaveData

Retrieves a pointer to a block of PCM data that represents the currently playing waveform for this channel group.
?This function is useful for a very easy way to plot an oscilliscope.?

Syntax

```
FMOD_RESULT ChannelGroup::getWaveData(
  float *  wavearray,
  int  numvalues,
  int  channeloffset
);
```

Parameters

wavearray

Address of a variable that receives the currently playing waveform data. This is an array of floating point values.

numvalues

Number of floats to write to the array. Maximum value = 16384.

channeloffset

Offset into multichannel data. Mono channels use 0. Stereo channels use 0 = left, 1 = right. More than stereo use the appropriate index.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This is the actual resampled, filtered and volume scaled data, at the time this function is called.

Do not use this function to try and display the whole waveform of the sound, as this is more of a 'snapshot' of the current waveform at the time it is called, and could return the same data if it is called very quickly in succession. See the DSP API to capture a continual stream of wave data as it plays, or see Sound::lock / Sound::unlock if you want to simply display the waveform of a sound.

This function allows retrieval of left and right data for a stereo sound individually. To combine them into one signal, simply add the entries of each seperate buffer together and then divide them by 2.
Note: This function only displays data for sounds playing that were created with FMOD_SOFTWARE. FMOD_HARDWARE based sounds are played using the sound card driver and are not accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getMasterChannelGroup
- System::createChannelGroup
- Sound::lock
- Sound::unlock

ChannelGroup::override3DAttributes

Overrides the position and velocity of all channels within this channel group and those of any sub channelgroups.?

Syntax

```
  FMOD_RESULT ChannelGroup::override3DAttributes(
  const FMOD_VECTOR *  pos,
  const FMOD_VECTOR *  vel
);
```

Parameters

pos

Position in 3D space of the channels in the group. Specifying 0 / null will ignore this parameter.

vel

Velocity in 'distance units per second' in 3D space of the group of channels. See remarks. Specifying 0 / null will ignore this parameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified by System::set3DSettings. By default this is set to meters which is a distance scale of 1.0.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::set3DAttributes
- Channel::get3DAttributes
- FMOD_VECTOR
- System::set3DSettings

ChannelGroup::overrideFrequency

Overrides the frequency or playback rate, in HZ of all channels within this channel group and those of any sub channelgroups.?

Syntax

```
FMOD_RESULT ChannelGroup::overrideFrequency(
  float  frequency
);
```

Parameters

frequency

A frequency value in HZ. This value can also be negative to play the sound backwards (negative frequencies allowed with FMOD_SOFTWARE based non-stream sounds only). DirectSound hardware voices have limited frequency range on some soundcards. Please see remarks for more on this.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a sound is played, it plays at the default frequency of the sound which can be set by Sound::setDefaults.
For most file formats, the volume is determined by the audio format.

Frequency limitations for sounds created with FMOD_HARDWARE in DirectSound.
Every hardware device has a minimum and maximum frequency. This means setting the frequency above the maximum and below the minimum will have no effect.
FMOD clamps frequencies to these values when playing back on hardware, so if you are setting the frequency outside of this range, the frequency will stay at either the minimum or maximum.
Note that FMOD_SOFTWARE based sounds do not have this limitation.
To find out the minimum and maximum value before initializing FMOD (maybe to decide whether to use a different soundcard, output mode, or drop back fully to software mixing), you can use the System::getDriverCaps function.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setFrequency
- Channel::getFrequency
- System::getDriverCaps
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::overridePan

Sets pan position linearly of all channels within this channel group and those of any sub channelgroups.?

Syntax

```
  FMOD_RESULT ChannelGroup::overridePan(
    float  pan
);
```

Parameters

pan

A left/right pan level, from -1.0 to 1.0 inclusive. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Panning only works on sounds created with FMOD_2D. 3D sounds are not pannable.
Only sounds that are mono or stereo can be panned. Multichannel sounds (ie >2 channels) cannot be panned.

Mono sounds are panned from left to right using constant power panning. This means when pan = 0.0, the balance for the sound in each speaker is 71% left and 71% right, not 50% left and 50% right. This gives (audibly) smoother pans.
Stereo sounds heave each left/right value faded up and down according to the specified pan position. This means when pan = 0.0, the balance for the sound in each speaker is 100% left and 100% right. When pan = -1.0, only the left channel of the stereo sound is audible, when pan = 1.0, only the right channel of the stereo sound is audible.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getMasterChannelGroup
- System::createChannelGroup
- Channel::setPan
- Channel::getPan

ChannelGroup::overrideReverbProperties

Overrides the reverb properties of all channels within this channel group and those of any sub channelgroups.?

Syntax

```
  FMOD_RESULT ChannelGroup::overrideReverbProperties(
  const FMOD_REVERB_CHANNELPROPERTIES *  prop
);
```

Parameters

prop

Pointer to a FMOD_REVERB_CHANNELPROPERTIES structure definition.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

With FMOD_HARDWARE on Windows using EAX, the reverb will only work on FMOD_3D based sounds. FMOD_SOFTWARE does not have this problem and works on FMOD_2D and FMOD_3D based sounds.

On PlayStation 2, the 'Room' parameter is the only parameter supported. The hardware only allows 'on' or 'off', so the reverb will be off when 'Room' is -10000 and on for every other value.

On Xbox, it is possible to apply reverb to FMOD_2D and FMOD_HARDWARE based voices using this function. By default reverb is turned off for FMOD_2D hardware based voices, to make it compatible with EAX.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_CHANNELPROPERTIES
- System::setReverbProperties
- System::getReverbProperties

- Channel::setReverbProperties
- Channel::getReverbProperties
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::overrideSpeakerMix

Overrides all channel speaker levels for each speaker individually.?

Syntax

```
FMOD_RESULT ChannelGroup::overrideSpeakerMix(
  float  frontleft,
  float  frontright,
  float  center,
  float  lfe,
  float  backleft,
  float  backright,
  float  sideleft,
  float  sideright
);
```

Parameters

frontleft

Level for this channel in the front left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

frontright

Level for this channel in the front right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

center

Level for this channel in the center speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

lfe

Level for this channel in the subwoofer speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

backleft

Level for this channel in the back left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

backright

Level for this channel in the back right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

sideleft

Level for this channel in the side left speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

sideright

Level for this channel in the side right speaker of a multichannel speaker setup. 0.0 = silent, 1.0 = full volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function only works on sounds created with FMOD_2D. 3D sounds are not pannable and will return FMOD_ERR_NEEDS2D.

Only sounds create with FMOD_SOFTWARE playing on this channel will allow this functionality.

Speakers specified that don't exist will simply be ignored.

For more advanced speaker control, including sending the different channels of a stereo sound to arbitrary speakers, see Channel::setSpeakerLevels.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setSpeakerMix
- Channel::getSpeakerMix
- Channel::setSpeakerLevels

ChannelGroup::overrideVolume

Overrides the volume of all channels within this channel group and those of any sub channelgroups.?

Syntax

```
  FMOD_RESULT ChannelGroup::overrideVolume(
    float  volume
);
```

Parameters

volume

A linear volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
This is not to be used as a master volume for the group, as it will modify the volumes of the channels themselves.

If you want to scale the volume of the group, use ChannelGroup::setVolume.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getMasterChannelGroup
- System::createChannelGroup
- ChannelGroup::setVolume

ChannelGroup::release

Frees a channel group.?

Syntax

```
FMOD_RESULT ChannelGroup::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

All channels assigned to this group are returned back to the master channel group owned by the System object. See System::getMasterChannelGroup.
All child groups assigned to this group are returned back to the master channel group owned by the System object. See System::getMasterChannelGroup.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createChannelGroup
- System::getMasterChannelGroup

ChannelGroup::set3DOcclusion

Sets the master occlusion factors for the channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::set3DOcclusion(
    float  directocclusion,
    float  reverbocclusion
);
```

Parameters

directocclusion

Occlusion factor for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

reverbocclusion

Occlusion factor for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not go through and overwrite the channel occlusion factors. It scales them by the channel group's occlusion factors.
That way when Channel::set3DOcclusion / Channel::get3DOcclusion is called the respective individual channel occlusion factors will still be preserved. This means that final Channel occlusion values will be affected by both ChannelGroup occlusion and geometry (if any).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::get3DOcclusion
- Channel::set3DOcclusion
- Channel::get3DOcclusion
- System::getMasterChannelGroup

ChannelGroup::setMute

Mutes a channelgroup, and the channels within it, or unmutes any unmuted channels if set to false.?

Syntax

```
  FMOD_RESULT ChannelGroup::setMute(
    bool mute
);
```

Parameters

mute

Mute state to set. true = channelgroup state is set to muted. false = channelgroup state is set to unmuted.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A channelgroup maintains a mute state, that affects channelgroups and channels within it. If a channelgroup is muted, all channelgroups and channels below it will become muted.
Channels will not have their per channel mute state overwritten, so that when a channelgroup is unmuted, the muted state of the channels will correct as they were set on a per channel basis.
This means even though a channel is muted, it can return false when you call Channel::getMute on that channel, because that was the state of the channel at the time before the ChannelGroup was muted.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getMute
- Channel::setMute
- Channel::getMute
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::setPaused

Pauses a channelgroup, and the channels within it, or unpauses any unpaused channels if set to false.?

Syntax

```
  FMOD_RESULT ChannelGroup::setPaused(
    bool  paused
);
```

Parameters

paused

Paused state to set. true = channelgroup state is set to paused. false = channelgroup state is set to unpaused.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A channelgroup maintains a paused state, that affects channelgroups and channels within it. If a channelgroup is paused, all channelgroups and channels below it will become paused.
Channels will not have their per channel pause state overwritten, so that when a channelgroup is unpaused, the paused state of the channels will correct as they were set on a per channel basis.
This means even though a channel is paused, it can return false when you call Channel::getPaused on that channel, because that was the state of the channel at the time before the ChannelGroup was paused.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getPaused
- Channel::setPaused
- Channel::getPaused
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::setPitch

Sets the master pitch for the channel group.?

Syntax

```
  FMOD_RESULT ChannelGroup::setPitch(
    float  pitch
);
```

Parameters

pitch

A pitch level, from 0.0 to 10.0 inclusive. 0.5 = half pitch, 2.0 = double pitch. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not go through and overwrite the channel frequencies. It scales them by the channel group's pitch. That way when Channel::setFrequency / Channel::getFrequency is called the respective individual channel frequencies will still be preserved.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::overrideFrequency
- System::getMasterChannelGroup
- ChannelGroup::getPitch
- Channel::setFrequency
- Channel::getFrequency
- ChannelGroup::overrideFrequency

ChannelGroup::setUserData

Sets a user value that the ChannelGroup object will store internally. Can be retrieved with ChannelGroup::getUserData.?

Syntax

```
FMOD_RESULT ChannelGroup::setUserData(
  void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the ChannelGroup object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using ChannelGroup::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::getUserData
- System::getMasterChannelGroup
- System::createChannelGroup

ChannelGroup::setVolume

Sets the master volume for the channel group linearly.?

Syntax

```
  FMOD_RESULT ChannelGroup::setVolume(
    float  volume
);
```

Parameters

volume

A linear volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not go through and overwrite the channel volumes. It scales them by the channel group's volume. That way when Channel::setVolume / Channel::getVolume is called the respective individual channel volumes will still be preserved.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- ChannelGroup::setVolume
- Channel::setVolume
- Channel::getVolume
- ChannelGroup::overrideVolume

ChannelGroup::stop

Stops all channels within the channelgroup.?

Syntax

```
FMOD_RESULT ChannelGroup::stop();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::getMasterChannelGroup
- System::createChannelGroup

SoundGroup Interface

SoundGroup::getMaxAudible

Retrieves the number of concurrent playbacks of sounds in a sound group to the specified value.
?If the sounds in the sound group are playing this many times, any attepts to play more of the sounds in the sound group will fail with FMOD_ERR_MAXAUDIBLE.?

Syntax

```
  FMOD_RESULT SoundGroup::getMaxAudible(
  int *  maxaudible
);
```

Parameters

maxaudible

Address of a variable to recieve the number of playbacks to be audible at once. -1 = unlimited. 0 means no sounds in this group will succeed. Default = -1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

SoundGroup::getNumPlaying can be used to determine how many instances of the sounds in the sound group are playing.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::setMaxAudible
- SoundGroup::getNumPlaying
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getMaxAudibleBehavior

Retrieves the current max audible behavior method.?

Syntax

```
  FMOD_RESULT SoundGroup::getMaxAudibleBehavior(
    FMOD_SOUNDGROUP_BEHAVIOR *  behavior
);
```

Parameters

behavior

Address of a variable to recieve the current sound group max playbacks behavior. Default is FMOD_SOUNDGROUP_BEHAVIOR_FAIL.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_SOUNDGROUP_BEHAVIOR
- SoundGroup::setMaxAudibleBehavior
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- SoundGroup::setMuteFadeSpeed
- SoundGroup::getMuteFadeSpeed
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getMuteFadeSpeed

Retrieves the current time in seconds for FMOD_SOUNDGROUP_BEHAVIOR_MUTE behavior to fade with.?

Syntax

```
  FMOD_RESULT SoundGroup::getMuteFadeSpeed(
   float * speed
);
```

Parameters

speed

Address of a variable to receive the fade time in seconds (1.0 = 1 second). Default = 0.0. (no fade).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a mode besides FMOD_SOUNDGROUP_BEHAVIOR_MUTE is used, the fade speed is ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::setMuteFadeSpeed
- SoundGroup::setMaxAudibleBehavior
- SoundGroup::getMaxAudibleBehavior
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getName

Retrieves the name of the sound group.?

Syntax

```
  FMOD_RESULT SoundGroup::getName(
  char * name,
  int namelen
);
```

Parameters

name

Address of a variable that receives the name of the sound group.

namelen

Length in bytes of the target buffer to receieve the string.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getNumPlaying

Retrieves the number of currently playing channels for the sound group.?

Syntax

```
  FMOD_RESULT SoundGroup::getNumPlaying(
  int *  numplaying
);
```

Parameters

numplaying

Address of a variable to receive the number of actively playing channels from sounds in this sound group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This routine returns the number of channels playing. If the sound group only has 1 sound, and that sound is playing twice, the figure returned will be 2.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getNumSounds

Retrieves the current number of sounds in this sound group.?

Syntax

```
  FMOD_RESULT SoundGroup::getNumSounds(
  int *  numsounds
);
```

Parameters

numsounds

Address of a variable to receive the number of sounds in this sound group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::getMasterSoundGroup
- SoundGroup::setMaxAudible
- SoundGroup::getSound

SoundGroup::getSound

Retrieves a pointer to a sound from within a sound group.?

Syntax

```
  FMOD_RESULT SoundGroup::getSound(
  int  index,
   FMOD::Sound **  sound
);
```

Parameters

index

Index of the sound that is to be retrieved.

sound

Address of a variable to receieve a pointer to a Sound object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use SoundGroup::getNumSounds in conjunction with this function to enumerate all sounds in a sound group.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::createSound
- SoundGroup::getNumSounds
- System::getMasterSoundGroup

SoundGroup::getSystemObject

Retrieves the parent System object that was used to create this object.?

Syntax

```
  FMOD_RESULT SoundGroup::getSystemObject(
    FMOD::System ** system
);
```

Parameters

system

Address of a pointer that receives the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getUserData

Retrieves the user value that that was set by calling the SoundGroup::setUserData function.?

Syntax

```
  FMOD_RESULT SoundGroup::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the SoundGroup::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::setUserData
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::getVolume

Retrieves the volume for the sounds within a soundgroup.?

Syntax

```
  FMOD_RESULT SoundGroup::getVolume(
    float *  volume
);
```

Parameters

volume

Address of a variable to receive the soundgroup volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::setVolume
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::release

Releases a soundgroup object and returns all sounds back to the master sound group.?

Syntax

```
FMOD_RESULT SoundGroup::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

You cannot release the master sound group.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::setMaxAudible

Limits the number of concurrent playbacks of sounds in a sound group to the specified value.
?After this, if the sounds in the sound group are playing this many times, any attepts to play more of the sounds in the sound group will by default fail with FMOD_ERR_MAXAUDIBLE.
?Use SoundGroup::setMaxAudibleBehavior to change the way the sound playback behaves when too many sounds are playing. Muting, failing and stealing behaviors can be specified.
?

Syntax

```
  FMOD_RESULT SoundGroup::setMaxAudible(
  int maxaudible
);
```

Parameters

maxaudible

Number of playbacks to be audible at once. -1 = unlimited. 0 means no sounds in this group will succeed. Default = -1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

SoundGroup::getNumPlaying can be used to determine how many instances of the sounds in the sound group are currently playing.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSoundGroup
- SoundGroup::getMaxAudible
- SoundGroup::getNumPlaying
- SoundGroup::setMaxAudibleBehavior
- SoundGroup::getMaxAudibleBehavior
- System::getMasterSoundGroup

Version 4.12.03 Built on Feb 18, 2008

SoundGroup::setMaxAudibleBehavior

This function changes the way the sound playback behaves when too many sounds are playing in a soundgroup. Muting, failing and stealing behaviors can be specified.
?

Syntax

```
  FMOD_RESULT SoundGroup::setMaxAudibleBehavior(
    FMOD_SOUNDGROUP_BEHAVIOR  behavior
);
```

Parameters

behavior

Specify a behavior determined with a FMOD_SOUNDGROUP_BEHAVIOR flag. Default is FMOD_SOUNDGROUP_BEHAVIOR_FAIL.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_SOUNDGROUP_BEHAVIOR
- SoundGroup::getMaxAudibleBehavior
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- SoundGroup::setMuteFadeSpeed
- SoundGroup::getMuteFadeSpeed
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::setMuteFadeSpeed

Specify a time in seconds for FMOD_SOUNDGROUP_BEHAVIOR_MUTE behavior to fade with. By default there is no fade.
?When more sounds are playing in a SoundGroup than are specified with SoundGroup::setMaxAudible, the least important sound (ie lowest priority / lowest audible volume due to 3d position, volume etc) will fade to silence if FMOD_SOUNDGROUP_BEHAVIOR_MUTE is used, and any previous sounds that were silent because of this rule will fade in if they are more important.
?

Syntax

```
  FMOD_RESULT SoundGroup::setMuteFadeSpeed(
    float speed
);
```

Parameters

speed

Fade time in seconds (1.0 = 1 second). Default = 0.0. (no fade).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a mode besides FMOD_SOUNDGROUP_BEHAVIOR_MUTE is used, the fade speed is ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::getMuteFadeSpeed
- SoundGroup::setMaxAudibleBehavior
- SoundGroup::getMaxAudibleBehavior
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::setUserData

Sets a user value that the SoundGroup object will store internally. Can be retrieved with SoundGroup::getUserData.?

Syntax

```
  FMOD_RESULT SoundGroup::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the sound group object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using SoundGroup::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::getUserData
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::setVolume

Sets the volume for a sound group, affecting all channels playing the sounds in this soundgroup.?

Syntax

```
  FMOD_RESULT SoundGroup::setVolume(
    float  volume
);
```

Parameters

volume

A linear volume level, from 0.0 to 1.0 inclusive. 0.0 = silent, 1.0 = full volume. Default = 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- SoundGroup::getVolume
- System::createSoundGroup
- System::getMasterSoundGroup

SoundGroup::stop

Stops all sounds within this soundgroup.?

Syntax

```
FMOD_RESULT SoundGroup::stop();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::createSoundGroup
- System::getMasterSoundGroup

DSP Interface

DSP::addInput

Adds the specified DSP unit as an input of the DSP object.?

Syntax

```
  FMOD_RESULT DSP::addInput(
    FMOD::DSP *  target
);
```

Parameters

target

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
Adding a unit as an input means that there can be multiple units added to the target.

Inputs are automatically mixed together, then the mixed data is sent to the unit's output(s).

To find the number of inputs or outputs a unit has use DSP::getNumInputs or DSP::getNumOutputs.

Remarks

If you want to add a unit as an output of another unit, then add 'this' unit as an input of that unit instead.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getNumInputs
- DSP::getInput
- DSP::getNumOutputs
- DSP::disconnectFrom

DSP::disconnectAll

Helper function to disconnect either all inputs or all outputs of a dsp unit.?

Syntax

```
  FMOD_RESULT DSP::disconnectAll(
    bool inputs,
    bool outputs
);
```

Parameters

inputs

outputs

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is optimized to be faster than disconnecting inputs and outputs manually one by one.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::disconnectFrom

DSP::disconnectFrom

Disconnect the DSP unit from the specified target.?

Syntax

```
  FMOD_RESULT DSP::disconnectFrom(
    FMOD::DSP *  target
);
```

Parameters

target

The unit that this unit is to be removed from. Specify 0 or NULL to disconnect the unit from all outputs and inputs.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that when you disconnect a unit, it is up to you to reconnect the network so that data flow can continue.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::addInput
- DSP::disconnectAll

DSP::getActive

Retrieves the active state of a DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getActive(
    bool * active
);
```

Parameters

active

Address of a variable that receives the active state of the unit. true = unit is activated, false = unit is deactivated.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setActive
- DSP::setBypass

DSP::getBypass

Retrieves the bypass state of the DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getBypass(
    bool * bypass
);
```

Parameters

bypass

Address of a variable that receieves the bypass state for a DSP unit. true = unit is not processing audio data, false = unit is processing audio data. Default = false.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a unit is bypassed, it will still process its inputs, unlike DSP::setActive (when set to false) which causes inputs to stop processing as well.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setBypass
- DSP::setActive

DSP::getDefaults

Retrieves the default frequency, volume, pan and more for this DSP unit if it was to ever be played on a channel using System::playDSP.?

Syntax

```
  FMOD_RESULT DSP::getDefaults(
    float *  frequency,
    float *  volume,
    float *  pan,
  int *  priority
);
```

Parameters

frequency

Address of a variable that receives the default frequency for the DSP unit. Optional. Specify 0 or NULL to ignore.

volume

Address of a variable that receives the default volume for the DSP unit. Result will be from 0.0 to 1.0. 0.0 = Silent, 1.0 = full volume. Default = 1.0. Optional. Specify 0 or NULL to ignore.

pan

Address of a variable that receives the default pan for the DSP unit. Result will be from -1.0 to +1.0. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0. Optional. Specify 0 or NULL to ignore.

priority

Address of a variable that receives the default priority for the DSP unit when played on a channel. Result will be from 0 to 256. 0 = most important, 256 = least important. Default = 128. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setDefaults

- System::playDSP

DSP::getInfo

Retrieves information about the current DSP unit, including name, version, default channels and width and height of configuration dialog box if it exists.?

Syntax

```
  FMOD_RESULT DSP::getInfo(
  char *  name,
  unsigned int *  version,
  int * channels,
  int * configwidth,
  int * configheight
);
```

Parameters

name

Address of a variable that receives the name of the unit. This will be a maximum of 32bytes. If the DSP unit has filled all 32 bytes with the name with no terminating \0 null character it is up to the caller to append a null character. Optional. Specify 0 or NULL to ignore.

version

Address of a variable that receives the version number of the DSP unit. Version number is usually formated as hex AAAABBBB where the AAAA is the major version number and the BBBB is the minor version number. Optional. Specify 0 or NULL to ignore.

channels

Address of a variable that receives the number of channels the unit was initialized with. 0 means the plugin will process whatever number of channels is currently in the network. >0 would be mostly used if the unit is a unit that only generates sound, or is not flexible enough to take any number of input channels. Optional. Specify 0 or NULL to ignore.

configwidth

Address of a variable that receives the width of an optional configuration dialog box that can be displayed with DSP::showConfigDialog. 0 means the dialog is not present. Optional. Specify 0 or NULL to ignore.

configheight

Address of a variable that receives the height of an optional configuration dialog box that can be displayed with DSP::showConfigDialog. 0 means the dialog is not present. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::showConfigDialog

DSP::getInput

Retrieves a pointer to a DSP unit which is acting as an input to this unit.?

Syntax

```
  FMOD_RESULT DSP::getInput(
  int index,
   FMOD::DSP ** input
);
```

Parameters

index

Index of the input unit to retrieve.

input

Address of a variable that receieves the pointer to the desired input unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

An input is a unit which feeds audio data to this unit.
If there are more than 1 input to this unit, the inputs will be mixed, and the current unit processes the mixed result.
Find out the number of input units to this unit by calling DSP::getNumInputs.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getNumInputs
- DSP::addInput
- DSP::getOutput

DSP::getInputLevels

Retrieves the speaker mix for a DSP unit's input.?

Syntax

```
  FMOD_RESULT DSP::getInputLevels(
  int index,
   FMOD_SPEAKER speaker,
   float * levels,
  int numlevels
);
```

Parameters

index

DSP input index to get the speaker levels from. The number of inputs for a DSP unit can be found with DSP::getNumInputs.

speaker

The target speaker to get the levels from. This can be cast to an integer if you are using a device with more than the pre-defined speaker range.

levels

Address of an array of floating point numbers to get the speaker levels of an input.

numlevels

The number of floats within the levels parameter being passed to this function. In the case of the above mono or stereo sound, 1 or 2 could be used respectively. If the sound being played was an 8 channel multichannel sound then 8 levels would be used.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setInputLevels

- DSP::getNumInputs
- DSP::getOutputLevels

DSP::getInputMix

Retrieves the volume of the specified input to be mixed into this unit.?

Syntax

```
  FMOD_RESULT DSP::getInputMix(
  int index,
   float * volume
);
```

Parameters

index

Input index to retrieve the volume for. The number of inputs for a DSP unit can be found with DSP::getNumInputs.

volume

Address of a variable to receive the volume or mix level of the specified input. 0.0 = silent, 1.0 = full volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setInputMix
- DSP::getNumInputs
- DSP::getOutputMix

DSP::getNumInputs

Retrieves the number of inputs connected to the DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getNumInputs(
  int * numinputs
);
```

Parameters

numinputs

Address of a variable that receives the number of inputs connected to this unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Inputs are units that feed data to this unit. When there are multiple inputs, they are mixed together.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getNumOutputs
- DSP::getInput

DSP::getNumOutputs

Retrieves the number of outputs connected to the DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getNumOutputs(
  int *  numoutputs
);
```

Parameters

numoutputs

Address of a variable that receives the number of outputs connected to this unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Outputs are units that this unit feeds data to. When there are multiple outputs, the data is split and sent to each unit individually.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getNumInputs
- DSP::getOutput

DSP::getNumParameters

Retrieves the number of parameters a DSP unit has to control its behaviour.?

Syntax

```
  FMOD_RESULT DSP::getNumParameters(
  int *  numparams
);
```

Parameters

numparams

Address of a variable that receives the number of parameters contained within this DSP unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this to enumerate all parameters of a DSP unit with DSP::getParameter and DSP::getParameterInfo.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setParameter
- DSP::getParameter
- DSP::getParameterInfo

DSP::getOutput

Retrieves a pointer to a DSP unit which is acting as an output to this unit.?

Syntax

```
  FMOD_RESULT DSP::getOutput(
  int index,
   FMOD::DSP ** output
);
```

Parameters

index

Index of the output unit to retrieve.

output

Address of a variable that receieves the pointer to the desired output unit.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

An output is a unit which this unit will feed data too once it has processed its data.
Find out the number of output units to this unit by calling DSP::getNumOutputs.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getNumOutputs
- DSP::addInput
- DSP::getInput

DSP::getOutputLevels

Retrieves the speaker mix for a DSP unit's output.?

Syntax

```
  FMOD_RESULT DSP::getOutputLevels(
  int  index,
   FMOD_SPEAKER  speaker,
   float *  levels,
  int  numlevels
);
```

Parameters

index

DSP output index to get the speaker levels from. The number of outputs for a DSP unit can be found with DSP::getNumOutputs.

speaker

The target speaker to get the levels from. This can be cast to an integer if you are using a device with more than the pre-defined speaker range.

levels

Address of an array of floating point numbers to get the speaker levels of an output.

numlevels

The number of floats within the levels parameter being passed to this function. In the case of the above mono or stereo sound, 1 or 2 could be used respectively. If the sound being played was an 8 channel multichannel sound then 8 levels would be used.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setOutputLevels

- DSP::getNumOutputs
- DSP::getOutputLevels

DSP::getOutputMix

Retrieves the volume of the specified output to be mixed into this unit.?

Syntax

```
  FMOD_RESULT DSP::getOutputMix(
  int  index,
   float *  volume
);
```

Parameters

index

Output index to retrieve the volume for. The number of outputs for a DSP unit can be found with DSP::getNumOutputs.

volume

Address of a variable to receive the volume or mix level of the specified output. 0.0 = silent, 1.0 = full volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setOutputMix
- DSP::getNumOutputs
- DSP::getInputMix

DSP::getParameter

Retrieves a DSP unit's parameter by index. To find out the parameter names and range, see the see also field.?

Syntax

```
  FMOD_RESULT DSP::getParameter(
  int  index,
   float *  value,
  char *  valuestr,
  int  valuestrlen
);
```

Parameters

index

Parameter index for this unit. Find the number of parameters with DSP::getNumParameters.

value

Address of a variable that receives the parameter value. The parameter properties can be retrieved with DSP::getParameterInfo.

valuestr

Address of a variable that receives the string containing a formatted or more meaningful representation of the DSP parameter's value. For example if a switch parameter has on and off (0.0 or 1.0) it will display "ON" or "OFF" by using this parameter.

valuestrlen

Length of the user supplied memory in bytes that valuestr will write to. This will not exceed 16 bytes.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getParameterInfo
- DSP::getNumParameters

- DSP::setParameter

DSP::getParameterInfo

Retrieve information about a specified parameter within the DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getParameterInfo(
  int index,
  char * name,
  char * label,
  char * description,
  int descriptionlen,
   float * min,
   float * max
);
```

Parameters

index

Parameter index for this unit. Find the number of parameters with DSP::getNumParameters.

name

Address of a variable that receives the name of the parameter. An example is "Gain". This is a maximum string length of 16bytes (append \0 in case the plugin has used all 16 bytes for the string).

label

Address of a variable that receives the label of the parameter (ie a parameter type that might go next to the parameter). An example is "dB". This is a maximum string length of 16bytes (append \0 in case the plugin has used all 16 bytes for the string).

description

Address of a variable that receives the more descriptive text about the parameter (ie for a tooltip). An example is "Controls the input level for the effect in decibels".

descriptionlen

Maximum length of user supplied description string in bytes that FMOD will write to.

min

Minimum range of the parameter.

max

Maximum range of the parameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use DSP::getNumParameters to find out the number of parameters for this DSP unit.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setParameter
- DSP::getParameter
- DSP::getNumParameters

DSP::getSystemObject

Retrieves the parent System object that was used to create this object.?

Syntax
```
  FMOD_RESULT DSP::getSystemObject(
    FMOD::System ** system
);
```

Parameters

system

Address of a variable that receives the System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also
- System::createDSP
- System::createDSPByType
- System::getDSPHead
- Channel::getDSPHead
- ChannelGroup::getDSPHead

DSP::getType

Retrieves the pre-defined type of a FMOD registered DSP unit.?

Syntax

```
  FMOD_RESULT DSP::getType(
    FMOD_DSP_TYPE *  type
);
```

Parameters

type

Address of a variable to recieve the FMOD dsp type.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This is only valid for built in FMOD effects. Any user plugins will simply return FMOD_DSP_TYPE_UNKNOWN.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_DSP_TYPE

DSP::getUserData

Retrieves the user value that that was set by calling the DSP::setUserData function.?

Syntax

```
  FMOD_RESULT DSP::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the to user data specified with the DSP::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::setUserData

DSP::release

Frees a DSP object.?

Syntax

```
FMOD_RESULT DSP::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This will free the DSP object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createDSP
- System::createDSPByType
- System::getDSPHead
- Channel::getDSPHead
- ChannelGroup::getDSPHead

DSP::remove

Removes a unit from a DSP chain and connects the unit's input and output together after it is gone.?

Syntax

```
FMOD_RESULT DSP::remove();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is generally only used with units that have been added with System::addDSP or Channel::addDSP.
A unit that has been added in this way generally only has one input and one output, so this function assumes this and takes input 0 and connects it with output 0 after it has been removed, so that the data flow is not broken.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::addDSP
- Channel::addDSP
- ChannelGroup::addDSP

DSP::reset

Calls the DSP unit's reset function, which will clear internal buffers and reset the unit back to an initial state.?

Syntax

```
FMOD_RESULT DSP::reset();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Calling this function is useful if the DSP unit relies on a history to process itself (ie an echo filter).
If you disconnected the unit and reconnected it to a different part of the network with a different sound, you would want to call this to reset the units state (ie clear and reset the echo filter) so that you dont get left over artifacts from the place it used to be connected.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

DSP::setActive

Enables or disables a unit for being processed.?

Syntax

```
  FMOD_RESULT DSP::setActive(
    bool  active
);
```

Parameters

active

true = unit is activated, false = unit is deactivated.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This does not connect or disconnect a unit in any way, it just disables it so that it is not processed.
If a unit is disabled, and has inputs, they will also cease to be processed.
To disable a unit but allow the inputs of the unit to continue being processed, use DSP::setBypass instead.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getActive
- DSP::setBypass

DSP::setBypass

Enables or disables the read callback of a DSP unit so that it does or doesn't process the data coming into it. ?A DSP unit that is disabled still processes its inputs, it will just be 'dry'.?

Syntax

```
  FMOD_RESULT DSP::setBypass(
   bool  bypass
);
```

Parameters

bypass

Boolean to cause the read callback of the DSP unit to be bypassed or not. Default = false.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If a unit is bypassed, it will still process its inputs.
To disable the unit and all of its inputs, use DSP::setActive instead.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getBypass
- DSP::setActive

DSP::setDefaults

If a DSP unit is to be played on a channel with System::playDSP, this will set the defaults for frequency, volume, pan and more for the channel.?

Syntax

```
FMOD_RESULT DSP::setDefaults(
  float  frequency,
  float  volume,
  float  pan,
  int  priority
);
```

Parameters

frequency

Default playback frequency for the DSP unit, in hz. (ie 44100hz).

volume

Default volume for the DSP unit. 0.0 to 1.0. 0.0 = Silent, 1.0 = full volume. Default = 1.0.

pan

Default pan for the DSP unit. -1.0 to +1.0. -1.0 = Full left, 0.0 = center, 1.0 = full right. Default = 0.0.

priority

Default priority for the DSP unit when played on a channel. 0 to 256. 0 = most important, 256 = least important. Default = 128.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

There are no 'ignore' values for these parameters. Use DSP::getDefaults if you want to change only 1 and leave others unaltered.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playDSP
- DSP::getDefaults

DSP::setInputLevels

Sets the speaker mix for a DSP unit's input.?

Syntax

```
FMOD_RESULT DSP::setInputLevels(
  int index,
  FMOD_SPEAKER speaker,
  float * levels,
  int numlevels
);
```

Parameters

index

DSP input index to set the speaker levels for. The number of inputs for a DSP unit can be found with DSP::getNumInputs.

speaker

The target speaker to modify the levels for. This can be cast to an integer if you are using a device with more than the pre-defined speaker range.

levels

An array of floating point numbers from 0.0 to 1.0 representing the volume of each input channel of a sound. See remarks for more.

numlevels

The number of floats within the levels parameter being passed to this function. In the case of the above mono or stereo sound, 1 or 2 could be used respectively. If the sound being played was an 8 channel multichannel sound then 8 levels would be used.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

As an example of usage of this function, if the sound played on this speaker was mono, only 1 level would be needed.
If the sound played on this channel was stereo, then an array of 2 floats could be specified. For example { 0, 1 } on a channel playing a stereo sound would mute the left part of the stereo sound when it is played on this speaker.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getInputLevels
- DSP::getNumInputs
- DSP::setOutputLevels

DSP::setInputMix

Sets the volume of the specified input to be mixed into this unit.?

Syntax

```
  FMOD_RESULT DSP::setInputMix(
  int index,
   float volume
);
```

Parameters

index

Input index to set the volume level for. The number of inputs for a DSP unit can be found with DSP::getNumInputs.

volume

Volume or mix level of the specified input. 0.0 = silent, 1.0 = full volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getInputMix
- DSP::getNumInputs
- DSP::setOutputMix

DSP::setOutputLevels

Sets the speaker mix for a DSP unit's output.?

Syntax

```
  FMOD_RESULT DSP::setOutputLevels(
  int  index,
   FMOD_SPEAKER  speaker,
   float *  levels,
  int  numlevels
);
```

Parameters

index

DSP output index to set the speaker levels for. The number of outputs for a DSP unit can be found with DSP::getNumOutputs.

speaker

The target speaker to modify the levels for. This can be cast to an integer if you are using a device with more than the pre-defined speaker range.

levels

An array of floating point numbers from 0.0 to 1.0 representing the volume of each output channel of a sound. See remarks for more.

numlevels

The number of floats within the levels parameter being passed to this function. In the case of the above mono or stereo sound, 1 or 2 could be used respectively. If the sound being played was an 8 channel multichannel sound then 8 levels would be used.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

As an example of usage of this function, if the sound played on this speaker was mono, only 1 level would be needed.
If the sound played on this channel was stereo, then an array of 2 floats could be specified. For example { 0, 1 } on a channel playing a stereo sound would mute the left part of the stereo sound when it is played on this speaker.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getOutputLevels
- DSP::getNumOutputs
- DSP::setInputLevels

DSP::setOutputMix

Sets the volume of the specified output to be mixed into this unit.?

Syntax

```
  FMOD_RESULT DSP::setOutputMix(
  int index,
   float volume
);
```

Parameters

index

Output index to set the volume level for. The number of outputs for a DSP unit can be found with DSP::getNumOutputs.

volume

Volume or mix level of the specified output. 0.0 = silent, 1.0 = full volume.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getOutputMix
- DSP::getNumOutputs
- DSP::setInputMix

DSP::setParameter

Sets a DSP unit's parameter by index. To find out the parameter names and range, see the see also field.?

Syntax

```
  FMOD_RESULT DSP::setParameter(
  int index,
   float value
);
```

Parameters

index

Parameter index for this unit. Find the number of parameters with DSP::getNumParameters.

value

Parameter value. The parameter properties can be retrieved with DSP::getParameterInfo.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getParameterInfo
- DSP::getNumParameters
- DSP::getParameter

DSP::setUserData

Sets a user value that the DSP object will store internally. Can be retrieved with DSP::getUserData.?

Syntax

```
  FMOD_RESULT DSP::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the DSP object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using DSP::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getUserData

DSP::showConfigDialog

Display or hide a DSP unit configuration dialog box inside the target window.?

Syntax

```
  FMOD_RESULT DSP::showConfigDialog(
    void *  hwnd,
    bool  show
);
```

Parameters

hwnd

Target HWND in windows to display configuration dialog.

show

true = show dialog box inside target hwnd. false = remove dialog from target hwnd.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Dialog boxes are used by DSP plugins that prefer to use a graphical user interface to modify their parameters rather than using the other method of enumerating the parameters and using DSP::setParameter.
These are usually VST plugins. FMOD Ex plugins do not have configuration dialog boxes. To find out what size window to create to store the configuration screen, use DSP::getInfo where you can get the width and height.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- DSP::getInfo
- DSP::setParameter
- DSP::getParameter

Geometry Interface

Geometry::addPolygon

Adds a polygon to an existing geometry object.?

Syntax

```
  FMOD_RESULT Geometry::addPolygon(
   float  directocclusion,
   float  reverbocclusion,
   bool  doublesided,
  int  numvertices,
  const FMOD_VECTOR *  vertices,
  int *  polygonindex
);
```

Parameters

directocclusion

Occlusion value from 0.0 to 1.0 which affects volume or audible frequencies. 0.0 = The polygon does not occlude volume or audible frequencies (sound will be fully audible), 1.0 = The polygon fully occludes (sound will be silent).

reverbocclusion

Occlusion value from 0.0 to 1.0 which affects the reverb mix. 0.0 = The polygon does not occlude reverb (reverb reflections still travel through this polygon), 1.0 = The polyfully fully occludes reverb (reverb reflections will be silent through this polygon).

doublesided

Description of polygon if it is double sided or single sided. true = polygon is double sided, false = polygon is single sided, and the winding of the polygon (which determines the polygon's normal) determines which side of the polygon will cause occlusion.

numvertices

Number of vertices in this polygon. This must be at least 3. Polygons (more than 3 sides) are supported.

vertices

A pointer to an array of vertices located in object space, with the count being the number of vertices described using the numvertices parameter.

polygonindex

Address of a variable to receieve the polygon index for this object. This index can be used later with other per polygon based geometry functions.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note!
- All vertices must lay in the same plane otherwise behaviour may be unpredictable.
- The polygon is assumed to be convex. A non convex polygon will produce unpredictable behaviour.
- Polygons with zero area will be ignored.

Vertices of an object are in object space, not world space, and so are relative to the position, or center of the object. See Geometry::setPosition.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getNumPolygons
- Geometry::setPosition
- FMOD_VECTOR

Geometry::getActive

Retrieves the user set active state of the geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getActive(
    bool * active
);
```

Parameters

active

Address of a variable to receive the active state of the object. true = active, false = not active. Default = true.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::setActive

Geometry::getMaxPolygons

Retrieves the maximum number of polygons and vertices allocatable for this object. This is not the number of polygons or vertices currently present.
?The maximum number was set with System::createGeometry.?

Syntax

```
  FMOD_RESULT Geometry::getMaxPolygons(
  int * maxpolygons,
  int * maxvertices
);
```

Parameters

maxpolygons

Address of a variable to receieve the maximum possible number of polygons in this object.

maxvertices

Address of a variable to receieve the maximum possible number of vertices in this object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createGeometry
- System::loadGeometry

Geometry::getNumPolygons

Retrieves the number of polygons stored within this geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getNumPolygons(
  int * numpolygons
);
```

Parameters

numpolygons

Address of a variable to receive the number of polygons within this object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Polygons are added to a geometry object via Geometry::addPolygon.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::AddPolygon

Geometry::getPolygonAttributes

Retrieves the attributes for a particular polygon inside a geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getPolygonAttributes(
  int index,
   float *  directocclusion,
   float *  reverbocclusion,
   bool *  doublesided
);
```

Parameters

index

Polygon index inside the object.

directocclusion

Address of a variable to receieve the occlusion value from 0.0 to 1.0 which affects volume or audible frequencies. 0.0 = The polygon does not occlude volume or audible frequencies (sound will be fully audible), 1.0 = The polygon fully occludes (sound will be silent).

reverbocclusion

Address of a variable to receieve the occlusion value from 0.0 to 1.0 which affects the reverb mix. 0.0 = The polygon does not occlude reverb (reverb reflections still travel through this polygon), 1.0 = The polyfully fully occludes reverb (reverb reflections will be silent through this polygon).

doublesided

Address of a variable to receieve the description of polygon if it is double sided or single sided. true = polygon is double sided, false = polygon is single sided, and the winding of the polygon (which determines the polygon's normal) determines which side of the polygon will cause occlusion.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getPolygonAttributes
- Geometry::getNumPolygons

Geometry::getPolygonNumVertices

Gets the number of vertices in a polygon which is part of the geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getPolygonNumVertices(
  int  index,
  int *  numvertices
);
```

Parameters

index

Polygon index. This must be in the range of 0 to Geometry::getNumPolygons minus 1.

numvertices

Address of a variable to receive the number of vertices for the selected polygon.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getNumPolygons

Geometry::getPolygonVertex

Retrieves the position of the vertex inside a geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getPolygonVertex(
  int  index,
  int  vertexindex,
   FMOD_VECTOR *  vertex
);
```

Parameters

index

Polygon index. This must be in the range of 0 to Geometry::getNumPolygons minus 1.

vertexindex

Vertex index inside the polygon. This must be in the range of 0 to Geometry::getPolygonNumVertices minus 1.

vertex

Address of an FMOD_VECTOR structure which will receive the new vertex location in object space.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Vertices are relative to the position of the object. See Geometry::setPosition.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getPolygonNumVertices
- Geometry::setPosition
- Geometry::getNumPolygons
- FMOD_VECTOR

Geometry::getPosition

Retrieves the position of the object in 3D world space.?

Syntax

```
  FMOD_RESULT Geometry::getPosition(
    FMOD_VECTOR *  position
);
```

Parameters

position

Address of a variable to receive the 3d position of the object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::setPosition
- FMOD_VECTOR

Geometry::getRotation

Retrieves the orientation of the geometry object.?

Syntax

```
  FMOD_RESULT Geometry::getRotation(
    FMOD_VECTOR *  forward,
    FMOD_VECTOR *  up
);
```

Parameters

forward

Address of a variable that receives the forwards orientation of the geometry object. Specify 0 or NULL to ignore.

up

Address of a variable that receives the upwards orientation of the geometry object. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See remarks in System::set3DListenerAttributes for more description on forward and up vectors.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::setRotation
- System::set3DListenerAttributes
- FMOD_VECTOR

Geometry::getScale

Retrieves the relative scale vector of the geometry object. An object can be scaled/warped in all 3 dimensions separately using the vector.?

Syntax

```
  FMOD_RESULT Geometry::getScale(
    FMOD_VECTOR * scale
);
```

Parameters

scale

Address of a variable to receieve the scale vector of the object. Default = 1.0, 1.0, 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::setScale
- FMOD_VECTOR

Geometry::getUserData

Retrieves the user value that that was set by calling the Geometry::setUserData function.?

Syntax

```
  FMOD_RESULT Geometry::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the Geometry::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::setUserData

Geometry::release

Frees a geometry object and releases its memory.?

Syntax

```
FMOD_RESULT Geometry::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

Geometry::save

Saves the geometry object as a serialized binary block, to a user memory buffer. This can then be saved to a file if required and loaded later with System::loadGeometry.?

Syntax

```
  FMOD_RESULT Geometry::save(
   void *  data,
  int *  datasize
);
```

Parameters

data

Address of a variable to receive the serialized geometry object. Specify 0 or NULL to have the datasize parameter return the size of the memory required for this saved object.

datasize

Address of a variable to receive the size in bytes required to save this object when 'data' parameter is 0 or NULL.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

To use this function you will normally need to call it twice. Once to get the size of the data, then again to write the data to your pointer.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::loadGeometry
- System::createGeometry

Geometry::setActive

Enables or disables an object from being processed in the geometry engine.?

Syntax

```
  FMOD_RESULT Geometry::setActive(
   bool active
);
```

Parameters

active

true = active, false = not active. Default = true.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getActive

Geometry::setPolygonAttributes

Sets individual attributes for each polygon inside a geometry object.?

Syntax

```
 FMOD_RESULT Geometry::setPolygonAttributes(
 int index,
  float directocclusion,
  float reverbocclusion,
  bool doublesided
);
```

Parameters

index

Polygon index inside the object.

directocclusion

Occlusion value from 0.0 to 1.0 which affects volume or audible frequencies. 0.0 = The polygon does not occlude volume or audible frequencies (sound will be fully audible), 1.0 = The polygon fully occludes (sound will be silent).

reverbocclusion

Occlusion value from 0.0 to 1.0 which affects the reverb mix. 0.0 = The polygon does not occlude reverb (reverb reflections still travel through this polygon), 1.0 = The polyfully fully occludes reverb (reverb reflections will be silent through this polygon).

doublesided

Description of polygon if it is double sided or single sided. true = polygon is double sided, false = polygon is single sided, and the winding of the polygon (which determines the polygon's normal) determines which side of the polygon will cause occlusion.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getPolygonAttributes
- Geometry::getNumPolygons

Geometry::setPolygonVertex

Alters the position of a polygon's vertex inside a geometry object.?

Syntax

```
  FMOD_RESULT Geometry::setPolygonVertex(
  int index,
  int vertexindex,
  const FMOD_VECTOR * vertex
);
```

Parameters

index

Polygon index. This must be in the range of 0 to Geometry::getNumPolygons minus 1.

vertexindex

Vertex index inside the polygon. This must be in the range of 0 to Geometry::getPolygonNumVertices minus 1.

vertex

Address of an FMOD_VECTOR which holds the new vertex location.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! There may be some significant overhead with this function as it may cause some reconfiguration of internal data structures used to speed up sound-ray testing.
You may get better results if you want to modify your object by using Geometry::setPosition, Geometry::setScale and Geometry::setRotation.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getPolygonNumVertices

- Geometry::getPolygonNumVertices
- Geometry::setPosition
- Geometry::setScale
- Geometry::setRotation
- Geometry::getNumPolygons
- FMOD_VECTOR

Geometry::setPosition

Sets the position of the object in world space, which is the same space FMOD sounds and listeners reside in.?

Syntax

```
  FMOD_RESULT Geometry::setPosition(
  const FMOD_VECTOR *  position
);
```

Parameters

position

Pointer to a vector containing the 3d position of the object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getPosition
- Geometry::setRotation
- Geometry::setScale
- FMOD_VECTOR

Geometry::setRotation

Sets the orientation of the geometry object.?

Syntax

```
  FMOD_RESULT Geometry::setRotation(
  const FMOD_VECTOR *  forward,
  const FMOD_VECTOR *  up
);
```

Parameters

forward

The forwards orientation of the geometry object. This vector must be of unit length and perpendicular to the up vector. You can specify 0 or NULL to not update the forwards orientation of the geometry object.

up

The upwards orientation of the geometry object. This vector must be of unit length and perpendicular to the forwards vector. You can specify 0 or NULL to not update the upwards orientation of the geometry object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See remarks in System::set3DListenerAttributes for more description on forward and up vectors.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getRotation
- System::set3DListenerAttributes
- FMOD_VECTOR

Geometry::setScale

Sets the relative scale vector of the geometry object. An object can be scaled/warped in all 3 dimensions separately using the vector without having to modify polygon data.?

Syntax

```
  FMOD_RESULT Geometry::setScale(
  const FMOD_VECTOR * scale
);
```

Parameters

scale

The scale vector of the object. Default = 1.0, 1.0, 1.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getScale
- Geometry::setRotation
- Geometry::setPosition
- FMOD_VECTOR

Geometry::setUserData

Sets a user value that the Geometry object will store internally. Can be retrieved with Geometry::getUserData.?

Syntax

```
  FMOD_RESULT Geometry::setUserData(
   void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the Geometry object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using Geometry::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Geometry::getUserData

Reverb Interface

Reverb::get3DAttributes

Retrieves the 3d attributes of a Reverb object.?

Syntax

```
  FMOD_RESULT Reverb::get3DAttributes(
    FMOD_VECTOR *  position,
    float *  mindistance,
    float *  maxdistance
);
```

Parameters

position

Address of a variable that will receive the 3d position of the center of the reverb in 3d space. Default = { 0,0,0 }.

mindistance

Address of a variable that will receive the distance from the centerpoint that the reverb will have full effect at. Default = 0.0.

maxdistance

Address of a variable that will receive the distance from the centerpoint that the reverb will not have any effect. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::set3DAttributes
- System::createReverb

Reverb::getActive

Retrieves the active state of the reverb object.?

Syntax

```
  FMOD_RESULT Reverb::getActive(
    bool * active
);
```

Parameters

active

Address of a variable to receive the current active state of the reverb object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::setActive
- System::createReverb

Reverb::getProperties

Retrieves the current reverb environment.?

Syntax

```
  FMOD_RESULT Reverb::getProperties(
    FMOD_REVERB_PROPERTIES *  properties
);
```

Parameters

properties

Address of a variable that receives the current reverb environment description.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::setProperties
- System::createReverb

Reverb::getUserData

Retrieves the user value that that was set by calling the Reverb::setUserData function.?

Syntax

```
  FMOD_RESULT Reverb::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the Reverb::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::setUserData

Reverb::release

Releases the memory for a reverb object and makes it inactive.?

Syntax

```
FMOD_RESULT Reverb::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If no reverb objects are created, the ambient reverb will be the only audible reverb. By default this ambient reverb setting is set to OFF.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createReverb
- System::setReverbAmbientProperties

Reverb::set3DAttributes

Sets the 3d properties of a 'virtual' reverb object.?

Syntax

```
  FMOD_RESULT Reverb::set3DAttributes(
  const FMOD_VECTOR *  position,
   float  mindistance,
   float  maxdistance
);
```

Parameters

position

Pointer to a vector containing the 3d position of the center of the reverb in 3d space. Default = { 0,0,0 }.

mindistance

The distance from the centerpoint that the reverb will have full effect at. Default = 0.0.

maxdistance

The distance from the centerpoint that the reverb will not have any effect. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::get3DAttributes
- System::createReverb

Reverb::setActive

Disables or enables a reverb object so that it does or does not contribute to the 3d scene.?

Syntax

```
  FMOD_RESULT Reverb::setActive(
    bool  active
);
```

Parameters

active

true = active, false = not active. Default = true.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::setActive
- System::createReverb

Reverb::setProperties

Sets reverb parameters for the current reverb object.
?Reverb parameters can be set manually, or automatically using the pre-defined presets given in the fmod.h header.?

Syntax

```
  FMOD_RESULT Reverb::setProperties(
  const FMOD_REVERB_PROPERTIES *  properties
);
```

Parameters

properties

Address of an FMOD_REVERB_PROPERTIES structure which defines the attributes for the reverb.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_PROPERTIES
- Reverb::getProperties
- System::createReverb

Reverb::setUserData

Sets a user value that the Reverb object will store internally. Can be retrieved with Reverb::getUserData.?

Syntax

```
  FMOD_RESULT Reverb::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the Reverb object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using Reverb::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Reverb::getUserData

Functions

Debug_GetLevel

Retrieves the current debug logging level.?

Syntax

```
FMOD_RESULT Debug_GetLevel(
  FMOD_DEBUGLEVEL *  level
);
```

Parameters

level

Address of a variable to receieve current debug level.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This only has an effect with 'logging' versions of FMOD Ex. For example on windows it must be via fmodexL.dll, not fmodex.dll.
On Xbox it would be fmodxboxL.lib not fmodxbox.lib.
FMOD_ERR_UNSUPPORTED will be returned on non logging versions of FMOD Ex (ie full release).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Debug_GetLevel
- FMOD_DEBUGLEVEL

Debug_SetLevel

Sets the level of debug logging to the tty / output for logging versions of FMOD Ex.?

Syntax

```
  FMOD_RESULT Debug_SetLevel(
    FMOD_DEBUGLEVEL  level
);
```

Parameters

level

Logging level to set.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This only has an effect with 'logging' versions of FMOD Ex. For example on windows it must be via fmodexL.dll, not fmodex.dll.
On Xbox it would be fmodxboxL.lib not fmodxbox.lib.
FMOD_ERR_UNSUPPORTED will be returned on non logging versions of FMOD Ex (ie full release).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Debug_GetLevel
- FMOD_DEBUGLEVEL

File_GetDiskBusy

Callback for opening a file.?

Syntax

```
  FMOD_RESULT File_GetDiskBusy(
  int *  busy
);
```

Parameters

busy

Address of an integer to recieve the busy state of the disk at the current time.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Do not use this function to syncrhonize your own reads with, as due to timing, you might call this function and it says false = it is not busy, but the split second after call this function, internally FMOD might set it to busy. Use File_SetDiskBusy for proper mutual exclusion as it uses semaphores.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- File_SetDiskBusy

File_SetDiskBusy

Mutex function to synchronize user file reads with FMOD's file reads. This function tells fmod that you are using the disk so that it will?block until you are finished with it.
?This function also blocks if FMOD is already using the disk, so that you cannot do a read at the same time FMOD is reading.?

Syntax

```
  FMOD_RESULT File_SetDiskBusy(
  int  busy
);
```

Parameters

busy

1 = you are about to perform a disk access. 0 = you are finished with the disk.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this function as a wrapper around your own file reading functions if you want to do simulatenous file reading while FMOD is also reading. ie

```
 FMOD_File_SetDiskBusy(1);
myfread(...);
 FMOD_File_SetDiskBusy(0);
```

Warning! This is a critical section internally. If you do not match your busy = true with a busy = false your program may hang!
If you forget to set diskbusy to false it will stop FMOD from reading from the disk.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- File_GetDiskBusy

Memory_GetStats

Returns information on the memory usage of FMOD. This is useful for determining a fixed memory size to make FMOD work within for fixed memory machines such as consoles.?

Syntax

```
  FMOD_RESULT Memory_GetStats(
  int * currentalloced,
  int * maxalloced
);
```

Parameters

currentalloced

Address of a variable that receives the currently allocated memory at time of call. Optional. Specify 0 or NULL to ignore.

maxalloced

Address of a variable that receives the maximum allocated memory since System::init or Memory_Initialize. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that if using FMOD::Memory_Initialize, the memory usage will be slightly higher than without it, as FMOD has to have a small amount of memory overhead to manage the available memory.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- Memory_Initialize

Memory_Initialize

Specifies a method for FMOD to allocate memory, either through callbacks or its own internal memory management. You can also supply a pool of memory for FMOD to work with and it will do so with no extra calls to malloc or free.
?This is useful for systems that want FMOD to use their own memory management, or fixed memory devices such as Xbox, Xbox360, PS2 and GameCube that don't want any allocations occurring out of their control causing fragmentation or unpredictable overflows in a tight memory space.
?See remarks for more useful information.
?

Syntax

```
FMOD_RESULT Memory_Initialize(
  void * poolmem,
  int  poollen,
  FMOD_MEMORY_ALLOC_CALLBACK  useralloc,
  FMOD_MEMORY_REALLOC_CALLBACK  userrealloc,
  FMOD_MEMORY_FREE_CALLBACK  userfree
);
```

Parameters

poolmem

If you want a fixed block of memory for FMOD to use, pass it in here. Specify the length in poollen. Specifying NULL doesn't use internal management and it relies on callbacks.

poollen

Length in bytes of the pool of memory for FMOD to use specified in. Specifying 0 turns off internal memory management and relies purely on callbacks. Length must be a multiple of 512.

useralloc

Only supported if pool is NULL. Otherwise it overrides the FMOD internal calls to alloc. Compatible with ansi malloc().

userrealloc

Only supported if pool is NULL. Otherwise it overrides the FMOD internal calls to realloc. Compatible with ansi realloc().

userfree

Only supported if pool is NULL. Otherwise it overrides the FMOD internal calls to free. Compatible with ansi free().

Return Values

If the function succeeds then the return value is FMOD_OK.

If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

FMOD has been tested to stay in a limit and fail gracefully if the fixed pool size is not large enough with FMOD_ERR_MEMORY errors.
FMOD only does allocation when creating streams, music or samples and the FMOD_Init stage. It never allocates or deallocates memory during the course of runtime processing.
To find out the required fixed size the user can call FMOD::Memory_GetStats with a larger than necessary pool size (or no pool), and find out the maximum ram usage at any one time within FMOD.

FMOD behaves differently based on what you pass into this function in 3 different combinations.
Here are the examples.

```
FMOD::Memory_Initialize(NULL, 0,   NULL,    NULL,      NULL);   // Falls back purely to
ansi C malloc, realloc and free.
FMOD::Memory_Initialize(NULL, 0,   myalloc, myrealloc, myfree); // Calls user supplied
callbacks every time FMOD does a memory allocation or deallocation.
FMOD::Memory_Initialize(ptr,  len, NULL,    NULL,      NULL);   // Uses "ptr" and
manages memory internally.  NO extra mallocs or frees are performed from this point.
```

Callbacks and memory pools cannot be combined, as if a pool is specified FMOD, manipulates the pool of memory internally with its own allocate and free scheme.
The memory management algorithm to work within a fixed size of ram is extremely efficient and faster than the standard C malloc or free.

On Xbox 1 you MUST specify a pointer and length. The memory provided must be enough to store all sample data.

NOTE! Your memory callbacks must be thread safe. If not unexpected behaviour may occur. FMOD calls memory allocation functions from asynchronous threads, such as the thread related to FMOD_NONBLOCKING flag, and sometimes from the mixer thread.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MEMORY_ALLOCCALLBACK
- FMOD_MEMORY_REALLOCCALLBACK
- FMOD_MEMORY_FREECALLBACK
- Memory_GetStats
- System::close

System_Create

FMOD System creation function. This must be called to create an FMOD System object before you can do anything else.?Use this function to create 1, or multiple instances of FMOD System objects.?

Syntax

```
  FMOD_RESULT System_Create(
   FMOD::System ** system
);
```

Parameters

system

Address of a pointer that receives the new FMOD System object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use System::release to free a system object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- System::release

Callbacks

FMOD_3D_ROLLOFFCALLBACK

Callback for system wide 3d channel volume calculation which overrides fmod's internal calculation code.?

Syntax

```
  float F_CALLBACK FMOD_3D_ROLLOFFCALLBACK(
    FMOD_CHANNEL * channel,
    float  distance
);
```

Parameters

channel

Pointer to a channel handle.

distance

Distance in units (meters by default).

Return Values

Remarks

C++ Users. Cast **FMOD_CHANNEL *** to **FMOD::Channel *** inside the callback and use as normal.

NOTE: When using the event system, call Channel::getUserData to get the event instance handle of the event that spawned the channel in question.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DRolloffCallback
- System::set3DListenerAttributes
- System::get3DListenerAttributes
- Channel::getUserData

FMOD_CHANNEL_CALLBACK

Callback for channel events.?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_CHANNEL_CALLBACK(
  FMOD_CHANNEL * channel,
  FMOD_CHANNEL_CALLBACKTYPE type,
  int command,
  unsigned int commanddata1,
  unsigned int commanddata2
);
```

Parameters

channel

Pointer to a channel handle.

type

The type of callback. Refer to FMOD_CHANNEL_CALLBACKTYPE.

command

The command value passed into Channel::setCallback.

commanddata1

The first callback type specific data generated by the callback. See remarks for meaning.

commanddata2

The second callback type specific data generated by the callback. See remarks for meaning.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_CHANNEL *** to **FMOD::Channel *** inside the callback and use as normal.

'commanddata1' and 'commanddata2' meanings.

These 2 values are set by the callback depending on what is happening in the callback and the type of callback.

- **FMOD_CHANNEL_CALLBACKTYPE_END**

commanddata1: Always 0.
commanddata2: Always 0.

- **FMOD_CHANNEL_CALLBACKTYPE_VIRTUALVOICE**

commanddata1: **0** when voice is swapped from emulated to real. **1** when voice is swapped from real to emulated.
commanddata2: Always 0.

- **FMOD_CHANNEL_CALLBACKTYPE_SYNCPOINT**

commanddata1: The index of the sync point. Use Sound::getSyncPointInfo to retrieve the sync point's attributes.
commanddata2: Always 0.

Note! Currently the user must call System::update for these callbacks to trigger!

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setCallback
- FMOD_CHANNEL_CALLBACKTYPE
- System::update

FMOD_CODEC_CLOSECALLBACK

Close callback for the codec for when FMOD tries to close a sound using this codec.
?This is the callback any codec related memory is freed, and things are generally de-initialized / shut down for the codec.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_CLOSECALLBACK (
   FMOD_CODEC_STATE  * codec_state
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT .

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK

- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_GETLENGTHCALLBACK

Callback to return the length of the song in whatever format required when Sound::getLength is called.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_GETLENGTHCALLBACK(
   FMOD_CODEC_STATE * codec_state,
  unsigned int * length,
   FMOD_TIMEUNIT lengthtype
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

length

Address of a variable that is to receive the length of the sound determined by the format specified in the lengttype parameter.

lengthtype

Timeunit type of length to return. This will be one of the timeunits supplied by the codec author in the FMOD_CODEC_DESCRIPTION structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT .

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_TIMEUNIT
- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_GETPOSITIONCALLBACK

Tell callback for the codec for when FMOD tries to get the current position within the with Channel::getPosition.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_GETPOSITIONCALLBACK(
    FMOD_CODEC_STATE * codec_state,
  unsigned int * position,
    FMOD_TIMEUNIT postype
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

position

Address of a variable to receive the current position in the codec based on the timeunit specified in the postype parameter.

postype

Timeunit type of the position parameter that is requested. This will be one of the timeunits supplied by the codec author in the FMOD_CODEC_DESCRIPTION structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT .

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- Channel::getPosition
- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_METADATACALLBACK

Callback for sounds that have their?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_CODEC_METADATACALLBACK(
  FMOD_CODEC_STATE * codec_state,
  FMOD_TAGTYPE type,
 char * name,
  void * data,
 unsigned int datalen,
  FMOD_TAGDATATYPE datatype,
 int unique
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

type

Source of tag being updated, ie id3v2 or oggvorbis tag for example. See FMOD_TAGDATATYPE.

name

Name of the tag being updated.

data

Contents of tag.

datalen

Length of the tag data in bytes.

datatype

Data type of tag. Binary / string / unicode etc. See FMOD_TAGDATATYPE.

unique

If this is true, then the tag (determined by the name) being updated is the only one of its type. If it is false then there are multiple versions of this tag with the same name.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This callback is usually called from sounds that can udate their metadata / tag info at runtime. Such a sound could be an internet SHOUTcast / Icecast stream for example.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK
- FMOD_TAGDATATYPE

FMOD_CODEC_OPENCALLBACK

Open callback for the codec for when FMOD tries to open a sound using this codec.?This is the callback the file format check is done in, codec related memory is allocated, and things are generally initialized / set up for the codec.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_OPENCALLBACK(
    FMOD_CODEC_STATE * codec_state,
    FMOD_MODE usermode,
    FMOD_CREATESOUNDEXINFO * userexinfo
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

usermode

Mode that the user supplied via System::createSound. This is informational and can be ignored, or used if it has relevance to your codec.

userexinfo

Extra info structure that the user supplied via System::createSound. This is informational and can be ignored, or used if it has relevance to your codec.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The usermode and userexinfo parameters tell the codec what was passed in by the user.
Generally these can be ignored, as the file format usually determines the format and frequency of the sound.

If you have a flexible format codec (ie you don't mind what output format your codec writes to), you might want to use the parameter that was passed in by the user to specify the output sound format / frequency.
For example if you normally create a codec that is always 32bit floating point, the user might supply 16bit integer to save memory, so you could use this information to decode your data to this format instead of the original default format.
Read and seek within the file using the 'fileread' and 'fileseek' members of the FMOD_CODEC codec that is passed

in.
Note: **DO NOT USE YOUR OWN FILESYSTEM.**
The reasons for this are:

- The user may have set their own file system via user filesystem callbacks.
- FMOD allows file reading via disk, memory and TCP/IP. If you use your own file routines you will lose this ability.

Important! FMOD will ping all codecs trying to find the right one for the file the user has passed in. Make sure the first line of your codec open is a FAST format check. Ie it reads an identifying string, checks it and returns an error FMOD_ERR_FORMAT if it is not found.
There may be a lot of codecs loaded into FMOD, so you don't want yours slowing down the System::createSound call because it is inneficient in determining if it is the right format or not.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::createSound
- FMOD_CREATESOUNDEXINFO
- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_READCALLBACK

Read callback for the codec for when FMOD tries to read some data from the file to the destination format (format specified in the open callback).?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_READCALLBACK(
   FMOD_CODEC_STATE * codec_state,
   void * buffer,
  unsigned int sizebytes,
  unsigned int * bytesread
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

buffer

Buffer to read PCM data to. Note that the format of this data is the format described in FMOD_CODEC_WAVEFORMAT.

sizebytes

Number of bytes to read

bytesread

Number of bytes actually read

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If you cannot read number of bytes requested, simply return FMOD_OK and give bytesread the number of bytes you read.
Read and seek within the file using the 'fileread' and 'fileseek' members of the FMOD_CODEC codec that is passed in.
Note: **DO NOT USE YOUR OWN FILESYSTEM.**

The reasons for this are:

- The user may have set their own file system via user filesystem callbacks.
- FMOD allows file reading via disk, memory and TCP/IP. If you use your own file routines you will lose this ability.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_SETPOSITIONCALLBACK

Seek callback for the codec for when FMOD tries to seek within the file with Channel::setPosition.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_CODEC_SETPOSITIONCALLBACK(
   FMOD_CODEC_STATE * codec_state,
  int subsound,
  unsigned int position,
   FMOD_TIMEUNIT postype
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

subsound

Subsound within which to seek.

position

Position to seek to in the sound based on the timeunit specified in the postype parameter.

postype

Timeunit type of the position parameter. This will be one of the timeunits supplied by the codec author in the FMOD_CODEC_DESCRIPTION structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Read and seek within the file using the 'fileread' and 'fileseek' members of the FMOD_CODEC codec that is passed in.
Note: **DO NOT USE YOUR OWN FILESYSTEM.**
The reasons for this are:

- The user may have set their own file system via user filesystem callbacks.
- FMOD allows file reading via disk, memory and TCP/IP. If you use your own file routines you will lose this

ability.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- Channel::setPosition
- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK
- FMOD_CODEC_SOUNDCREATECALLBACK

FMOD_CODEC_SOUNDCREATECALLBACK

Sound creation callback for the codec when FMOD finishes creating the sound. Ie so the codec can set more parameters for the related created sound, ie loop points/mode or 3D attributes etc.?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_CODEC_SOUNDCREATECALLBACK (
  FMOD_CODEC_STATE * codec_state,
  int subsound,
  FMOD_SOUND * sound
);
```

Parameters

codec_state

Pointer to the codec state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

subsound

Subsound index being created.

sound

Pointer to the sound being created.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::createSound
- System::createStream
- FMOD_CODEC_STATE
- FMOD_CODEC_DESCRIPTION
- FMOD_CODEC_OPENCALLBACK
- FMOD_CODEC_CLOSECALLBACK
- FMOD_CODEC_READCALLBACK
- FMOD_CODEC_GETLENGTHCALLBACK
- FMOD_CODEC_SETPOSITIONCALLBACK
- FMOD_CODEC_GETPOSITIONCALLBACK

FMOD_DSP_CREATECALLBACK

This callback is called once when a user creates a DSP unit of this type. It is used to allocate memory, initialize variables and the like.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_CREATECALLBACK (
    FMOD_DSP_STATE *  dsp_state
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
System::createDSP
System::createDSPByType
System::createDSPByIndex
Sometimes a user will re-use a DSP unit instead of releasing it and creating a new one, so it may be useful to implement FMOD_DSP_RESETCALLBACK to reset any variables or buffers when the user calls it.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- System::createDSP
- System::createDSPByType
- System::createDSPByIndex
- FMOD_DSP_RESETCALLBACK

FMOD_DSP_DIALOGCALLBACK

This callback is called when the user wants the plugin to display a configuration dialog box. This is not always nescessary, so this can be left blank if wanted.?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_DSP_DIALOGCALLBACK(
  FMOD_DSP_STATE * dsp_state,
 int show,
  void * hwnd
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

show

1 = show the dialog, 0 = hide/remove the dialog.

hwnd

This is the target hwnd to display the dialog in. It must not pop up on this hwnd, it must actually be drawn within it.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
DSP::showConfigDialog.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::showConfigDialog

FMOD_DSP_GETPARAMCALLBACK

This callback is called when the user wants to get an indexed parameter from a DSP unit.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_GETPARAMCALLBACK(
   FMOD_DSP_STATE * dsp_state,
  int index,
   float * value,
  char * valuestr
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

index

The index into the parameter list for the parameter the user wants to get.

value

Pointer to a floating point variable to receive the selected parameter value.

valuestr

A pointer to a string to receive the value of the selected parameter, but in text form. This might be useful to display words instead of numbers. For example "ON" or "OFF" instead of 1.0 and 0.0. The length of the buffer being passed in is always 16 bytes, so do not exceed this.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
DSP::getParameter.
FMOD_DSP_GETPARAMCALLBACK.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::getParameter
- FMOD_DSP_SETPARAMCALLBACK

FMOD_DSP_READCALLBACK

This callback is called back regularly when the unit has been created, inserted to the DSP network, and set to active by the user.
?This callback requires the user to fill the output pointer with data. Incoming data is provided and may be filtered on its way to the output pointer.
?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_DSP_READCALLBACK (
  FMOD_DSP_STATE *  dsp_state,
  float * inbuffer,
  float * outbuffer,
 unsigned int  length,
 int inchannels,
 int outchannels
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

inbuffer

Pointer to incoming floating point -1.0 to +1.0 ranged data.

outbuffer

Pointer to outgoing floating point -1.0 to +1.0 ranged data. The dsp writer must write to this pointer else there will be silence.

length

The length of the incoming and outgoing buffer in samples. To get the length of the buffer in bytes, the user must multiply this number by the number of channels coming in (and out, they may be different) and then multiply by 4 for 1 float = 4 bytes.

inchannels

The number of channels of interleaved PCM data in the inbuffer parameter. A mono signal coming in would be 1. A stereo signal coming in would be 2.

outchannels

The number of channels of interleaved PCM data in the outbuffer parameter. A mono signal going out would be 1. A stereo signal going out would be 2.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
None.
This callback is called automatically and periodically when the DSP engine updates.
For a read update to be called it would have to be enabled, and this is done with DSP::setActive.
Data passed into the callback is always floating point, and of the range -1.0 to +1.0. This is a soft limit though, because FMOD will clip it to these ranges in the final stage of the pipeline, so the dsp unit writer does not have to worry about this.

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::setActive

FMOD_DSP_RELEASECALLBACK

This callback is called when the user releases the DSP unit. It is used to free any resources allocated during the course of the lifetime of the DSP or perform any shut down code needed to clean up the DSP unit.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_RELEASECALLBACK(
   FMOD_DSP_STATE *  dsp_state
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
DSP::release

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::release

FMOD_DSP_RESETCALLBACK

This callback function is called by DSP::reset to allow the effect to reset itself to a default state.
?This is useful if an effect is for example still holding audio data for a sound that has stopped, and the unit wants to be relocated to a new sound. Resetting the unit would clear any buffers, put the effect back to its initial state, and get it ready for new sound data.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_RESETCALLBACK(
   FMOD_DSP_STATE * dsp_state
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
DSP::reset

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT .

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::reset

Version 4.12.03 Built on Feb 18, 2008

FMOD_DSP_SETPARAMCALLBACK

This callback is called when the user wants to set a parameter for a DSP unit.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_SETPARAMCALLBACK(
    FMOD_DSP_STATE * dsp_state,
  int index,
    float value
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

index

The index into the parameter list for the parameter the user wants to set.

value

The value passed in by the user to set for the selected parameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
DSP::setParameter.

Range checking is not needed. FMOD will clamp the incoming value to the specified min/max.
Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- DSP::setParameter
- FMOD_DSP_GETPARAMCALLBACK

FMOD_DSP_SETPOSITIONCALLBACK

Callback that is called when the user sets the position of a channel with Channel::setPosition.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_DSP_SETPOSITIONCALLBACK (
   FMOD_DSP_STATE * dsp_state,
  unsigned int position
);
```

Parameters

dsp_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data. Do not cast this to FMOD_DSP! The handle to the user created DSP handle is stored within the FMOD_DSP_STATE structure.

position

Position in channel stream to set to. Units are PCM samples (ie FMOD_TIMEUNIT_PCM).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Functions that the user would have to call for this callback to be called.
Channel::setPosition.
If a DSP unit is attached to a channel and the user calls Channel::setPosition then this funciton will be called.
Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_DSP_STATE
- Channel::setPosition

FMOD_FILE_CLOSECALLBACK

Calback for closing a file.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_FILE_CLOSECALLBACK(
    void * handle,
    void * userdata
);
```

Parameters

handle

This is the handle returned from the open callback to use for your own file routines.

userdata

Userdata initialized in the FMOD_FILE_OPENCALLBACK.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Close any user created file handle and perform any cleanup nescessary for the file here. If the callback is from System::attachFileSystem, then the return value is ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::setFileSystem
- System::attachFileSystem
- FMOD_FILE_OPENCALLBACK
- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK

Version 4.12.03 Built on Feb 18, 2008

FMOD_FILE_OPENCALLBACK

Callback for opening a file.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_FILE_OPENCALLBACK(
  const char *  name,
  int unicode,
  unsigned int *  filesize,
   void **  handle,
   void ** userdata
);
```

Parameters

name

This is the filename passed in by the user. You may treat this as you like.

unicode

Tells the callback if the string being passed in is a double byte unicode string or not. You may have to support this unless you know the target application will not support unicode.

filesize

The size of the file to be passed back to fmod, in bytes.

handle

This is to store a handle generated by the user. This will be the handle that gets passed into the other callbacks. Optional but may be needed.

userdata

This is to store userdata to be passed into the other callbacks. Optional.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Return the appropriate error code such as FMOD_ERR_FILE_NOTFOUND if the file fails to open. If the callback is from System::attachFileSystem, then the return value is ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::setFileSystem
- System::attachFileSystem
- FMOD_FILE_CLOSECALLBACK
- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK

FMOD_FILE_READCALLBACK

Callback for reading from a file.?

Syntax

```
 FMOD_RESULT F_CALLBACK FMOD_FILE_READCALLBACK(
  void *  handle,
  void *  buffer,
 unsigned int sizebytes,
 unsigned int *  bytesread,
  void * userdata
);
```

Parameters

handle

This is the handle you returned from the open callback to use for your own file routines.

buffer

The buffer to read your data into.

sizebytes

The number of bytes to read.

bytesread

The number of bytes successfully read.

userdata

Userdata initialized in the FMOD_FILE_OPENCALLBACK.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If the callback is from System::attachFileSystem, then the return value is ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::setFileSystem
- System::attachFileSystem
- FMOD_FILE_OPENCALLBACK
- FMOD_FILE_CLOSECALLBACK
- FMOD_FILE_SEEKCALLBACK

FMOD_FILE_SEEKCALLBACK

Callback for seeking within a file.?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_FILE_SEEKCALLBACK(
  void * handle,
  unsigned int pos,
  void * userdata
);
```

Parameters

handle

This is the handle returned from the open callback to use for your own file routines.

pos

This is the position or offset to seek to in the file in bytes.

userdata

Data initialized in the FMOD_FILE_OPENCALLBACK.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::setFileSystem
- FMOD_FILE_OPENCALLBACK
- FMOD_FILE_CLOSECALLBACK
- FMOD_FILE_READCALLBACK

FMOD_MEMORY_ALLOCCALLBACK

Callback to allocate a block of memory.?

Syntax

```
  void * F_CALLBACK FMOD_MEMORY_ALLOCCALLBACK(
  unsigned int  size,
   FMOD_MEMORY_TYPE  type
);
```

Parameters

size

Size in bytes of the memory block to be allocated and returned.

type

Type of memory allocation.

Return Values

On success, a pointer to the newly allocated block of memory is returned.
On failure, NULL is returned.

Remarks

Returning an aligned pointer, of 16 byte alignment is recommended for speed purposes.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Memory_Initialize
- Memory_GetStats
- FMOD_MEMORY_REALLOCCALLBACK
- FMOD_MEMORY_FREECALLBACK
- FMOD_MEMORY_TYPE

FMOD_MEMORY_FREECALLBACK

Callback to free a block of memory.?

Syntax

```
  void F_CALLBACK FMOD_MEMORY_FREECALLBACK(
    void * ptr,
    FMOD_MEMORY_TYPE  type
);
```

Parameters

ptr

Pointer to a pre-existing block of memory to be freed.

type

Type of memory to be freed.

Return Values

void

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Memory_Initialize
- Memory_GetStats
- FMOD_MEMORY_ALLOCCALLBACK
- FMOD_MEMORY_REALLOCCALLBACK
- FMOD_MEMORY_TYPE

FMOD_MEMORY_REALLOCCALLBACK

Callback to re-allocate a block of memory to a different size.?

Syntax

```
  void * F_CALLBACK  FMOD_MEMORY_REALLOCCALLBACK(
    void *  ptr,
  unsigned int size,
    FMOD_MEMORY_TYPE   type
);
```

Parameters

ptr

Pointer to a block of memory to be resized. If this is NULL then a new block of memory is simply allocated.

size

Size of the memory to be reallocated. The original memory must be preserved.

type

Type of memory allocation.

Return Values

On success, a pointer to the newly re-allocated block of memory is returned.
On failure, NULL is returned.

Remarks

Returning an aligned pointer, of 16 byte alignment is recommended for speed purposes.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Memory_Initialize

- Memory_GetStats
- FMOD_MEMORY_ALLOCCALLBACK
- FMOD_MEMORY_FREECALLBACK
- FMOD_MEMORY_TYPE

FMOD_OUTPUT_CLOSECALLBACK

Shut down callback which is called when the user calls System::close or System::release. (System::release calls System::close internally)?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_OUTPUT_CLOSECALLBACK (
  FMOD_OUTPUT_STATE * output_state
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::release
- System::close

FMOD_OUTPUT_GETDRIVERCAPSCALLBACK

Called when the user calls System::getDriverCaps.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_GETDRIVERCAPSCALLBACK(
    FMOD_OUTPUT_STATE * output_state,
  int id,
    FMOD_CAPS * caps,
  int * minfrequency,
  int * maxfrequency,
    FMOD_SPEAKERMODE * controlpanelspeakermode
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

id

Index into the total number of outputs possible, provided by the FMOD_OUTPUT_GETNUMDRIVERSCALLBACK callback.

caps

Address of a variable to receive the caps available by this output device. See FMOD_CAPS. Fill this in.

minfrequency

maxfrequency

controlpanelspeakermode

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::getDriverCaps
- System::getDriverInfo
- System::getNumDrivers
- FMOD_OUTPUT_GETNUMDRIVERSCALLBACK

FMOD_OUTPUT_GETDRIVERNAMECALLBACK

Called when the user calls System::getDriverInfo.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_GETDRIVERNAMECALLBACK(
    FMOD_OUTPUT_STATE * output_state,
  int id,
  char * name,
  int namelen
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

id

Index into the total number of outputs possible, provided by the FMOD_OUTPUT_GETNUMDRIVERSCALLBACK callback.

name

Address of a variable to receive the driver name relevant to the index passed in. Fill this in.

namelen

Length of name buffer being passed in by the user.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::getDriverInfo
- System::getNumDrivers
- FMOD_OUTPUT_GETNUMDRIVERSCALLBACK

FMOD_OUTPUT_GETHANDLECALLBACK

Called when the user calls System::getOutputHandle.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_GETHANDLECALLBACK (
    FMOD_OUTPUT_STATE * output_state,
    void ** handle
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

handle

Address of a variable to receieve the current plugin's output 'handle'. This is only if the plugin writer wants to allow the user access to the main handle behind the plugin (for example the file handle in a file writer plugin). The pointer type must be published to the user somehow, as is done in fmod.h.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

FMOD_OUTPUT_GETNUMDRIVERSCALLBACK

Called when the user calls System::getNumDrivers.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_GETNUMDRIVERSCALLBACK(
   FMOD_OUTPUT_STATE * output_state,
  int * numdrivers
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

numdrivers

Address of a variable to receive the number of output drivers in your plugin.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.
Optional. FMOD will assume 0 if this is not specified.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::getNumDrivers
- System::getDriverInfo
- FMOD_OUTPUT_GETDRIVERNAMECALLBACK

FMOD_OUTPUT_GETPOSITIONCALLBACK

Returns the current PCM offset or playback position for the output stream.
?Called from the mixer thread, only when the 'polling' member of FMOD_OUTPUT_DESCRIPTION is set to **true**.
?The internal FMOD output thread calls this function periodically to determine if it should ask for a block of audio data or not.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_GETPOSITIONCALLBACK(
    FMOD_OUTPUT_STATE *  output_state,
  unsigned int *  pcm
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

pcm

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_OUTPUT_DESCRIPTION
- FMOD_OUTPUT_LOCKCALLBACK
- FMOD_OUTPUT_UNLOCKCALLBACK

FMOD_OUTPUT_INITCALLBACK

Initialization callback which is called when the user calls System::init.?

Syntax

```
FMOD_RESULT F_CALLBACK FMOD_OUTPUT_INITCALLBACK(
  FMOD_OUTPUT_STATE * output_state,
  int selecteddriver,
  FMOD_INITFLAGS flags,
  int * outputrate,
  int outputchannels,
  FMOD_SOUND_FORMAT * outputformat,
  int dspbufferlength,
  int dspnumbuffers,
  void * extradriverdata
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

selecteddriver

This is the selected driver id that the user chose from calling System::setDriver.

flags

Initialization flags passed in by the user.

outputrate

Output rate selected by the user. If not possible, change the rate to the closest match.

outputchannels

Output channel count selected by the user. For example 1 = mono output. 2 = stereo output.

outputformat

Output format specified by the user. If not possible to support, return FMOD_ERR_FORMAT.

dspbufferlength

Size of the buffer fmod will mix to in one mix update. This value is in PCM samples.

dspnumbuffers

Number of buffers fmod will mix to in a circular fashion. Multiply this by dspbufferlength to get the total size of the output sound buffer to allocate.

extradriverdata

Data passed in by the user specific to this driver. May be used for any purpose.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT
.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_RESULT
- System::init
- System::setDriver

FMOD_OUTPUT_LOCKCALLBACK

Called from the mixer thread, only when the 'polling' member of FMOD_OUTPUT_DESCRIPTION is set to true.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_LOCKCALLBACK (
   FMOD_OUTPUT_STATE * output_state,
  unsigned int offset,
  unsigned int length,
   void ** ptr1,
   void ** ptr2,
  unsigned int * len1,
  unsigned int * len2
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

offset

Offset in *bytes* to the position the caller wants to lock in the sample buffer.

length

Number of *bytes* the caller want to lock in the sample buffer.

ptr1

Address of a pointer that will point to the first part of the locked data.

ptr2

Address of a pointer that will point to the second part of the locked data. This will be null if the data locked hasn't wrapped at the end of the buffer.

len1

Length of data in *bytes* that was locked for ptr1

len2

Length of data in *bytes* that was locked for ptr2. This will be 0 if the data locked hasn't wrapped at the end of the buffer.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_OUTPUT_DESCRIPTION
- FMOD_OUTPUT_UNLOCKCALLBACK
- FMOD_OUTPUT_GETPOSITIONCALLBACK

FMOD_OUTPUT_READFROMMIXER

Called by the plugin, when the 'polling' member of FMOD_OUTPUT_DESCRIPTION is set to false.
?Use this function from your own driver irq/timer to read some data from FMOD's DSP engine. All of the resulting output caused by playing sounds and specifying effects by the user will be mixed here and written to the memory provided by the plugin writer.
?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_READFROMMIXER(
   FMOD_OUTPUT_STATE * output_state,
   void * buffer,
  unsigned int length
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

buffer

Plugin-writer provided memory for the FMOD Ex mixer to write to.

length

Length of the buffer in samples.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

FMOD_OUTPUT_UNLOCKCALLBACK

Called from the mixer thread, only when the 'polling' member of FMOD_OUTPUT_DESCRIPTION is set to true.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_UNLOCKCALLBACK(
   FMOD_OUTPUT_STATE  * output_state,
   void *  ptr1,
   void *  ptr2,
  unsigned int  len1,
  unsigned int  len2
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

ptr1

Pointer to the 1st locked portion of sample data, from Sound::lock.

ptr2

Pointer to the 2nd locked portion of sample data, from Sound::lock.

len1

Length of data in *bytes* that was locked for ptr1

len2

Length of data in *bytes* that was locked for ptr2. This will be 0 if the data locked hasn't wrapped at the end of the buffer.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is normally called after data has been read/written to from Sound::lock. This function will do any post processing nescessary and if needed, send it to sound ram.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_OUTPUT_DESCRIPTION
- FMOD_OUTPUT_LOCKCALLBACK
- FMOD_OUTPUT_GETPOSITIONCALLBACK

FMOD_OUTPUT_UPDATECALLBACK

Called when the user calls System::update.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_OUTPUT_UPDATECALLBACK (
    FMOD_OUTPUT_STATE * output_state
);
```

Parameters

output_state

Pointer to the plugin state. The user can use this variable to access runtime plugin specific variables and plugin writer user data.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Remember to return FMOD_OK at the bottom of the function, or an appropriate error code from FMOD_RESULT.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

FMOD_SOUND_NONBLOCKCALLBACK

Callback to be called when a sound has finished loading.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_SOUND_NONBLOCKCALLBACK(
    FMOD_SOUND * sound,
    FMOD_RESULT result
);
```

Parameters

sound

Pointer to the sound. C++ users see remarks.

result

Error code. FMOD_OK if sound was created successfully, or an error code otherwise.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_SOUND *** to **FMOD::Sound *** inside the callback and use as normal.

Return code currently ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- System::createSound
- FMOD_CREATESOUNDEXINFO

FMOD_SOUND_PCMREADCALLBACK

Used for 2 purposes.
?One use is for user created sounds when FMOD_OPENUSER is specified when creating the sound.
?The other use is to 'piggyback' on FMOD's read functions when opening a normal sound, therefore the callee can read (rip) or even write back new PCM data while FMOD is opening the sound.?

Syntax

```
  FMOD_RESULT F_CALLBACK  FMOD_SOUND_PCMREADCALLBACK(
   FMOD_SOUND *  sound,
   void *  data,
  unsigned int  datalen
);
```

Parameters

sound

Pointer to the sound. C++ users see remarks.

data

Pointer to raw PCM data that the user can either read or write to.

datalen

Length of the data in bytes.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_SOUND *** to **FMOD::Sound *** inside the callback and use as normal.

The format of the sound can be retrieved with Sound::getFormat from this callback. This will allow the user to determine what type of pointer to use if they are not sure what format the sound is.
If the callback is used for the purpose of 'piggybacking' normal FMOD sound loads, then you do not have to do anything at all, and it can be treated as purely informational. The return value is also ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- Sound::getFormat
- FMOD_SOUND_PCMSETPOSCALLBACK
- System::createSound
- System::createStream
- FMOD_CREATESOUNDEXINFO

FMOD_SOUND_PCMSETPOSCALLBACK

Callback for when the caller calls a seeking function such as Channel::setTime or Channel::setPosition.
?If the sound is a user created sound, this can be used to seek within the user's resource.
?

Syntax

```
  FMOD_RESULT F_CALLBACK  FMOD_SOUND_PCMSETPOSCALLBACK(
    FMOD_SOUND *  sound,
  int  subsound,
  unsigned int   position,
    FMOD_TIMEUNIT   postype
);
```

Parameters

sound

Pointer to the sound. C++ users see remarks.

subsound

In a multi subsound type sound (ie fsb/dls/cdda), this will contain the index into the list of sounds.

position

Position to seek to that has been requested. This value will be of format FMOD_TIMEUNIT and must be parsed to determine what it is. Generally FMOD_TIMEUNIT_PCM will be the most common format.

postype

Position type that the user wanted to seek with. If the sound is a user create sound and the seek type is unsupported return FMOD_ERR_FORMAT.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_SOUND *** to **FMOD::Sound *** inside the callback and use as normal.

If the callback is used for the purpose of 'piggybacking' normal FMOD sound loads, then you do not have to do anything at all, and it can be treated as purely informational. The return value is also ignored.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- FMOD_SOUND_PCMREADCALLBACK
- System::createSound
- System::createStream
- FMOD_CREATESOUNDEXINFO

FMOD_SYSTEM_CALLBACK

Callback for system events.?

Syntax

```
  FMOD_RESULT F_CALLBACK FMOD_SYSTEM_CALLBACK(
   FMOD_SYSTEM * system,
   FMOD_SYSTEM_CALLBACKTYPE  type,
  unsigned int commanddata1,
  unsigned int commanddata2
);
```

Parameters

system

Pointer to a system handle.

type

The type of callback. Refer to FMOD_SYSTEM_CALLBACKTYPE.

commanddata1

The first callback type specific data generated by the callback. See remarks for meaning.

commanddata2

The second callback type specific data generated by the callback. See remarks for meaning.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_SYSTEM *** to **FMOD::System *** inside the callback and use as normal.

'commanddata1' and 'commanddata2' meanings.
These 2 values are set by the callback depending on what is happening in the callback and the type of callback.

- **FMOD_SYSTEM_CALLBACKTYPE_DEVICELISTCHANGED**

commanddata1: Always 0.
commanddata2: Always 0.

- **FMOD_SYSTEM_CALLBACKTYPE_MEMORYALLOCATIONFAILED**

commanddata1: A string (char*) which represents the file and line number of the allocation.
commanddata2: The size (int) of the requested allocation.

Note! Currently the user must call System::update for some of these callbacks to trigger! See FMOD_SYSTEM_CALLBACKTYPE for details.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setCallback
- FMOD_SYSTEM_CALLBACKTYPE
- System::update

Structures

FMOD_ADVANCEDSETTINGS

Settings for advanced features like configuring memory and cpu usage for the FMOD_CREATECOMPRESSEDSAMPLE feature.?

Structure

```
typedef struct {
  int cbsize;
  int maxMPEGcodecs;
  int maxADPCMcodecs;
  int maxXMAcodecs;
  int maxPCMcodecs;
  int ASIONumChannels;
  char ** ASIOChannelList;
   FMOD_SPEAKER * ASIOSpeakerList;
  int max3DReverbDSPs;
   float HRTFMinAngle;
   float HRTFMaxAngle;
   float HRTFFreq;
   float vol0virtualvol;
  int eventqueuesize;
  unsigned int defaultDecodeBufferSize;
} FMOD_ADVANCEDSETTINGS;
```

Members

cbsize

[in] Size of this structure. Use sizeof(FMOD_ADVANCEDSETTINGS) NOTE: This must be set before calling System::getAdvancedSettings!

maxMPEGcodecs

[in/out] Optional. Specify 0 to ignore. For use with FMOD_CREATECOMPRESSEDSAMPLE only. Mpeg codecs consume 29,424 bytes per instance and this number will determine how many mpeg channels can be played simultaneously. Default = 16.

maxADPCMcodecs

[in/out] Optional. Specify 0 to ignore. For use with FMOD_CREATECOMPRESSEDSAMPLE only. ADPCM codecs consume 2,136 bytes per instance (based on FSB encoded ADPCM block size - see remarks) and this number will determine how many ADPCM channels can be played simultaneously. Default = 32.

maxXMAcodecs

[in/out] Optional. Specify 0 to ignore. For use with FMOD_CREATECOMPRESSEDSAMPLE only. XMA codecs consume 20,512 bytes per instance and this number will determine how many XMA channels can be played simultaneously. Default = 32.

maxPCMcodecs

[in/out] Optional. Specify 0 to ignore. For use with PS3 only. PCM codecs consume 12,672 bytes per instance and this number will determine how many streams and PCM voices can be played simultaneously. Default = 16

ASIONumChannels

[in/out] Optional. Specify 0 to ignore. Number of channels available on the ASIO device.

ASIOChannelList

[in/out] Optional. Specify 0 to ignore. Pointer to an array of strings (number of entries defined by ASIONumChannels) with ASIO channel names.

ASIOSpeakerList

[in/out] Optional. Specify 0 to ignore. Pointer to a list of speakers that the ASIO channels map to. This can be called after System::init to remap ASIO output.

max3DReverbDSPs

[in/out] Optional. Specify 0 to ignore. The max number of 3d reverb DSP's in the system.

HRTFMinAngle

[in/out] Optional. Specify 0 to ignore. For use with FMOD_INIT_SOFTWARE_HRTF. The angle (0-360) of a 3D sound from the listener's forward vector at which the HRTF function begins to have an effect. Default = 180.0.

HRTFMaxAngle

[in/out] Optional. Specify 0 to ignore. For use with FMOD_INIT_SOFTWARE_HRTF. The angle (0-360) of a 3D sound from the listener's forward vector at which the HRTF function begins to have maximum effect. Default = 360.0.

HRTFFreq

[in/out] Optional. Specify 0 to ignore. For use with FMOD_INIT_SOFTWARE_HRTF. The cutoff frequency of the HRTF's lowpass filter function when at maximum effect. (i.e. at HRTFMaxAngle). Default = 4000.0.

vol0virtualvol

[in/out] Optional. Specify 0 to ignore. For use with FMOD_INIT_VOL0_BECOMES_VIRTUAL. If this flag is used, and the volume is 0.0, then the sound will become virtual. Use this value to raise the threshold to a different point where a sound goes virtual.

eventqueuesize

[in/out] Optional. Specify 0 to ignore. For use with FMOD Event system only. Specifies the number of slots available for simultaneous non blocking loads. Default = 32.

defaultDecodeBufferSize

[in/out] Optional. Specify 0 to ignore. For streams. This determines the default size of the double buffer (in milliseconds) that a stream uses. Default = 400ms

Remarks

maxMPEGcodecs / maxADPCMcodecs / maxXMAcodecs will determine the maximum cpu usage of playing realtime samples. Use this to lower potential excess cpu usage and also control memory usage.

maxPCMcodecs is for use with PS3 only. It will determine the maximum number of PCM voices that can be played at once. This includes streams of any format and all sounds created *without* the FMOD_CREATECOMPRESSEDSAMPLE flag.
Memory will be allocated for codecs 'up front' (during System::init) if these values are specified as non zero. If any are zero, it allocates memory for the codec whenever a file of the type in question is loaded. So if maxMPEGcodecs is 0 for example, it will allocate memory for the mpeg codecs the first time an mp3 is loaded or an mp3 based .FSB file is loaded.

Due to inefficient encoding techniques on certain .wav based ADPCM files, FMOD can can need an extra 29720 bytes per codec. This means for lowest memory consumption. Use FSB as it uses an optimal/small ADPCM block size.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setAdvancedSettings
- System::getAdvancedSettings
- System::init
- FMOD_MODE

FMOD_CDTOC

Structure describing a CD/DVD table of contents?

Structure

```
  typedef struct {
  int  numtracks;
  int min[100];
  int sec[100];
  int  frame[100];
}  FMOD_CDTOC;
```

Members

numtracks

[out] The number of tracks on the CD

min

[out] The start offset of each track in minutes

sec

[out] The start offset of each track in seconds

frame

[out] The start offset of each track in frames

Remarks

Members marked with [in] mean the user sets the value before passing it to the function.
Members marked with [out] mean FMOD sets the value to be used after the function exits.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getTag

FMOD_CODEC_DESCRIPTION

When creating a codec, declare one of these and provide the relevant callbacks and name for FMOD to use when it opens and reads a file.?

Structure

```
  typedef struct {
  const char * name;
  unsigned int version
  int defaultasstream;
   FMOD_TIMEUNIT timeunits;
   FMOD_CODEC_OPENCALLBACK open
   FMOD_CODEC_CLOSECALLBACK close;
   FMOD_CODEC_READCALLBACK read
   FMOD_CODEC_GETLENGTHCALLBACK getlength
   FMOD_CODEC_SETPOSITIONCALLBACK setposition
   FMOD_CODEC_GETPOSITIONCALLBACK getposition
   FMOD_CODEC_SOUNDCREATECALLBACK soundcreate;
   FMOD_CODEC_GETWAVEFORMAT getwaveformat
}  FMOD_CODEC_DESCRIPTION
```

Members

name

[in] Name of the codec.

version

[in] Plugin writer's version number.

defaultasstream

[in] Tells FMOD to open the file as a stream when calling System::createSound, and not a static sample. Should normally be 0 (FALSE), because generally the user wants to decode the file into memory when using System::createSound. Mainly used for formats that decode for a very long time, or could use large amounts of memory when decoded. Usually sequenced formats such as mod/s3m/xm/it/midi fall into this category. It is mainly to stop users that don't know what they're doing from getting FMOD_ERR_MEMORY returned from createSound when they should have in fact called System::createStream or used FMOD_CREATESTREAM in System::createSound.

timeunits

[in] When setposition codec is called, only these time formats will be passed to the codec. Use bitwise OR to accumulate different types.

open

[in] Open callback for the codec for when FMOD tries to open a sound using this codec.

close

[in] Close callback for the codec for when FMOD tries to close a sound using this codec.

read

[in] Read callback for the codec for when FMOD tries to read some data from the file to the destination format (specified in the open callback).

getlength

[in] Callback to return the length of the song in whatever format required when Sound::getLength is called.

setposition

[in] Seek callback for the codec for when FMOD tries to seek within the file with Channel::setPosition.

getposition

[in] Tell callback for the codec for when FMOD tries to get the current position within the with Channel::getPosition.

soundcreate

[in] Sound creation callback for the codec when FMOD finishes creating the sound. (So the codec can set more parameters for the related created sound, ie loop points/mode or 3D attributes etc).

getwaveformat

[in] Callback to tell FMOD about the waveformat of a particular subsound. This is to save memory, rather than saving 1000 FMOD_CODEC_WAVEFORMAT structures in the codec, the codec might have a more optimal way of storing this information.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_CODEC_STATE

FMOD_CODEC_STATE

Codec plugin structure that is passed into each callback.
?
?Set these numsubsounds and waveformat members when called in FMOD_CODEC_OPENCALLBACK to tell fmod what sort of sound to create.
?
?The format, channels and frequency tell FMOD what sort of hardware buffer to create when you initialize your code. So if you wrote an MP3 codec that decoded to stereo 16bit integer PCM, you would specify FMOD_SOUND_FORMAT_PCM16, and channels would be equal to 2.
?

Structure

```
  typedef struct {
  int  numsubsounds;
   FMOD_CODEC_WAVEFORMAT * waveformat;
   void *  plugindata;
   void *  filehandle;
  unsigned int  filesize;
   FMOD_FILE_READCALLBACK   fileread;
   FMOD_FILE_SEEKCALLBACK   fileseek;
   FMOD_CODEC_METADATACALLBACK  metadata;
}  FMOD_CODEC_STATE;
```

Members

numsubsounds

[in] Number of 'subsounds' in this sound. Anything other than 0 makes it a 'container' format (ie CDDA/DLS/FSB etc which contain 1 or more su bsounds). For most normal, single sound codec such as WAV/AIFF/MP3, this should be 0 as they are not a container for subsounds, they are the sound by itself.

waveformat

[in] Pointer to an array of format structures containing information about each sample. Can be 0 or NULL if FMOD_CODEC_GETWAVEFORMAT callback is preferred. The number of entries here must equal the number of subsounds defined in the subsound parameter. If numsubsounds = 0 then there should be 1 instance of this structure.

plugindata

[in] Plugin writer created data the codec author wants to attach to this object.

filehandle

[out] This will return an internal FMOD file handle to use with the callbacks provided.

filesize

[out] This will contain the size of the file in bytes.

fileread

[out] This will return a callable FMOD file function to use from codec.

fileseek

[out] This will return a callable FMOD file function to use from codec.

metadata

[out] This will return a callable FMOD metadata function to use from codec.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

An FMOD file might be from disk, memory or internet, however the file may be opened by the user.

'numsubsounds' should be 0 if the file is a normal single sound stream or sound. Examples of this would be .WAV, .WMA, .MP3, .AIFF.
'numsubsounds' should be 1+ if the file is a container format, and does not contain wav data itself. Examples of these types would be CDDA (multiple CD tracks), FSB (contains multiple sounds), DLS (contain instruments).
The arrays of format, channel, frequency, length and blockalign should point to arrays of information based on how many subsounds are in the format. If the number of subsounds is 0 then it should point to 1 of each attribute, the same as if the number of subsounds was 1. If subsounds was 100 for example, each pointer should point to an array of 100 of each attribute.
When a sound has 1 or more subsounds, you must play the individual sounds specified by first obtaining the subsound with Sound::getSubSound.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_SOUND_FORMAT
- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK
- FMOD_CODEC_METADATACALLBACK
- Sound::getSubSound
- Sound::getNumSubSounds

FMOD_CODEC_WAVEFORMAT

Set these values marked 'in' to tell fmod what sort of sound to create.
?The format, channels and frequency tell FMOD what sort of hardware buffer to create when you initialize your code. So if you wrote an MP3 codec that decoded to stereo 16bit integer PCM, you would specify FMOD_SOUND_FORMAT_PCM16, and channels would be equal to 2.
?Members marked as 'out' are set by fmod. Do not modify these. Simply specify 0 for these values when declaring the structure, FMOD will fill in the values for you after creation with the correct function pointers.
?

Structure

```
  typedef struct {
  char name[256];
   FMOD_SOUND_FORMAT format;
  int channels;
  int frequency;
  unsigned int lengthbytes;
  unsigned int lengthpcm;
  int blockalign;
  int loopstart;
  int loopend;
   FMOD_MODE mode;
  unsigned int channelmask;
}  FMOD_CODEC_WAVEFORMAT;
```

Members

name

[in] Name of sound.

format

[in] Format for (decompressed) codec output, ie FMOD_SOUND_FORMAT_PCM8, FMOD_SOUND_FORMAT_PCM16.

channels

[in] Number of channels used by codec, ie mono = 1, stereo = 2.

frequency

[in] Default frequency in hz of the codec, ie 44100.

lengthbytes

[in] Length in bytes of the source data.

lengthpcm

[in] Length in decompressed, PCM samples of the file, ie length in seconds * frequency. Used for Sound::getLength and for memory allocation of static decompressed sample data.

blockalign

[in] Blockalign in decompressed, PCM samples of the optimal decode chunk size for this format. The codec read callback will be called in multiples of this value.

loopstart

[in] Loopstart in decompressed, PCM samples of file.

loopend

[in] Loopend in decompressed, PCM samples of file.

mode

[in] Mode to determine whether the sound should by default load as looping, non looping, 2d or 3d.

channelmask

[in] Microsoft speaker channel mask, as defined for WAVEFORMATEXTENSIBLE and is found in ksmedia.h. Leave at 0 to play in natural speaker order.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

An FMOD file might be from disk, memory or network, however the file may be opened by the user.

'numsubsounds' should be 0 if the file is a normal single sound stream or sound. Examples of this would be .WAV, .WMA, .MP3, .AIFF.
'numsubsounds' should be 1+ if the file is a container format, and does not contain wav data itself. Examples of these types would be CDDA (multiple CD tracks), FSB (contains multiple sounds), MIDI/MOD/S3M/XM/IT (contain instruments).
The arrays of format, channel, frequency, length and blockalign should point to arrays of information based on how many subsounds are in the format. If the number of subsounds is 0 then it should point to 1 of each attribute, the same as if the number of subsounds was 1. If subsounds was 100 for example, each pointer should point to an array of 100 of each attribute.
When a sound has 1 or more subsounds, you must play the individual sounds specified by first obtaining the subsound with Sound::getSubSound.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_SOUND_FORMAT
- FMOD_FILE_READCALLBACK
- FMOD_FILE_SEEKCALLBACK
- FMOD_CODEC_METADATACALLBACK
- Sound::getSubSound
- Sound::getNumSubSounds

FMOD_CREATESOUNDEXINFO

Use this structure with System::createSound when more control is needed over loading.
?The possible reasons to use this with System::createSound are:
?

- Loading a file from memory.?
- Loading a file from within another larger (possibly wad/pak) file, by giving the loader an offset and length.?
- To create a user created / non file based sound.?
- To specify a starting subsound to seek to within a multi-sample sounds (ie FSB/DLS/SF2) when created as a stream.?
- To specify which subsounds to load for multi-sample sounds (ie FSB/DLS/SF2) so that memory is saved and only a subset is actually loaded/read from disk.?
- To specify 'piggyback' read and seek callbacks for capture of sound data as fmod reads and decodes it. Useful for ripping decoded PCM data from sounds as they are loaded / played.?
- To specify a MIDI DLS/SF2 sample set file to load when opening a MIDI file.?See below on what members to fill for each of the above types of sound you want to create.?

Structure

```
typedef struct {
  int cbsize;
  unsigned int length
  unsigned int fileoffset
  int numchannels;
  int defaultfrequency;
   FMOD_SOUND_FORMAT format
  unsigned int decodebuffersize;
  int initialsubsound
  int numsubsounds;
  int * inclusionlist
  int inclusionlistnum;
   FMOD_SOUND_PCMREADCALLBACK pcmreadcallback;
   FMOD_SOUND_PCMSETPOSCALLBACK pcmsetposcallback;
   FMOD_SOUND_NONBLOCKCALLBACK nonblockcallback;
  const char * dlsname;
  const char * encryptionkey;
  int maxpolyphony;
   void * userdata;
   FMOD_SOUND_TYPE suggestedsoundtype;
   FMOD_FILE_OPENCALLBACK useropen
   FMOD_FILE_CLOSECALLBACK userclose;
   FMOD_FILE_READCALLBACK userread
   FMOD_FILE_SEEKCALLBACK userseek;
   FMOD_SPEAKERMAPTYPE speakermap
   FMOD_SOUNDGROUP * initialsoundgroup
  unsigned int initialseekposition
   FMOD_TIMEUNIT initialseekpostype;
}  FMOD_CREATESOUNDEXINFO;
```

Members

cbsize

[in] Size of this structure. This is used so the structure can be expanded in the future and still work on older versions of FMOD Ex.

length

[in] Optional. Specify 0 to ignore. Size in bytes of file to load, or sound to create (in this case only if FMOD_OPENUSER is used). Required if loading from memory. If 0 is specified, then it will use the size of the file (unless loading from memory then an error will be returned).

fileoffset

[in] Optional. Specify 0 to ignore. Offset from start of the file to start loading from. This is useful for loading files from inside big data files.

numchannels

[in] Optional. Specify 0 to ignore. Number of channels in a sound mandatory if FMOD_OPENUSER or FMOD_OPENRAW is used.

defaultfrequency

[in] Optional. Specify 0 to ignore. Default frequency of sound in a sound mandatory if FMOD_OPENUSER or FMOD_OPENRAW is used. Other formats use the frequency determined by the file format.

format

[in] Optional. Specify 0 or FMOD_SOUND_FORMAT_NONE to ignore. Format of the sound mandatory if FMOD_OPENUSER or FMOD_OPENRAW is used. Other formats use the format determined by the file format.

decodebuffersize

[in] Optional. Specify 0 to ignore. For streams. This determines the size of the double buffer (in PCM samples) that a stream uses. Use this for user created streams if you want to determine the size of the callback buffer passed to you. Specify 0 to use FMOD's default size which is currently equivalent to 400ms of the sound format created/loaded.

initialsubsound

[in] Optional. Specify 0 to ignore. In a multi-sample file format such as .FSB/.DLS/.SF2, specify the initial subsound to seek to, only if FMOD_CREATESTREAM is used.

numsubsounds

[in] Optional. Specify 0 to ignore or have no subsounds. In a user created multi-sample sound, specify the number of subsounds within the sound that are accessable with Sound::getSubSound.

inclusionlist

[in] Optional. Specify 0 to ignore. In a multi-sample format such as .FSB/.DLS/.SF2 it may be desirable to specify only a subset of sounds to be loaded out of the whole file. This is an array of subsound indices to load into memory when created.

inclusionlistnum

[in] Optional. Specify 0 to ignore. This is the number of integers contained within the inclusionlist array.

pcmreadcallback

[in] Optional. Specify 0 to ignore. Callback to 'piggyback' on FMOD's read functions and accept or even write PCM data while FMOD is opening the sound. Used for user sounds created with FMOD_OPENUSER or for capturing decoded data as FMOD reads it.

pcmsetposcallback

[in] Optional. Specify 0 to ignore. Callback for when the user calls a seeking function such as Channel::setTime or Channel::setPosition within a multi-sample sound, and for when it is opened.

nonblockcallback

[in] Optional. Specify 0 to ignore. Callback for successful completion, or error while loading a sound that used the FMOD_NONBLOCKING flag.

dlsname

[in] Optional. Specify 0 to ignore. Filename for a DLS or SF2 sample set when loading a MIDI file. If not specified, on Windows it will attempt to open /windows/system32/drivers/gm.dls or /windows/system32/drivers/etc/gm.dls, on Mac it will attempt to load /System/Library/Components/CoreAudio.component/Contents/Resources/gs_instruments.dls, otherwise the MIDI will fail to open. Current DLS support is for level 1 of the specification.

encryptionkey

[in] Optional. Specify 0 to ignore. Key for encrypted FSB file. Without this key an encrypted FSB file will not load.

maxpolyphony

[in] Optional. Specify 0 to ignore. For sequenced formats with dynamic channel allocation such as .MID and .IT, this specifies the maximum voice count allowed while playing. .IT defaults to 64. .MID defaults to 32.

userdata

[in] Optional. Specify 0 to ignore. This is user data to be attached to the sound during creation. Access via Sound::getUserData.

suggestedsoundtype

[in] Optional. Specify 0 or FMOD_SOUND_TYPE_UNKNOWN to ignore. Instead of scanning all codec types, use this to speed up loading by making it jump straight to this codec.

useropen

[in] Optional. Specify 0 to ignore. Callback for opening this file.

userclose

[in] Optional. Specify 0 to ignore. Callback for closing this file.

userread

[in] Optional. Specify 0 to ignore. Callback for reading from this file.

userseek

[in] Optional. Specify 0 to ignore. Callback for seeking within this file.

speakermap

[in] Optional. Specify 0 to ignore. Use this to differ the way fmod maps multichannel sounds to speakers. See FMOD_SPEAKERMAPTYPE for more.

initialsoundgroup

[in] Optional. Specify 0 to ignore. Specify a sound group if required, to put sound in as it is created.

initialseekposition

[in] Optional. Specify 0 to ignore. For streams. Specify an initial position to seek the stream to.

initialseekpostype

[in] Optional. Specify 0 to ignore. For streams. Specify the time unit for the position set in initialseekposition.

Remarks

This structure is optional! Specify 0 or NULL in System::createSound if you don't need it!

Members marked with [in] mean the user sets the value before passing it to the function.
Members marked with [out] mean FMOD sets the value to be used after the function exits.

Loading a file from memory.

- Create the sound using the FMOD_OPENMEMORY flag.
- Mandatory. Specify 'length' for the size of the memory block in bytes.
- Other flags are optional.

Loading a file from within another larger (possibly wad/pak) file, by giving the loader an offset and length.

- Mandatory. Specify 'fileoffset' and 'length'.
- Other flags are optional.

To create a user created / non file based sound.

- Create the sound using the FMOD_OPENUSER flag.
- Mandatory. Specify 'defaultfrequency, 'numchannels' and 'format'.
- Other flags are optional.

To specify a starting subsound to seek to and flush with, within a multi-sample stream (ie FSB/DLS/SF2).

- Mandatory. Specify 'initialsubsound'.

To specify which subsounds to load for multi-sample sounds (ie FSB/DLS/SF2) so that memory is saved and only a subset is actually loaded/read from disk.

- Mandatory. Specify 'inclusionlist' and 'inclusionlistnum'.

To specify 'piggyback' read and seek callbacks for capture of sound data as fmod reads and decodes it. Useful for ripping decoded PCM data from sounds as they are loaded / played.

- Mandatory. Specify 'pcmreadcallback' and 'pcmseekcallback'.

To specify a MIDI DLS/SF2 sample set file to load when opening a MIDI file.

- Mandatory. Specify 'dlsname'.

Setting the 'decodebuffersize' is for cpu intensive codecs that may be causing stuttering, not file intensive codecs (ie those from CD or netstreams) which are normally altered with System::setStreamBufferSize. As an example of cpu intensive codecs, an mp3 file will take more cpu to decode than a PCM wav file.
If you have a stuttering effect, then it is using more cpu than the decode buffer playback rate can keep up with. Increasing the decode buffersize will most likely solve this problem.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound
- System::setStreamBufferSize
- FMOD_MODE
- FMOD_SOUND_FORMAT
- FMOD_SOUND_TYPE
- FMOD_SPEAKERMAPTYPE

FMOD_DSP_DESCRIPTION

When creating a DSP unit, declare one of these and provide the relevant callbacks and name for FMOD to use when it creates and uses a DSP unit of this type.?

Structure

```
  typedef struct {
  char name[32];
  unsigned int version;
  int channels;
   FMOD_DSP_CREATECALLBACK create;
   FMOD_DSP_RELEASECALLBACK release;
   FMOD_DSP_RESETCALLBACK reset;
   FMOD_DSP_READCALLBACK read;
   FMOD_DSP_SETPOSITIONCALLBACK setposition;
  int numparameters;
   FMOD_DSP_PARAMETERDESC * paramdesc;
   FMOD_DSP_SETPARAMCALLBACK setparameter;
   FMOD_DSP_GETPARAMCALLBACK getparameter;
   FMOD_DSP_DIALOGCALLBACK config;
  int configwidth;
  int configheight;
   void * userdata;
} FMOD_DSP_DESCRIPTION;
```

Members

name

[in] Name of the unit to be displayed in the network.

version

[in] Plugin writer's version number.

channels

[in] Number of channels. Use 0 to process whatever number of channels is currently in the network. >0 would be mostly used if the unit is a unit that only generates sound.

create

[in] Create callback. This is called when DSP unit is created. Can be null.

release

[in] Release callback. This is called just before the unit is freed so the user can do any cleanup needed for the unit. Can be null.

reset

[in] Reset callback. This is called by the user to reset any history buffers that may need resetting for a filter, when it is to be used or re-used for the first time to its initial clean state. Use to avoid clicks or artifacts.

read

[in] Read callback. Processing is done here. Can be null.

setposition

[in] Set position callback. This is called if the unit wants to update its position info but not process data, or reset a cursor position internally if it is reading data from a certain source. Can be null.

numparameters

[in] Number of parameters used in this filter. The user finds this with DSP::getNumParameters

paramdesc

[in] Variable number of parameter structures.

setparameter

[in] This is called when the user calls DSP::setParameter. Can be null.

getparameter

[in] This is called when the user calls DSP::getParameter. Can be null.

config

[in] This is called when the user calls DSP::showConfigDialog. Can be used to display a dialog to configure the filter. Can be null.

configwidth

[in] Width of config dialog graphic if there is one. 0 otherwise.

configheight

[in] Height of config dialog graphic if there is one. 0 otherwise.

userdata

[in] Optional. Specify 0 to ignore. This is user data to be attached to the DSP unit during creation. Access via DSP::getUserData.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

There are 2 different ways to change a parameter in this architecture.
One is to use DSP::setParameter / DSP::getParameter. This is platform independant and is dynamic, so new unknown

plugins can have their parameters enumerated and used.
The other is to use DSP::showConfigDialog. This is platform specific and requires a GUI, and will display a dialog box to configure the plugin.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::createDSP
- FMOD_DSP_STATE

FMOD_DSP_PARAMETERDESC

Structure to define a parameter for a DSP unit.?

Structure

```
typedef struct {
  float min;
  float max;
  float defaultval;
 char name[16];
 char label[16];
 const char * description;
} FMOD_DSP_PARAMETERDESC;
```

Members

min

[in] Minimum value of the parameter (ie 100.0).

max

[in] Maximum value of the parameter (ie 22050.0).

defaultval

[in] Default value of parameter.

name

[in] Name of the parameter to be displayed (ie "Cutoff frequency").

label

[in] Short string to be put next to value to denote the unit type (ie "hz").

description

[in] Description of the parameter to be displayed as a help item / tooltip for this parameter.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

The step parameter tells the gui or application that the parameter has a certain granularity.
For example in the example of cutoff frequency with a range from 100.0 to 22050.0 you might only want the selection to be in 10hz increments. For this you would simply use 10.0 as the step value.
For a boolean, you can use min = 0.0, max = 1.0, step = 1.0. This way the only possible values are 0.0 and 1.0.
Some applications may detect min = 0.0, max = 1.0, step = 1.0 and replace a graphical slider bar with a checkbox instead.
A step value of 1.0 would simulate integer values only.
A step value of 0.0 would mean the full floating point range is accessable.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::createDSP
- DSP::setParameter

FMOD_DSP_STATE

DSP plugin structure that is passed into each callback.?

Structure

```
typedef struct {
  FMOD_DSP * instance;
  void * plugindata;
} FMOD_DSP_STATE;
```

Members

instance

[out] Handle to the DSP hand the user created. Not to be modified. C++ users cast to FMOD::DSP to use.

plugindata

[in] Plugin writer created data the output author wants to attach to this object.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_DSP_DESCRIPTION

FMOD_GUID

Structure describing a globally unique identifier.?

Structure

```
  typedef struct {
  unsigned int  Data1;
  unsigned short  Data2;
  unsigned short  Data3;
  unsigned char  Data4[8];
}  FMOD_GUID;
```

Members

Data1

Specifies the first 8 hexadecimal digits of the GUID

Data2

Specifies the first group of 4 hexadecimal digits.

Data3

Specifies the second group of 4 hexadecimal digits.

Data4

Array of 8 bytes. The first 2 bytes contain the third group of 4 hexadecimal digits. The remaining 6 bytes contain the final 12 hexadecimal digits.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDriverInfo

FMOD_OUTPUT_DESCRIPTION

When creating an output, declare one of these and provide the relevant callbacks and name for FMOD to use when it opens and reads a file of this type.?

Structure

```
  typedef struct {
  const char *  name;
  unsigned int  version;
  int  polling;
   FMOD_OUTPUT_GETNUMDRIVERSCALLBACK  getnumdrivers;
   FMOD_OUTPUT_GETDRIVERNAMECALLBACK  getdrivername;
   FMOD_OUTPUT_GETDRIVERCAPSCALLBACK  getdrivercaps;
   FMOD_OUTPUT_INITCALLBACK  init;
   FMOD_OUTPUT_CLOSECALLBACK  close;
   FMOD_OUTPUT_UPDATECALLBACK  update;
   FMOD_OUTPUT_GETHANDLECALLBACK  gethandle;
   FMOD_OUTPUT_GETPOSITIONCALLBACK  getposition;
   FMOD_OUTPUT_LOCKCALLBACK  lock;
   FMOD_OUTPUT_UNLOCKCALLBACK  unlock;
}  FMOD_OUTPUT_DESCRIPTION;
```

Members

name

[in] Name of the output.

version

[in] Plugin writer's version number.

polling

[in] If TRUE (non zero), this tells FMOD to start a thread and call getposition / lock / unlock for feeding data. If 0, the output is probably callback based, so all the plugin needs to do is call readfrommixer to the appropriate pointer.

getnumdrivers

[in] For sound device enumeration. This callback is to give System::getNumDrivers somthing to return.

getdrivername

[in] For sound device enumeration. This callback is to give System::getDriverName somthing to return.

getdrivercaps

[in] For sound device enumeration. This callback is to give System::getDriverCaps somthing to return.

init

[in] Initialization function for the output device. This is called from System::init.

close

[in] Cleanup / close down function for the output device. This is called from System::close.

update

[in] Update function that is called once a frame by the user. This is called from System::update.

gethandle

[in] This is called from System::getOutputHandle. This is just to return a pointer to the internal system device object that the system may be using.

getposition

[in] This is called from the FMOD software mixer thread if 'polling' = true. This returns a position value in samples so that FMOD knows where and when to fill its buffer.

lock

[in] This is called from the FMOD software mixer thread if 'polling' = true. This function provides a pointer to data that FMOD can write to when software mixing.

unlock

[in] This is called from the FMOD software mixer thread if 'polling' = true. This optional function accepts the data that has been mixed and copies it or does whatever it needs to before sending it to the hardware.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_OUTPUT_STATE

FMOD_OUTPUT_STATE

Output plugin structure that is passed into each callback.?

Structure

```
  typedef struct {
    void *  plugindata;
    FMOD_OUTPUT_READFROMMIXER  readfrommixer;
} FMOD_OUTPUT_STATE;
```

Members

plugindata

[in] Plugin writer created data the output author wants to attach to this object.

readfrommixer

[out] Function to update mixer and write the result to the provided pointer. Used from callback based output only. Polling based output uses lock/unlock/getposition.

Remarks

Members marked with [in] mean the variable can be written to. The user can set the value.
Members marked with [out] mean the variable is modified by FMOD and is for reading purposes only. Do not change this value.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_OUTPUT_DESCRIPTION

FMOD_REVERB_CHANNELPROPERTIES

Structure defining the properties for a reverb source, related to a FMOD channel.
?
?For more indepth descriptions of the reverb properties under win32, please see the EAX3?documentation at http://developer.creative.com/ under the 'downloads' section.
?If they do not have the EAX3 documentation, then most information can be attained from?the EAX2 documentation, as EAX3 only adds some more parameters and functionality on top of?EAX2.
?
?Note the default reverb properties are the same as the FMOD_PRESET_GENERIC preset.
?Note that integer values that typically range from -10,000 to 1000 are represented in?decibels, and are of a logarithmic scale, not linear, wheras float values are typically linear.
?PORTABILITY: Each member has the platform it supports in braces ie (win32/Xbox).
?Some reverb parameters are only supported in win32 and some only on Xbox. If all parameters are set then?the reverb should product a similar effect on either platform.
?
?The numerical values listed below are the maximum, minimum and default values for each variable respectively.
?

Structure

```
  typedef struct {
  int  Direct;
  int  DirectHF;
  int  Room;
  int  RoomHF;
  int Obstruction;
   float ObstructionLFRatio;
  int Occlusion;
   float OcclusionLFRatio;
   float OcclusionRoomRatio;
   float OcclusionDirectRatio;
  int Exclusion;
   float ExclusionLFRatio;
  int OutsideVolumeHF;
   float  DopplerFactor;
   float  RolloffFactor;
   float  RoomRolloffFactor;
   float AirAbsorptionFactor;
  unsigned int  Flags;
}  FMOD_REVERB_CHANNELPROPERTIES;
```

Members

Direct

[in/out] -10000, 1000, 0, direct path level (at low and mid frequencies) (SUPPORTED:EAX/I3DL2/Xbox1/SFX)

DirectHF

[in/out] -10000, 0, 0, relative direct path level at high frequencies (SUPPORTED:EAX/I3DL2/Xbox1)

Room

[in/out] -10000, 1000, 0, room effect level (at low and mid frequencies) (SUPPORTED:EAX/I3DL2/Xbox1/GC/SFX)

RoomHF

[in/out] -10000, 0, 0, relative room effect level at high frequencies (SUPPORTED:EAX/I3DL2/Xbox1)

Obstruction

[in/out] -10000, 0, 0, main obstruction control (attenuation at high frequencies) (SUPPORTED:EAX/I3DL2/Xbox1)

ObstructionLFRatio

[in/out] 0.0, 1.0, 0.0, obstruction low-frequency level re. main control (SUPPORTED:EAX/I3DL2/Xbox1)

Occlusion

[in/out] -10000, 0, 0, main occlusion control (attenuation at high frequencies) (SUPPORTED:EAX/I3DL2/Xbox1)

OcclusionLFRatio

[in/out] 0.0, 1.0, 0.25, occlusion low-frequency level re. main control (SUPPORTED:EAX/I3DL2/Xbox1)

OcclusionRoomRatio

[in/out] 0.0, 10.0, 1.5, relative occlusion control for room effect (SUPPORTED:EAX)

OcclusionDirectRatio

[in/out] 0.0, 10.0, 1.0, relative occlusion control for direct path (SUPPORTED:EAX)

Exclusion

[in/out] -10000, 0, 0, main exlusion control (attenuation at high frequencies) (SUPPORTED:EAX)

ExclusionLFRatio

[in/out] 0.0, 1.0, 1.0, exclusion low-frequency level re. main control (SUPPORTED:EAX)

OutsideVolumeHF

[in/out] -10000, 0, 0, outside sound cone level at high frequencies (SUPPORTED:EAX)

DopplerFactor

[in/out] 0.0, 10.0, 0.0, like DS3D flDopplerFactor but per source (SUPPORTED:EAX)

RolloffFactor

[in/out] 0.0, 10.0, 0.0, like DS3D flRolloffFactor but per source (SUPPORTED:EAX)

RoomRolloffFactor

[in/out] 0.0, 10.0, 0.0, like DS3D flRolloffFactor but for room effect (SUPPORTED:EAX/I3DL2/Xbox1)

AirAbsorptionFactor

[in/out] 0.0, 10.0, 1.0, multiplies AirAbsorptionHF member of FMOD_REVERB_PROPERTIES (SUPPORTED:EAX)

Flags

[in/out] FMOD_REVERB_CHANNELFLAGS - modifies the behavior of properties (SUPPORTED:EAX)

Remarks

SUPPORTED next to each parameter means the platform the parameter can be set on. Some platforms support all parameters and some don't.
EAX means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support EAX 1 to 4.
EAX4 means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support EAX 4.
I3DL2 means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support I3DL2 non EAX native reverb.
GC means Nintendo Gamecube hardware reverb (must use FMOD_HARDWARE).
WII means Nintendo Wii hardware reverb (must use FMOD_HARDWARE).
Xbox1 means the original Xbox hardware reverb (must use FMOD_HARDWARE).
PS2 means Playstation 2 hardware reverb (must use FMOD_HARDWARE).
SFX means FMOD SFX software reverb. This works on any platform that uses FMOD_SOFTWARE for loading sounds.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setReverbProperties
- Channel::getReverbProperties
- FMOD_REVERB_CHANNELFLAGS

FMOD_REVERB_PROPERTIES

Structure defining a reverb environment.
?
?For more indepth descriptions of the reverb properties under win32, please see the EAX2 and EAX3?documentation at http://developer.creative.com/ under the 'downloads' section.
?If they do not have the EAX3 documentation, then most information can be attained from?the EAX2 documentation, as EAX3 only adds some more parameters and functionality on top of?EAX2.?

Structure

```
  typedef struct{
  int Instance;
  int Environment
   float EnvSize;
   float EnvDiffusion
  int Room;
  int RoomHF
  int RoomLF
   float DecayTime;
   float DecayHFRatio;
   float DecayLFRatio;
  int Reflections;
   float ReflectionsDelay;
   float ReflectionsPan[3];
  int Reverb
   float ReverbDelay;
   float ReverbPan[3];
   float EchoTime;
   float EchoDepth
   float ModulationTime;
   float ModulationDepth
   float AirAbsorptionHF
   float HFReference;
   float LFReference;
   float RoomRolloffFactor
   float Diffusion
   float Density;
  unsigned int Flags;
} FMOD_REVERB_PROPERTIES;
```

Members

Instance

[in] 0 , 3 , 0 , Environment Instance. Simultaneous HW reverbs are possible on some platforms. (SUPPORTED:EAX4(3 instances)/GC and Wii (2 instances))

Environment

[in/out] -1 , 25 , -1 , sets all listener properties. -1 = OFF. (SUPPORTED:EAX/PS2)

EnvSize

[in/out] 1.0 , 100.0 , 7.5 , environment size in meters (SUPPORTED:EAX)

EnvDiffusion

[in/out] 0.0 , 1.0 , 1.0 , environment diffusion (SUPPORTED:EAX/Xbox1/GC)

Room

[in/out] -10000, 0 , -1000 , room effect level (at mid frequencies) (SUPPORTED:EAX/Xbox1/GC/I3DL2/SFX)

RoomHF

[in/out] -10000, 0 , -100 , relative room effect level at high frequencies (SUPPORTED:EAX/Xbox1/I3DL2/SFX)

RoomLF

[in/out] -10000, 0 , 0 , relative room effect level at low frequencies (SUPPORTED:EAX/SFX)

DecayTime

[in/out] 0.1 , 20.0 , 1.49 , reverberation decay time at mid frequencies (SUPPORTED:EAX/Xbox1/GC/I3DL2/SFX)

DecayHFRatio

[in/out] 0.1 , 2.0 , 0.83 , high-frequency to mid-frequency decay time ratio (SUPPORTED:EAX/Xbox1/I3DL2/SFX)

DecayLFRatio

[in/out] 0.1 , 2.0 , 1.0 , low-frequency to mid-frequency decay time ratio (SUPPORTED:EAX)

Reflections

[in/out] -10000, 1000 , -2602 , early reflections level relative to room effect (SUPPORTED:EAX/Xbox1/GC/I3DL2/SFX)

ReflectionsDelay

[in/out] 0.0 , 0.3 , 0.007 , initial reflection delay time (SUPPORTED:EAX/Xbox1/I3DL2/SFX)

ReflectionsPan

[in/out] , , [0,0,0], early reflections panning vector (SUPPORTED:EAX)

Reverb

[in/out] -10000, 2000 , 200 , late reverberation level relative to room effect (SUPPORTED:EAX/Xbox1/I3DL2/SFX)

ReverbDelay

[in/out] 0.0 , 0.1 , 0.011 , late reverberation delay time relative to initial reflection (SUPPORTED:EAX/Xbox1/GC/I3DL2/SFX)

ReverbPan

[in/out] , , [0,0,0], late reverberation panning vector (SUPPORTED:EAX)

EchoTime

[in/out] .075 , 0.25 , 0.25 , echo time
(SUPPORTED:EAX/PS2(FMOD_PRESET_PS2_ECHO/FMOD_PRESET_PS2_DELAY only)

EchoDepth

[in/out] 0.0 , 1.0 , 0.0 , echo depth (SUPPORTED:EAX/PS2(FMOD_PRESET_PS2_ECHO only)

ModulationTime

[in/out] 0.04 , 4.0 , 0.25 , modulation time (SUPPORTED:EAX)

ModulationDepth

[in/out] 0.0 , 1.0 , 0.0 , modulation depth (SUPPORTED:EAX/GC)

AirAbsorptionHF

[in/out] -100 , 0.0 , -5.0 , change in level per meter at high frequencies (SUPPORTED:EAX)

HFReference

[in/out] 1000.0, 20000 , 5000.0 , reference high frequency (hz) (SUPPORTED:EAX/Xbox1/I3DL2/SFX)

LFReference

[in/out] 20.0 , 1000.0, 250.0 , reference low frequency (hz) (SUPPORTED:EAX/SFX)

RoomRolloffFactor

[in/out] 0.0 , 10.0 , 0.0 , like rolloffscale in System::set3DSettings but for reverb room size effect
(SUPPORTED:EAX/Xbox1/I3DL2/SFX)

Diffusion

[in/out] 0.0 , 100.0 , 100.0 , Value that controls the echo density in the late reverberation decay.
(SUPPORTED:I3DL2/Xbox1/SFX)

Density

[in/out] 0.0 , 100.0 , 100.0 , Value that controls the modal density in the late reverberation decay
(SUPPORTED:I3DL2/Xbox1/SFX)

Flags

[in/out] FMOD_REVERB_FLAGS - modifies the behavior of above properties
(SUPPORTED:EAX/PS2/GC/WII)

Remarks

Note the default reverb properties are the same as the FMOD_PRESET_GENERIC preset.
Note that integer values that typically range from -10,000 to 1000 are represented in decibels, and are of a logarithmic scale, not linear, wheras float values are always linear.

The numerical values listed below are the maximum, minimum and default values for each variable respectively.

SUPPORTED next to each parameter means the platform the parameter can be set on. Some platforms support all parameters and some don't.
EAX means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support EAX 1 to 4.
EAX4 means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support EAX 4.
I3DL2 means hardware reverb on FMOD_OUTPUTTYPE_DSOUND on windows only (must use FMOD_HARDWARE), on soundcards that support I3DL2 non EAX native reverb.
GC means Nintendo Gamecube hardware reverb (must use FMOD_HARDWARE).
WII means Nintendo Wii hardware reverb (must use FMOD_HARDWARE).
Xbox1 means the original Xbox hardware reverb (must use FMOD_HARDWARE).
PS2 means Playstation 2 hardware reverb (must use FMOD_HARDWARE).
SFX means FMOD SFX software reverb. This works on any platform that uses FMOD_SOFTWARE for loading sounds.

Members marked with [in] mean the user sets the value before passing it to the function.
Members marked with [out] mean FMOD sets the value to be used after the function exits.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setReverbProperties
- System::getReverbProperties
- FMOD_REVERB_PRESETS
- FMOD_REVERB_FLAGS

FMOD_TAG

Structure describing a piece of tag data.?

Structure

```
  typedef struct {
    FMOD_TAGTYPE      type;
    FMOD_TAGDATATYPE  datatype;
  char *  name;
    void *  data;
  unsigned int  datalen;
    FMOD_BOOL updated;
}   FMOD_TAG;
```

Members

type

[out] The type of this tag.

datatype

[out] The type of data that this tag contains

name

[out] The name of this tag i.e. "TITLE", "ARTIST" etc.

data

[out] Pointer to the tag data - its format is determined by the datatype member

datalen

[out] Length of the data contained in this tag

updated

[out] True if this tag has been updated since last being accessed with Sound::getTag

Remarks

Members marked with [in] mean the user sets the value before passing it to the function.
Members marked with [out] mean FMOD sets the value to be used after the function exits.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getTag
- FMOD_TAGTYPE
- FMOD_TAGDATATYPE

FMOD_VECTOR

Structure describing a point in 3D space.?

Structure

```
typedef struct {
  float  x;
  float  y;
  float  z;
} FMOD_VECTOR;
```

Members

x

X co-ordinate in 3D space.

y

Y co-ordinate in 3D space.

z

Z co-ordinate in 3D space.

Remarks

FMOD uses a left handed co-ordinate system by default.
To use a right handed co-ordinate system specify FMOD_INIT_3D_RIGHTHANDED from FMOD_INITFLAGS in System::init.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::set3DListenerAttributes
- System::get3DListenerAttributes
- Channel::set3DAttributes
- Channel::get3DAttributes
- Channel::set3DCustomRolloff
- Channel::get3DCustomRolloff
- Sound::set3DCustomRolloff

- Sound::get3DCustomRolloff
- Geometry::addPolygon
- Geometry::setPolygonVertex
- Geometry::getPolygonVertex
- Geometry::setRotation
- Geometry::getRotation
- Geometry::setPosition
- Geometry::getPosition
- Geometry::setScale
- Geometry::getScale
- FMOD_INITFLAGS

Defines

FMOD_CAPS

Bit fields to use with System::getDriverCaps to determine the capabilities of a card / output device.?It is important to check FMOD_CAPS_HARDWARE_EMULATED on windows machines, to then adjust System::setDSPBufferSize to (1024, 10) to compensate for the higher latency.?

Definition

```
 #define FMOD_CAPS_NONE   0x00000000
#define FMOD_CAPS_HARDWARE   0x00000001
#define FMOD_CAPS_HARDWARE_EMULATED  0x00000002
#define FMOD_CAPS_OUTPUT_MULTICHANNEL  0x00000004
#define FMOD_CAPS_OUTPUT_FORMAT_PCM8  0x00000008
#define FMOD_CAPS_OUTPUT_FORMAT_PCM16  0x00000010
#define FMOD_CAPS_OUTPUT_FORMAT_PCM24  0x00000020
#define FMOD_CAPS_OUTPUT_FORMAT_PCM32  0x00000040
#define FMOD_CAPS_OUTPUT_FORMAT_PCMFLOAT  0x00000080
#define FMOD_CAPS_REVERB_EAX2  0x00000100
#define FMOD_CAPS_REVERB_EAX3  0x00000200
#define FMOD_CAPS_REVERB_EAX4  0x00000400
#define FMOD_CAPS_REVERB_EAX5  0x00000800
#define FMOD_CAPS_REVERB_I3DL2  0x00001000
#define FMOD_CAPS_REVERB_LIMITED  0x00002000
```

Values

FMOD_CAPS_NONE

Device has no special capabilities.

FMOD_CAPS_HARDWARE

Device supports hardware mixing.

FMOD_CAPS_HARDWARE_EMULATED

User has device set to 'Hardware acceleration = off' in control panel, and now extra 200ms latency is incurred.

FMOD_CAPS_OUTPUT_MULTICHANNEL

Device can do multichannel output, ie greater than 2 channels.

FMOD_CAPS_OUTPUT_FORMAT_PCM8

Device can output to 8bit integer PCM.

FMOD_CAPS_OUTPUT_FORMAT_PCM16

Device can output to 16bit integer PCM.

FMOD_CAPS_OUTPUT_FORMAT_PCM24

Device can output to 24bit integer PCM.

FMOD_CAPS_OUTPUT_FORMAT_PCM32

Device can output to 32bit integer PCM.

FMOD_CAPS_OUTPUT_FORMAT_PCMFLOAT

Device can output to 32bit floating point PCM.

FMOD_CAPS_REVERB_EAX2

Device supports EAX2 reverb.

FMOD_CAPS_REVERB_EAX3

Device supports EAX3 reverb.

FMOD_CAPS_REVERB_EAX4

Device supports EAX4 reverb

FMOD_CAPS_REVERB_EAX5

Device supports EAX5 reverb

FMOD_CAPS_REVERB_I3DL2

Device supports I3DL2 reverb.

FMOD_CAPS_REVERB_LIMITED

Device supports some form of limited hardware reverb, maybe parameterless and only selectable by environment.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getDriverCaps
- System::setDSPBufferSize

FMOD_DEBUGLEVEL

Bit fields to use with FMOD::Debug_SetLevel / FMOD::Debug_GetLevel to control the level of tty debug output with logging versions of FMOD (fmodL).?

Definition

```
 #define FMOD_DEBUG_LEVEL_NONE   0x00000000
#define FMOD_DEBUG_LEVEL_LOG   0x00000001
#define FMOD_DEBUG_LEVEL_ERROR  0x00000002
#define FMOD_DEBUG_LEVEL_WARNING   0x00000004
#define FMOD_DEBUG_LEVEL_HINT  0x00000008
#define FMOD_DEBUG_LEVEL_ALL  0x000000FF
#define FMOD_DEBUG_TYPE_MEMORY   0x00000100
#define FMOD_DEBUG_TYPE_THREAD  0x00000200
#define FMOD_DEBUG_TYPE_FILE   0x00000400
#define FMOD_DEBUG_TYPE_NET  0x00000800
#define FMOD_DEBUG_TYPE_EVENT  0x00001000
#define FMOD_DEBUG_TYPE_ALL  0x0000FFFF
#define FMOD_DEBUG_DISPLAY_TIMESTAMPS   0x01000000
#define FMOD_DEBUG_DISPLAY_LINENUMBERS   0x02000000
#define FMOD_DEBUG_DISPLAY_COMPRESS   0x04000000
#define FMOD_DEBUG_DISPLAY_ALL  0x0F000000
#define FMOD_DEBUG_ALL  0xFFFFFFFF
```

Values

FMOD_DEBUG_LEVEL_NONE

FMOD_DEBUG_LEVEL_LOG

FMOD_DEBUG_LEVEL_ERROR

FMOD_DEBUG_LEVEL_WARNING

FMOD_DEBUG_LEVEL_HINT

FMOD_DEBUG_LEVEL_ALL

FMOD_DEBUG_TYPE_MEMORY

FMOD_DEBUG_TYPE_THREAD

FMOD_DEBUG_TYPE_FILE

FMOD_DEBUG_TYPE_NET

FMOD_DEBUG_TYPE_EVENT

FMOD_DEBUG_TYPE_ALL

FMOD_DEBUG_DISPLAY_TIMESTAMPS

FMOD_DEBUG_DISPLAY_LINENUMBERS

FMOD_DEBUG_DISPLAY_COMPRESS

FMOD_DEBUG_DISPLAY_ALL

FMOD_DEBUG_ALL

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Debug_SetLevel
- Debug_GetLevel

FMOD_INITFLAGS

Initialization flags. Use them with System::init in the flags parameter to change various behavior.?Use System::setAdvancedSettings to adjust settings for some of the features that are enabled by these flags.?

Definition

```
   #define FMOD_INIT_NORMAL  0x00000000
  #define FMOD_INIT_STREAM_FROM_UPDATE   0x00000001
  #define FMOD_INIT_3D_RIGHTHANDED  0x00000002
  #define FMOD_INIT_SOFTWARE_DISABLE   0x00000004
  #define FMOD_INIT_SOFTWARE_OCCLUSION  0x00000008
  #define FMOD_INIT_SOFTWARE_HRTF  0x00000010
  #define FMOD_INIT_ENABLE_PROFILE  0x00000020
  #define FMOD_INIT_VOL0_BECOMES_VIRTUAL  0x00000080
  #define FMOD_INIT_WASAPI_EXCLUSIVE   0x00000100
  #define FMOD_INIT_DSOUND_HRTFNONE   0x00000200
  #define FMOD_INIT_DSOUND_HRTFLIGHT  0x00000400
  #define FMOD_INIT_DSOUND_HRTFFULL  0x00000800
  #define FMOD_INIT_PS2_DISABLECORE0REVERB  0x00010000
  #define FMOD_INIT_PS2_DISABLECORE1REVERB  0x00020000
  #define FMOD_INIT_PS2_DONTUSESCRATCHPAD  0x00040000
  #define FMOD_INIT_PS2_SWAPDMACHANNELS   0x00080000
  #define FMOD_INIT_XBOX_REMOVEHEADROOM   0x00100000
  #define FMOD_INIT_360_MUSICMUTENOTPAUSE   0x00200000
  #define FMOD_INIT_SYNCMIXERWITHUPDATE   0x00400000
```

Values

FMOD_INIT_NORMAL

All platforms - Initialize normally

FMOD_INIT_STREAM_FROM_UPDATE

All platforms - No stream thread is created internally. Streams are driven from System::update. Mainly used with non-realtime outputs.

FMOD_INIT_3D_RIGHTHANDED

All platforms - FMOD will treat +X as right, +Y as up and +Z as backwards (towards you).

FMOD_INIT_SOFTWARE_DISABLE

All platforms - Disable software mixer to save memory. Anything created with FMOD_SOFTWARE will fail and DSP will not work.

FMOD_INIT_SOFTWARE_OCCLUSION

All platforms - All FMOD_SOFTWARE with FMOD_3D based voices will add a software lowpass filter effect into the DSP chain which is automatically used when Channel::set3DOcclusion is used or the geometry API.

FMOD_INIT_SOFTWARE_HRTF

All platforms - All FMOD_SOFTWARE with FMOD_3D based voices will add a software lowpass filter effect into the DSP chain which causes sounds to sound duller when the sound goes behind the listener. Use System::setAdvancedSettings to adjust cutoff frequency.

FMOD_INIT_ENABLE_DSPNET

All platforms - Enable TCP/IP based host which allows "DSPNet Listener.exe" to connect to it, and view the DSP dataflow network graph in real-time.

FMOD_INIT_VOL0_BECOMES_VIRTUAL

All platforms - Any sounds that are 0 volume will go virtual and not be processed except for having their positions updated virtually. Use System::setAdvancedSettings to adjust what volume besides zero to switch to virtual at.

FMOD_INIT_WASAPI_EXCLUSIVE

Win32 Vista only - for WASAPI output - Enable exclusive access to hardware, lower latency at the expense of excluding other applications from accessing the audio hardware.

FMOD_INIT_DSOUND_HRTFNONE

Win32 only - for DirectSound output - FMOD_HARDWARE | FMOD_3D buffers use simple stereo panning/doppler/attenuation when 3D hardware acceleration is not present.

FMOD_INIT_DSOUND_HRTFLIGHT

Win32 only - for DirectSound output - FMOD_HARDWARE | FMOD_3D buffers use a slightly higher quality algorithm when 3D hardware acceleration is not present.

FMOD_INIT_DSOUND_HRTFFULL

Win32 only - for DirectSound output - FMOD_HARDWARE | FMOD_3D buffers use full quality 3D playback when 3d hardware acceleration is not present.

FMOD_INIT_PS2_DISABLECORE0REVERB

PS2 only - Disable reverb on CORE 0 to regain 256k SRAM.

FMOD_INIT_PS2_DISABLECORE1REVERB

PS2 only - Disable reverb on CORE 1 to regain 256k SRAM.

FMOD_INIT_PS2_DONTUSESCRATCHPAD

PS2 only - Disable FMOD's usage of the scratchpad.

FMOD_INIT_PS2_SWAPDMACHANNELS

PS2 only - Changes FMOD from using SPU DMA channel 0 for software mixing, and 1 for sound data upload/file streaming, to 1 and 0 respectively.

FMOD_INIT_XBOX_REMOVEHEADROOM

Xbox only - By default DirectSound attenuates all sound by 6db to avoid clipping/distortion. CAUTION. If you use this flag you are responsible for the final mix to make sure clipping / distortion doesn't happen.

FMOD_INIT_360_MUSICMUTENOTPAUSE

Xbox 360 only - The "music" channelgroup which by default pauses when custom 360 dashboard music is played, can be changed to mute (therefore continues playing) instead of pausing, by using this flag.

FMOD_INIT_SYNCMIXERWITHUPDATE

Win32/Wii/PS3/Xbox/Xbox 360 - FMOD Mixer thread is woken up to do a mix when System::update is called rather than waking periodically on its own timer.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::init
- System::update
- System::setAdvancedSettings
- Channel::set3DOcclusion

FMOD_MEMORY_TYPE

Bit fields for memory allocation type being passed into FMOD memory callbacks.?

Definition

```
   #define FMOD_MEMORY_NORMAL 0x00000000
  #define FMOD_MEMORY_XBOX360_PHYSICAL 0x00100000
  #define FMOD_MEMORY_PERSISTENT 0x00200000
```

Values

FMOD_MEMORY_NORMAL

Standard memory.

FMOD_MEMORY_XBOX360_PHYSICAL

Requires XPhysicalAlloc / XPhysicalFree.

FMOD_MEMORY_PERSISTENT

Persistent memory. Memory will be freed when System::release is called.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_MEMORY_ALLOCCALLBACK
- FMOD_MEMORY_REALLOCCALLBACK
- FMOD_MEMORY_FREECALLBACK
- Memory_Initialize

FMOD_MODE

Sound description bitfields, bitwise OR them together for loading and describing sounds.?By default a sound will open as a static sound that is decompressed fully into memory to PCM. (ie equivalent of FMOD_CREATESAMPLE)
?To have a sound stream instead, use FMOD_CREATESTREAM, or use the wrapper function System::createStream.
?Some opening modes (ie FMOD_OPENUSER, FMOD_OPENMEMORY, FMOD_OPENMEMORY_POINT, FMOD_OPENRAW) will need extra information.
?This can be provided using the FMOD_CREATESOUNDEXINFO structure.?
?On Playstation 2, non VAG formats will default to FMOD_SOFTWARE if FMOD_HARDWARE is not specified.
?This is due to PS2 hardware not supporting PCM data.
?
?Specifying FMOD_OPENMEMORY_POINT will POINT to your memory rather allocating its own sound buffers and duplicating it internally.
?**This means you cannot free the memory while FMOD is using it, until after Sound::release is called.**?With FMOD_OPENMEMORY_POINT, for PCM formats, only WAV, FSB, and RAW are supported. For compressed formats, only those formats supported by FMOD_CREATECOMPRESSEDSAMPLE are supported.
?With FMOD_OPENMEMORY_POINT and FMOD_OPENRAW, if using them together, note that you must pad the data on each side by 16 bytes. This is so fmod can modify the ends of the data for looping/interpolation/mixing purposes.
?
?**Xbox 360 memory** On Xbox 360 Specifying FMOD_OPENMEMORY_POINT to a virtual memory address will cause FMOD_ERR_INVALID_ADDRESS?to be returned. Use physical memory only for this functionality.
?
?FMOD_LOWMEM is used on a sound if you want to minimize the memory overhead, by having FMOD not allocate memory for certain?features that are not likely to be used in a game environment. These are :
?1. Sound::getName functionality is removed. 256 bytes per sound is saved.
?2. For a stream, a default sentence is not created, 4 bytes per subsound. On a 2000 subsound FSB this can save 8kb for example.?Sound::setSubSoundSentence can simply be used to set up a sentence as normal, System::playSound just wont play through the?whole set of subsounds by default any more.
?

Definition

```
    #define FMOD_DEFAULT   0x00000000
   #define FMOD_LOOP_OFF   0x00000001
   #define FMOD_LOOP_NORMAL   0x00000002
   #define FMOD_LOOP_BIDI   0x00000004
   #define FMOD_2D   0x00000008
   #define FMOD_3D   0x00000010
   #define FMOD_HARDWARE   0x00000020
   #define FMOD_SOFTWARE   0x00000040
   #define FMOD_CREATESTREAM   0x00000080
   #define FMOD_CREATESAMPLE   0x00000100
   #define FMOD_CREATECOMPRESSEDSAMPLE   0x00000200
   #define FMOD_OPENUSER   0x00000400
   #define FMOD_OPENMEMORY   0x00000800
   #define FMOD_OPENMEMORY_POINT   0x10000000
   #define FMOD_OPENRAW   0x00001000
   #define FMOD_OPENONLY   0x00002000
   #define FMOD_ACCURATETIME   0x00004000
   #define FMOD_MPEGSEARCH   0x00008000
```

```
#define FMOD_NONBLOCKING 0x00010000
#define FMOD_UNIQUE 0x00020000
#define FMOD_3D_HEADRELATIVE 0x00040000
#define FMOD_3D_WORLDRELATIVE 0x00080000
#define FMOD_3D_LOGROLLOFF 0x00100000
#define FMOD_3D_LINEARROLLOFF 0x00200000
#define FMOD_3D_CUSTOMROLLOFF 0x04000000
#define FMOD_3D_IGNOREGEOMETRY 0x40000000
#define FMOD_CDDA_FORCEASPI 0x00400000
#define FMOD_CDDA_JITTERCORRECT 0x00800000
#define FMOD_UNICODE 0x01000000
#define FMOD_IGNORETAGS 0x02000000
#define FMOD_LOWMEM 0x08000000
#define FMOD_LOADSECONDARYRAM 0x20000000
#define FMOD_VIRTUAL_PLAYFROMSTART 0x80000000
```

Values

FMOD_DEFAULT

FMOD_DEFAULT is a default sound type. Equivalent to all the defaults listed below. FMOD_LOOP_OFF, FMOD_2D, FMOD_HARDWARE.

FMOD_LOOP_OFF

For non looping sounds. (DEFAULT). Overrides FMOD_LOOP_NORMAL / FMOD_LOOP_BIDI.

FMOD_LOOP_NORMAL

For forward looping sounds.

FMOD_LOOP_BIDI

For bidirectional looping sounds. (only works on software mixed static sounds).

FMOD_2D

Ignores any 3d processing. (DEFAULT).

FMOD_3D

Makes the sound positionable in 3D. Overrides FMOD_2D.

FMOD_HARDWARE

Attempts to make sounds use hardware acceleration. (DEFAULT).

FMOD_SOFTWARE

Makes the sound be mixed by the FMOD CPU based software mixer. Overrides FMOD_HARDWARE. Use this for FFT, DSP, compressed sample support, 2D multi-speaker support and other software related features.

FMOD_CREATESTREAM

Decompress at runtime, streaming from the source provided (ie from disk). Overrides FMOD_CREATESAMPLE and FMOD_CREATECOMPRESSEDSAMPLE. Note a stream can only be played once at a time due to a stream only having 1 stream buffer and file handle. Open multiple streams to have them play concurrently.

FMOD_CREATESAMPLE

Decompress at loadtime, decompressing or decoding whole file into memory as the target sample format (ie PCM). Fastest for FMOD_SOFTWARE based playback and most flexible.

FMOD_CREATECOMPRESSEDSAMPLE

Load MP2, MP3, IMAADPCM or XMA into memory and leave it compressed. During playback the FMOD software mixer will decode it in realtime as a 'compressed sample'. Can only be used in combination with FMOD_SOFTWARE. Overrides FMOD_CREATESAMPLE. If the sound data is not ADPCM, MPEG or XMA it will behave as if it was created with FMOD_CREATESAMPLE and decode the sound into PCM.

FMOD_OPENUSER

Opens a user created static sample or stream. Use FMOD_CREATESOUNDEXINFO to specify format and/or read callbacks. If a user created 'sample' is created with no read callback, the sample will be empty. Use Sound::lock and Sound::unlock to place sound data into the sound if this is the case.

FMOD_OPENMEMORY

"name_or_data" will be interpreted as a pointer to memory instead of filename for creating sounds. Use FMOD_CREATESOUNDEXINFO to specify length. FMOD duplicates the memory into its own buffers. Can be freed after open.

FMOD_OPENMEMORY_POINT

"name_or_data" will be interpreted as a pointer to memory instead of filename for creating sounds. Use FMOD_CREATESOUNDEXINFO to specify length. This differs to FMOD_OPENMEMORY in that it uses the memory as is, without duplicating the memory into its own buffers. FMOD_SOFTWARE only. Doesn't work with FMOD_HARDWARE, as sound hardware cannot access main ram on a lot of platforms. Cannot be freed after open, only after Sound::release. Will not work if the data is compressed and FMOD_CREATECOMPRESSEDSAMPLE is not used.

FMOD_OPENRAW

Will ignore file format and treat as raw pcm. Use FMOD_CREATESOUNDEXINFO to specify format. Requires at least defaultfrequency, numchannels and format to be specified before it will open. Must be little endian data.

FMOD_OPENONLY

Just open the file, dont prebuffer or read. Good for fast opens for info, or when sound::readData is to be used.

FMOD_ACCURATETIME

For System::createSound - for accurate Sound::getLength/Channel::setPosition on VBR MP3, and MOD/S3M/XM/IT/MIDI files. Scans file first, so takes longer to open. FMOD_OPENONLY does not affect this.

FMOD_MPEGSEARCH

For corrupted / bad MP3 files. This will search all the way through the file until it hits a valid MPEG header. Normally only searches for 4k.

FMOD_NONBLOCKING

For opening sounds and getting streamed subsounds (seeking) asyncronously. Use Sound::getOpenState to poll the state of the sound as it opens or retrieves the subsound in the background.

FMOD_UNIQUE

Unique sound, can only be played one at a time

FMOD_3D_HEADRELATIVE

Make the sound's position, velocity and orientation relative to the listener.

FMOD_3D_WORLDRELATIVE

Make the sound's position, velocity and orientation absolute (relative to the world). (DEFAULT)

FMOD_3D_LOGROLLOFF

This sound will follow the standard logarithmic rolloff model where mindistance = full volume, maxdistance = where sound stops attenuating, and rolloff is fixed according to the global rolloff factor. (DEFAULT)

FMOD_3D_LINEARROLLOFF

This sound will follow a linear rolloff model where mindistance = full volume, maxdistance = silence. Rolloffscale is ignored.

FMOD_3D_CUSTOMROLLOFF

This sound will follow a rolloff model defined by Sound::set3DCustomRolloff / Channel::set3DCustomRolloff.

FMOD_3D_IGNOREGEOMETRY

Is not affect by geometry occlusion. If not specified in Sound::setMode, or Channel::setMode, the flag is cleared and it is affected by geometry again.

FMOD_CDDA_FORCEASPI

For CDDA sounds only - use ASPI instead of NTSCSI to access the specified CD/DVD device.

FMOD_CDDA_JITTERCORRECT

For CDDA sounds only - perform jitter correction. Jitter correction helps produce a more accurate CDDA stream at the cost of more CPU time.

FMOD_UNICODE

Filename is double-byte unicode.

FMOD_IGNORETAGS

Skips id3v2/asf/etc tag checks when opening a sound, to reduce seek/read overhead when opening files (helps with CD performance).

FMOD_LOWMEM

Removes some features from samples to give a lower memory overhead, like Sound::getName. See remarks.

FMOD_LOADSECONDARYRAM

Load sound into the secondary RAM of supported platform. On PS3, sounds will be loaded into RSX/VRAM.

FMOD_VIRTUAL_PLAYFROMSTART

For sounds that start virtual (due to being quiet or low importance), instead of swapping back to audible, and playing at the correct offset according to time, this flag makes the sound play from the start.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound
- System::createStream
- Sound::setMode
- Sound::getMode
- Channel::setMode
- Channel::getMode
- Sound::set3DCustomRolloff
- Channel::set3DCustomRolloff
- Sound::getOpenState

FMOD_REVERB_CHANNELFLAGS

Values for the Flags member of the FMOD_REVERB_CHANNELPROPERTIES structure.?For EAX4 support with multiple reverb environments, set FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT0,?FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT1 or/and FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT2 in the flags member?of FMOD_REVERB_CHANNELPROPERTIES to specify which environment instance(s) to target. ?Only up to 2 environments to target can be specified at once. Specifying three will result in an error.?If the sound card does not support EAX4, the environment flag is ignored.?

Definition

```
   #define FMOD_REVERB_CHANNELFLAGS_DIRECTHFAUTO   0x00000001
  #define FMOD_REVERB_CHANNELFLAGS_ROOMAUTO   0x00000002
  #define FMOD_REVERB_CHANNELFLAGS_ROOMHFAUTO   0x00000004
  #define FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT0  0x00000008
  #define FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT1  0x00000010
  #define FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT2  0x00000020
  #define FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT3  0x00000040
  #define FMOD_REVERB_CHANNELFLAGS_DEFAULT  (FMOD_REVERB_CHANNELFLAGS_DIRECTHFAUTO |
 FMOD_REVERB_CHANNELFLAGS_ROOMAUTO | FMOD_REVERB_CHANNELFLAGS_ROOMHFAUTO |
 FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT0)
```

Values

FMOD_REVERB_CHANNELFLAGS_DIRECTHFAUTO

Automatic setting of 'Direct' due to distance from listener

FMOD_REVERB_CHANNELFLAGS_ROOMAUTO

Automatic setting of 'Room' due to distance from listener

FMOD_REVERB_CHANNELFLAGS_ROOMHFAUTO

Automatic setting of 'RoomHF' due to distance from listener

FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT0

EAX4/GameCube/Wii. Specify channel to target reverb instance 0.

FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT1

EAX4/GameCube/Wii. Specify channel to target reverb instance 1.

FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT2

EAX4/GameCube/Wii. Specify channel to target reverb instance 2.

FMOD_REVERB_CHANNELFLAGS_ENVIRONMENT3

EAX5. Specify channel to target reverb instance 3.

FMOD_REVERB_CHANNELFLAGS_DEFAULT

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_CHANNELPROPERTIES

FMOD_REVERB_FLAGS

Values for the Flags member of the FMOD_REVERB_PROPERTIES structure.?

Definition

```
   # d f e  FMO D_R E V RB_FLAGS_ ECAY TIMESCA E   0x00000001
  # d f e  FMO D_R E V RB_FLAGS_ R FEC TO NSCA E   0x00000002
  # d f e  FMO D_R E V RB_FLAGS_ R FEC TO N D LAYSCA E   0x00000004
  # d f e  FMO D_R E V RB_FLAGS_ R E RBCA E   0x00000008
  # d f e  FMO D_R E V RB_FLAGS_ R E RBD LAYSCA E   0x00000010
  # d f e  FMO D_R E V RB_FLAGS_ ECAY HFLIMI T  0x00000020
  # d f e  FMO D_R E V RB_FLAGS_EC O TIMESCA E   0x00000040
  # d f e  FMO D_R E V RB_FLAGS_MO DU A TO NTIMESCA E   0x00000080
  # d f e  FMO D_R E V RB_FLAGS_CO E 0   0x00000100
  # d f e  FMO D_R E V RB_FLAGS_CO E1   0x00000200
  # d f e  FMO D_R E V RB_FLAGS_ HG QUA LI T R E V RB  0x00000400
  # d f e  FMO D_R E V RB_FLAGS_ HG QUA LI T DPL2R E V RB  0x00000800
  # d f e  FMO D_R E V RB_FLAGS_ D FAU LT  (FMO D_R E V RB_FLAGS_ ECAY TIMESCA E  |
 FMO D_R E V RB_FLAGS_ R FEC TO NSCA E  |  FMO D_R E V RB_FLAGS_ R FEC TO N D LAYSCA E  |
 FMO D_R E V RB_FLAGS_ R E RBCA E  |  FMO D_R E V RB_FLAGS_ R E RBD LAYSCA E  |
 FMO D_R E V RB_FLAGS_ ECAY HFLIMI T |  FMO D_R E V RB_FLAGS_CO E 0 |  FMO D_R E V RB_FLAGS_CO E1)
```

Values

FMOD_REVERB_FLAGS_DECAYTIMESCALE

'EnvSize' affects reverberation decay time

FMOD_REVERB_FLAGS_REFLECTIONSSCALE

'EnvSize' affects reflection level

FMOD_REVERB_FLAGS_REFLECTIONSDELAYSCALE

'EnvSize' affects initial reflection delay time

FMOD_REVERB_FLAGS_REVERBSCALE

'EnvSize' affects reflections level

FMOD_REVERB_FLAGS_REVERBDELAYSCALE

'EnvSize' affects late reverberation delay time

FMOD_REVERB_FLAGS_DECAYHFLIMIT

AirAbsorptionHF affects DecayHFRatio

FMOD_REVERB_FLAGS_ECHOTIMESCALE

'EnvSize' affects echo time

FMOD_REVERB_FLAGS_MODULATIONTIMESCALE

'EnvSize' affects modulation time

FMOD_REVERB_FLAGS_CORE0

PS2 Only - Reverb is applied to CORE0 (hw voices 0-23)

FMOD_REVERB_FLAGS_CORE1

PS2 Only - Reverb is applied to CORE1 (hw voices 24-47)

FMOD_REVERB_FLAGS_HIGHQUALITYREVERB

GameCube/Wii. Use high quality reverb

FMOD_REVERB_FLAGS_HIGHQUALITYDPL2REVERB

GameCube/Wii. Use high quality DPL2 reverb

FMOD_REVERB_FLAGS_DEFAULT

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_REVERB_PROPERTIES

FMOD_REVERB_PRESETS

A set of predefined environment PARAMETERS, created by Creative Labs?These are used to initialize an FMOD_REVERB_PROPERTIES structure statically.?ie?FMOD_REVERB_PROPERTIES prop = FMOD_PRESET_GENERIC;?

Definition

```
   #define FMOD_PRESET_OFF {  0, -1,  7.5f,  1.00f, -10000, -10000,  0,  1.00f,
 1.00f, 1.0f,  -2602, 0.007f, { 0.0f,0.0f,0.0f} ,   200, 0.011f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f,   0.0f,   0.0f, 0x33f}
  #define FMOD_PRESET_GENERIC  {  0,  0,  7.5f,  1.00f, -1000,  -100,   0,  1.49f,
 0.83f, 1.0f,  -2602, 0.007f, { 0.0f,0.0f,0.0f} ,   200, 0.011f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_PADDEDCELL {  0,  1,  1.4f,  1.00f, -1000,  -6000,  0,   0.17f,
 0.10f, 1.0f,  -1204, 0.001f, { 0.0f,0.0f,0.0f} ,   207, 0.002f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_ROOM  {  0,  2,  1.9f,  1.00f, -1000,  -454,  0,   0.40f,
 0.83f, 1.0f,  -1646, 0.002f, { 0.0f,0.0f,0.0f} ,   53, 0.003f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_BATHROOM  {  0,  3,  1.4f,  1.00f, -1000,  -1200,  0,  1.49f,
 0.54f, 1.0f,   -370, 0.007f, { 0.0f,0.0f,0.0f} , 1030, 0.011f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f,  60.0f, 0x3f}
  #define FMOD_PRESET_LIVINGROOM  {  0,  4,  2.5f,  1.00f, -1000,  -6000,  0,   0.50f,
 0.10f, 1.0f,  -1376, 0.003f, { 0.0f,0.0f,0.0f} , -1104, 0.004f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_STONEROOM  {  0,  5,  11.6f,  1.00f, -1000,  -300,  0,   2.31f,
 0.64f, 1.0f,   -711, 0.012f, { 0.0f,0.0f,0.0f} ,    83, 0.017f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_AUDITORIUM  {  0,  6,  21.6f,  1.00f, -1000,  -476,  0,   4.32f,
 0.59f, 1.0f,   -789, 0.020f, { 0.0f,0.0f,0.0f} , -289, 0.030f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_CONCERTHALL {  0,  7,  19.6f,  1.00f, -1000,  -500,  0,   3.92f,
 0.70f, 1.0f,  -1230, 0.020f, { 0.0f,0.0f,0.0f} ,    -2, 0.029f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_CAVE  {  0,  8,  14.6f,  1.00f, -1000,  0,     0,   2.91f,
 1.30f, 1.0f,   -602, 0.015f, { 0.0f,0.0f,0.0f} , -302, 0.022f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x1f}
  #define FMOD_PRESET_ARENA  {  0,  9,  36.2f,  1.00f, -1000,  -698,  0,  7.24f,
 0.33f, 1.0f,  -1166, 0.020f, { 0.0f,0.0f,0.0f} ,   16, 0.030f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_HANGAR {  0,  10, 50.3f,  1.00f, -1000,  -1000,  0,  10.05f,
 0.23f, 1.0f,   -602, 0.020f, { 0.0f,0.0f,0.0f} ,   198, 0.030f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_CARPETTEDHALLWAY  {  0,  11, 1.9f,  1.00f, -1000,  -4000,  0,
 0.30f,  0.10f, 1.0f,  -1831, 0.002f, { 0.0f,0.0f,0.0f} , -1630, 0.030f, { 0.0f,0.0f,0.0f
} , 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_HALLWAY  {  0,  12, 1.8f,  1.00f, -1000,  -300,  0,  1.49f,
 0.59f, 1.0f,  -1219, 0.007f, { 0.0f,0.0f,0.0f} ,   441, 0.011f, { 0.0f,0.0f,0.0f} ,
 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_STONECORRIDOR {  0,  13, 13.5f,  1.00f, -1000,  -237,  0,
 2.70f,  0.79f, 1.0f,  -1214, 0.013f, { 0.0f,0.0f,0.0f} ,   395, 0.020f, { 0.0f,0.0f,0.0f
} , 0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_ALLEY  {  0,  14, 7.5f,   0.30f, -1000,  -270,  0,  1.49f,
 0.86f, 1.0f,  -1204, 0.007f, { 0.0f,0.0f,0.0f} ,    -4, 0.011f, { 0.0f,0.0f,0.0f} ,
 0.125f, 0.95f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f}
  #define FMOD_PRESET_FOREST {  0,  15, 38.0f,  0.30f, -1000,  -3300,  0,  1.49f,
 0.54f, 1.0f,  -2560, 0.162f, { 0.0f,0.0f,0.0f} , -229, 0.088f, { 0.0f,0.0f,0.0f} ,
0.125f, 1.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 79.0f, 100.0f, 0x3f }
  #define FMOD_PRESET_CITY   {  0, 16, 7.5f,   0.50f, -1000, -800,   0,  1.49f,
0.67f, 1.0f, -2273, 0.007f, { 0.0f,0.0f,0.0f }, -1691, 0.011f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 50.0f, 100.0f, 0x3f }
  #define FMOD_PRESET_MOUNTAINS  {  0, 17, 100.0f, 0.27f, -1000, -2500,  0,  1.49f,
0.21f, 1.0f, -2780, 0.300f, { 0.0f,0.0f,0.0f }, -1434, 0.100f, { 0.0f,0.0f,0.0f },
0.250f, 1.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f,  27.0f, 100.0f, 0x1f }
  #define FMOD_PRESET_QUARRY   {  0, 18, 17.5f, 1.00f, -1000, -1000,  0,  1.49f,
0.83f, 1.0f, -10000, 0.061f, { 0.0f,0.0f,0.0f },  500, 0.025f, { 0.0f,0.0f,0.0f },
0.125f, 0.70f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f }
  #define FMOD_PRESET_PLAIN {  0, 19, 42.5f,  0.21f, -1000, -2000,  0,  1.49f,
0.50f, 1.0f, -2466, 0.179f, { 0.0f,0.0f,0.0f }, -1926, 0.100f, { 0.0f,0.0f,0.0f },
0.250f, 1.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f,  21.0f, 100.0f, 0x3f }
  #define FMOD_PRESET_PARKINGLOT {  0,  20, 8.3f,  1.00f, -1000,  0,      0,  1.65f,
1.50f, 1.0f, -1363, 0.008f, { 0.0f,0.0f,0.0f }, -1153, 0.012f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x1f }
  #define FMOD_PRESET_SEWERPIPE  {  0,  21, 1.7f,  0.80f, -1000, -1000,  0,  2.81f,
0.14f, 1.0f,    429, 0.014f, { 0.0f,0.0f,0.0f },  1023, 0.021f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.25f, 0.000f, -5.0f, 5000.0f, 250.0f, 0.0f,  80.0f,  60.0f, 0x3f }
  #define FMOD_PRESET_UNDERWATER {  0,  22, 1.8f,  1.00f, -1000, -4000,  0,  1.49f,
0.10f, 1.0f,   -449, 0.007f, { 0.0f,0.0f,0.0f },  1700, 0.011f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 1.18f, 0.348f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x3f }
  #define FMOD_PRESET_DRUGGED {  0,  23, 1.9f,  0.50f, -1000,  0,      0,  8.39f,
1.39f, 1.0f,  -115,  0.002f, { 0.0f,0.0f,0.0f },   985, 0.030f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.25f, 1.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x1f }
  #define FMOD_PRESET_DIZZY   {  0,  24, 1.8f,  0.60f, -1000, -400,   0,  17.23f,
0.56f, 1.0f, -1713, 0.020f, { 0.0f,0.0f,0.0f },  -613, 0.030f, { 0.0f,0.0f,0.0f },
0.250f, 1.00f, 0.81f, 0.310f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x1f }
  #define FMOD_PRESET_PSYCHOTIC  {  0,  25, 1.0f,  0.50f, -1000, -151,   0,  7.56f,
0.91f, 1.0f,  -626,  0.020f, { 0.0f,0.0f,0.0f },   774, 0.030f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 4.00f, 1.000f, -5.0f, 5000.0f, 250.0f, 0.0f, 100.0f, 100.0f, 0x1f }
  #define FMOD_PRESET_PS2_ROOM  {  0,  1,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_STUDIO_A  {  0,  2,  0,       0,      0,      0,      0,   0.0f,
 0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_STUDIO_B {  0,  3,  0,       0,      0,      0,      0,   0.0f,
 0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_STUDIO_C  {  0,  4,  0,       0,      0,      0,      0,   0.0f,
 0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_HALL {  0,  5,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_SPACE  {  0,  6,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_ECHO  {  0,  7,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.75f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_DELAY  {  0,  8,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
  #define FMOD_PRESET_PS2_PIPE  {  0,  9,  0,       0,      0,      0,      0,   0.0f,
0.0f,  0.0f,      0,  0.000f, { 0.0f,0.0f,0.0f },      0, 0.000f, { 0.0f,0.0f,0.0f },
0.250f, 0.00f, 0.00f, 0.000f,   0.0f, 0000.0f,   0.0f, 0.0f,   0.0f,   0.0f, 0x31f }
```

Values

FMOD_PRESET_OFF

FMOD_PRESET_GENERIC

FMOD_PRESET_PADDEDCELL

FMOD_PRESET_ROOM

FMOD_PRESET_BATHROOM

FMOD_PRESET_LIVINGROOM

FMOD_PRESET_STONEROOM

FMOD_PRESET_AUDITORIUM

FMOD_PRESET_CONCERTHALL

FMOD_PRESET_CAVE

FMOD_PRESET_ARENA

FMOD_PRESET_HANGAR

FMOD_PRESET_CARPETTEDHALLWAY

FMOD_PRESET_HALLWAY

FMOD_PRESET_STONECORRIDOR

FMOD_PRESET_ALLEY

FMOD_PRESET_FOREST

FMOD_PRESET_CITY

FMOD_PRESET_MOUNTAINS

FMOD_PRESET_QUARRY

FMOD_PRESET_PLAIN

FMOD_PRESET_PARKINGLOT

FMOD_PRESET_SEWERPIPE

FMOD_PRESET_UNDERWATER

FMOD_PRESET_DRUGGED

FMOD_PRESET_DIZZY

FMOD_PRESET_PSYCHOTIC

FMOD_PRESET_PS2_ROOM

FMOD_PRESET_PS2_STUDIO_A

FMOD_PRESET_PS2_STUDIO_B

FMOD_PRESET_PS2_STUDIO_C

FMOD_PRESET_PS2_HALL

FMOD_PRESET_PS2_SPACE

FMOD_PRESET_PS2_ECHO

FMOD_PRESET_PS2_DELAY

FMOD_PRESET_PS2_PIPE

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii, Solaris

See Also

- System::setReverbProperties

FMOD_TIMEUNIT

List of time types that can be returned by Sound::getLength and used with Channel::setPosition or Channel::getPosition.?FMOD_TIMEUNIT_SENTENCE_MS, FMOD_TIMEUNIT_SENTENCE_PCM, FMOD_TIMEUNIT_SENTENCE_PCMBYTES, FMOD_TIMEUNIT_SENTENCE and FMOD_TIMEUNIT_SENTENCE_SUBSOUND are only supported by Channel functions.?

Definition

```
  #define FMOD_TIMEUNIT_MS   0x00000001
 #define FMOD_TIMEUNIT_PCM   0x00000002
 #define FMOD_TIMEUNIT_PCMBYTES   0x00000004
 #define FMOD_TIMEUNIT_RAWBYTES   0x00000008
 #define FMOD_TIMEUNIT_MODORDER  0x00000100
 #define FMOD_TIMEUNIT_MODROW  0x00000200
 #define FMOD_TIMEUNIT_MODPATTERN  0x00000400
 #define FMOD_TIMEUNIT_SENTENCE_MS   0x00010000
 #define FMOD_TIMEUNIT_SENTENCE_PCM   0x00020000
 #define FMOD_TIMEUNIT_SENTENCE_PCMBYTES   0x00040000
 #define FMOD_TIMEUNIT_SENTENCE   0x00080000
 #define FMOD_TIMEUNIT_SENTENCE_SUBSOUND  0x00100000
 #define FMOD_TIMEUNIT_BUFFERED  0x10000000
```

Values

FMOD_TIMEUNIT_MS

Milliseconds.

FMOD_TIMEUNIT_PCM

PCM Samples, related to milliseconds * samplerate / 1000.

FMOD_TIMEUNIT_PCMBYTES

Bytes, related to PCM samples * channels * datawidth (ie 16bit = 2 bytes).

FMOD_TIMEUNIT_RAWBYTES

Raw file bytes of (compressed) sound data (does not include headers). Only used by Sound::getLength and Channel::getPosition.

FMOD_TIMEUNIT_MODORDER

MOD/S3M/XM/IT. Order in a sequenced module format. Use Sound::getFormat to determine the PCM format being decoded to.

FMOD_TIMEUNIT_MODROW

MOD/S3M/XM/IT. Current row in a sequenced module format. Sound::getLength will return the number of rows in the currently playing or seeked to pattern.

FMOD_TIMEUNIT_MODPATTERN

MOD/S3M/XM/IT. Current pattern in a sequenced module format. Sound::getLength will return the number of patterns in the song and Channel::getPosition will return the currently playing pattern.

FMOD_TIMEUNIT_SENTENCE_MS

Currently playing subsound in a sentence time in milliseconds.

FMOD_TIMEUNIT_SENTENCE_PCM

Currently playing subsound in a sentence time in PCM Samples, related to milliseconds * samplerate / 1000.

FMOD_TIMEUNIT_SENTENCE_PCMBYTES

Currently playing subsound in a sentence time in bytes, related to PCM samples * channels * datawidth (ie 16bit = 2 bytes).

FMOD_TIMEUNIT_SENTENCE

Currently playing sentence index according to the channel.

FMOD_TIMEUNIT_SENTENCE_SUBSOUND

Currently playing subsound index in a sentence.

FMOD_TIMEUNIT_BUFFERED

Time value as seen by buffered stream. This is always ahead of audible time, and is only used for processing.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getLength
- Channel::setPosition
- Channel::getPosition

Enumerations

FMOD_CHANNELINDEX

Special channel index values for FMOD functions.?

Enumeration

```
  typedef enum {
    FMOD_CHANNEL_FREE,
    FMOD_CHANNEL_REUSE
} FMOD_CHANNELINDEX
```

Values

FMOD_CHANNEL_FREE

For a channel index, FMOD chooses a free voice using the priority system.

FMOD_CHANNEL_REUSE

For a channel index, re-use the channel handle that was passed in.

Remarks

To get 'all' of the channels, use System::getMasterChannelGroup.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::playSound
- System::playDSP
- System::getChannel
- System::getMasterChannelGroup

FMOD_CHANNEL_CALLBACKTYPE

These callback types are used with Channel::setCallback.?

Enumeration

```
  typedef enum {
    FMOD_CHANNEL_CALLBACKTYPE_END,
    FMOD_CHANNEL_CALLBACKTYPE_VIRTUALVOICE,
    FMOD_CHANNEL_CALLBACKTYPE_SYNCPOINT,
    FMOD_CHANNEL_CALLBACKTYPE_MAX
} FMOD_CHANNEL_CALLBACKTYPE;
```

Values

FMOD_CHANNEL_CALLBACKTYPE_END

Called when a sound ends.

FMOD_CHANNEL_CALLBACKTYPE_VIRTUALVOICE

Called when a voice is swapped out or swapped in.

FMOD_CHANNEL_CALLBACKTYPE_SYNCPOINT

Called when a syncpoint is encountered. Can be from wav file markers.

FMOD_CHANNEL_CALLBACKTYPE_MAX

Maximum number of callback types supported.

Remarks

Each callback has commanddata parameters passed as int unique to the type of callback.
See reference to FMOD_CHANNEL_CALLBACK to determine what they might mean for each type of callback.

Note! Currently the user must call System::update for these callbacks to trigger!

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Channel::setCallback
- FMOD_CHANNEL_CALLBACK
- System::update

FMOD_DSP_CHORUS

Parameter types for the FMOD_DSP_TYPE_CHORUS filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_CHORUS_DRYMIX,
    FMOD_DSP_CHORUS_WETMIX1,
    FMOD_DSP_CHORUS_WETMIX2,
    FMOD_DSP_CHORUS_WETMIX3,
    FMOD_DSP_CHORUS_DELAY,
    FMOD_DSP_CHORUS_RATE,
    FMOD_DSP_CHORUS_DEPTH,
    FMOD_DSP_CHORUS_FEEDBACK
}   FMOD_DSP_CHORUS;
```

Values

FMOD_DSP_CHORUS_DRYMIX

Volume of original signal to pass to output. 0.0 to 1.0. Default = 0.5.

FMOD_DSP_CHORUS_WETMIX1

Volume of 1st chorus tap. 0.0 to 1.0. Default = 0.5.

FMOD_DSP_CHORUS_WETMIX2

Volume of 2nd chorus tap. This tap is 90 degrees out of phase of the first tap. 0.0 to 1.0. Default = 0.5.

FMOD_DSP_CHORUS_WETMIX3

Volume of 3rd chorus tap. This tap is 90 degrees out of phase of the second tap. 0.0 to 1.0. Default = 0.5.

FMOD_DSP_CHORUS_DELAY

Chorus delay in ms. 0.1 to 100.0. Default = 40.0 ms.

FMOD_DSP_CHORUS_RATE

Chorus modulation rate in hz. 0.0 to 20.0. Default = 0.8 hz.

FMOD_DSP_CHORUS_DEPTH

Chorus modulation depth. 0.0 to 1.0. Default = 0.03.

FMOD_DSP_CHORUS_FEEDBACK

Chorus feedback. Controls how much of the wet signal gets fed back into the chorus buffer. 0.0 to 1.0. Default = 0.0.

Remarks

Chrous is an effect where the sound is more 'spacious' due to 1 to 3 versions of the sound being played along side the original signal but with the pitch of each copy modulating on a sine wave.
This is a highly configurable chorus unit. It supports 3 taps, small and large delay times and also feedback.
This unit also could be used to do a simple echo, or a flange effect.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_COMPRESSOR

Parameter types for the FMOD_DSP_TYPE_COMPRESSOR unit.?This is a simple linked multichannel software limiter that is uniform across the whole spectrum.
?

Enumeration

```
  typedef enum {
    FMOD_DSP_COMPRESSOR_THRESHOLD,
    FMOD_DSP_COMPRESSOR_ATTACK,
    FMOD_DSP_COMPRESSOR_RELEASE,
    FMOD_DSP_COMPRESSOR_GAINMAKEUP
}   FMOD_DSP_COMPRESSOR
```

Values

FMOD_DSP_COMPRESSOR_THRESHOLD

Threshold level (dB) in the range from -60 through 0. The default value is 0.

FMOD_DSP_COMPRESSOR_ATTACK

Gain reduction attack time (milliseconds), in the range from 10 through 200. The default value is 50.

FMOD_DSP_COMPRESSOR_RELEASE

Gain reduction release time (milliseconds), in the range from 20 through 1000. The default value is 50.

FMOD_DSP_COMPRESSOR_GAINMAKEUP

Make-up gain (dB) applied after limiting, in the range from 0 through 30. The default value is 0.

Remarks

The limiter is not guaranteed to catch every peak above the threshold level, because it cannot apply gain reduction instantaneously - the time delay is determined by the attack time. However setting the attack time too short will distort the sound, so it is a compromise. High level peaks can be avoided by using a short attack time - but not too short, and setting the threshold a few decibels below the critical level.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::SetParameter
- DSP::GetParameter
- FMOD_DSP_TYPE
- System::addDSP

FMOD_DSP_DISTORTION

Parameter types for the FMOD_DSP_TYPE_DISTORTION filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_DISTORTION_LEVEL
}   FMOD_DSP_DISTORTION
```

Values

FMOD_DSP_DISTORTION_LEVEL

Distortion value. 0.0 to 1.0. Default = 0.5.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_ECHO

Parameter types for the FMOD_DSP_TYPE_ECHO filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_ECHO_DELAY,
    FMOD_DSP_ECHO_DECAYRATIO,
    FMOD_DSP_ECHO_MAXCHANNELS,
    FMOD_DSP_ECHO_DRYMIX,
    FMOD_DSP_ECHO_WETMIX
} FMOD_DSP_ECHO;
```

Values

FMOD_DSP_ECHO_DELAY

Echo delay in ms. 10 to 5000. Default = 500.

FMOD_DSP_ECHO_DECAYRATIO

Echo decay per delay. 0 to 1. 1.0 = No decay, 0.0 = total decay (ie simple 1 line delay). Default = 0.5.

FMOD_DSP_ECHO_MAXCHANNELS

Maximum channels supported. 0 to 16. 0 = same as fmod's default output polyphony, 1 = mono, 2 = stereo etc. See remarks for more. Default = 0. It is suggested to leave at 0!

FMOD_DSP_ECHO_DRYMIX

Volume of original signal to pass to output. 0.0 to 1.0. Default = 1.0.

FMOD_DSP_ECHO_WETMIX

Volume of echo signal to pass to output. 0.0 to 1.0. Default = 1.0.

Remarks

Note. Every time the delay is changed, the plugin re-allocates the echo buffer. This means the echo will dissapear at that time while it refills its new buffer.
Larger echo delays result in larger amounts of memory allocated.

'*maxchannels*' also dictates the amount of memory allocated. By default, the maxchannels value is 0. If FMOD is set to stereo, the echo unit will allocate enough memory for 2 channels. If it is 5.1, it will allocate enough memory for a 6 channel echo, etc.
If the echo effect is only ever applied to the global mix (ie it was added with System::addDSP), then 0 is the value to set as it will be enough to handle all speaker modes.
When the echo is added to a channel (ie Channel::addDSP) then the channel count that comes in could be anything

from 1 to 8 possibly. It is only in this case where you might want to increase the channel count above the output's channel count.
If a channel echo is set to a lower number than the sound's channel count that is coming in, it will not echo the sound.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_FFT_WINDOW

List of windowing methods used in spectrum analysis to reduce leakage / transient signals intefering with the analysis.
?This is a problem with analysis of continuous signals that only have a small portion of the signal sample (the fft window size).
?Windowing the signal with a curve or triangle tapers the sides of the fft window to help alleviate this problem.?

Enumeration

```
  typedef enum {
    FMOD_DSP_FFT_WINDOW_RECT,
    FMOD_DSP_FFT_WINDOW_TRIANGLE,
    FMOD_DSP_FFT_WINDOW_HAMMING,
    FMOD_DSP_FFT_WINDOW_HANNING,
    FMOD_DSP_FFT_WINDOW_BLACKMAN,
    FMOD_DSP_FFT_WINDOW_BLACKMANHARRIS,
    FMOD_DSP_FFT_WINDOW_MAX
}   FMOD_DSP_FFT_WINDOW;
```

Values

FMOD_DSP_FFT_WINDOW_RECT

w[n] = 1.0

FMOD_DSP_FFT_WINDOW_TRIANGLE

w[n] = TRI(2n/N)

FMOD_DSP_FFT_WINDOW_HAMMING

w[n] = 0.54 - (0.46 * COS(n/N))

FMOD_DSP_FFT_WINDOW_HANNING

w[n] = 0.5 * (1.0 - COS(n/N))

FMOD_DSP_FFT_WINDOW_BLACKMAN

w[n] = 0.42 - (0.5 * COS(n/N)) + (0.08 * COS(2.0 * n/N))

FMOD_DSP_FFT_WINDOW_BLACKMANHARRIS

w[n] = 0.35875 - (0.48829 * COS(1.0 * n/N)) + (0.14128 * COS(2.0 * n/N)) - (0.01168 * COS(3.0 * n/N))

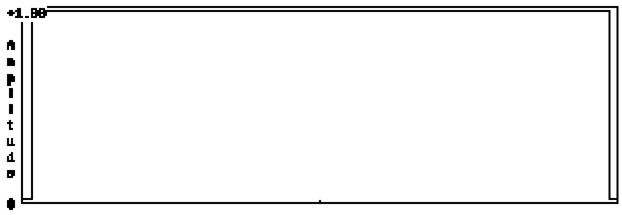
FMOD_DSP_FFT_WINDOW_MAX

Maximum number of FFT window types supported.

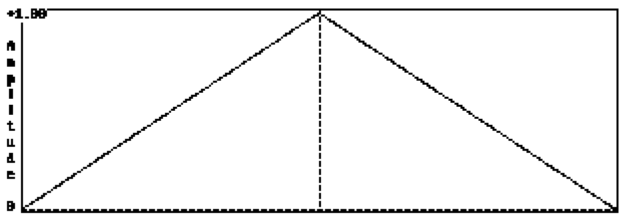
Remarks

Cyclic signals such as a sine wave that repeat their cycle in a multiple of the window size do not need windowing.
I.e. If the sine wave repeats every 1024, 512, 256 etc samples and the FMOD fft window is 1024, then the signal would not need windowing.
Not windowing is the same as FMOD_DSP_FFT_WINDOW_RECT, which is the default.
If the cycle of the signal (ie the sine wave) is not a multiple of the window size, it will cause frequency abnormalities, so a different windowing method is needed.

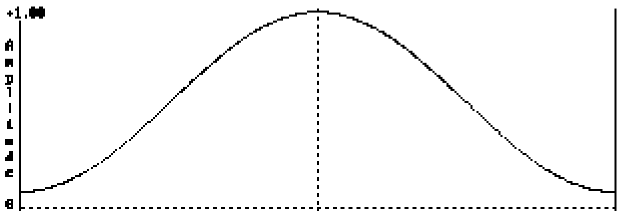
FMOD_DSP_FFT_WINDOW_RECT.



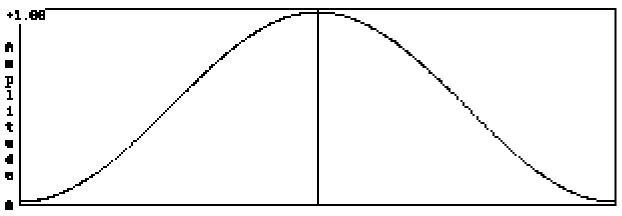
FMOD_DSP_FFT_WINDOW_TRIANGLE.



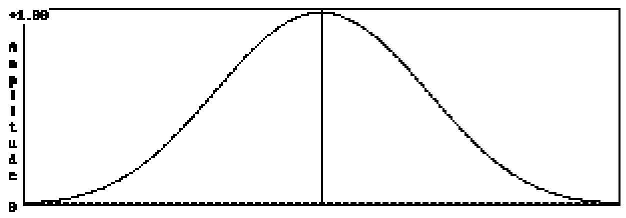
FMOD_DSP_FFT_WINDOW_HAMMING.



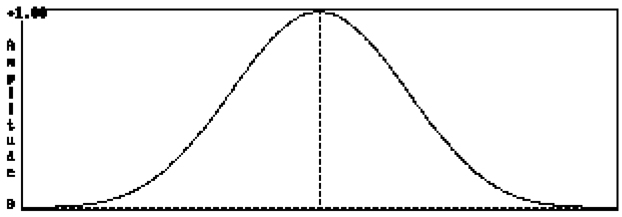
FMOD_DSP_FFT_WINDOW_HANNING.



FMOD_DSP_FFT_WINDOW_BLACKMAN.



FMOD_DSP_FFT_WINDOW_BLACKMANHARRIS.



Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getSpectrum
- Channel::getSpectrum

FMOD_DSP_FLANGE

Parameter types for the FMOD_DSP_TYPE_FLANGE filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_FLANGE_DRYMIX,
    FMOD_DSP_FLANGE_WETMIX,
    FMOD_DSP_FLANGE_DEPTH,
    FMOD_DSP_FLANGE_RATE
} FMOD_DSP_FLANGE;
```

Values

FMOD_DSP_FLANGE_DRYMIX

Volume of original signal to pass to output. 0.0 to 1.0. Default = 0.45.

FMOD_DSP_FLANGE_WETMIX

Volume of flange signal to pass to output. 0.0 to 1.0. Default = 0.55.

FMOD_DSP_FLANGE_DEPTH

Flange depth. 0.01 to 1.0. Default = 1.0.

FMOD_DSP_FLANGE_RATE

Flange speed in hz. 0.0 to 20.0. Default = 0.1.

Remarks

Flange is an effect where the signal is played twice at the same time, and one copy slides back and forth creating a whooshing or flanging effect.
As there are 2 copies of the same signal, by default each signal is given 50% mix, so that the total is not louder than the original unaffected signal.

Flange depth is a percentage of a 10ms shift from the original signal. Anything above 10ms is not considered flange because to the ear it begins to 'echo' so 10ms is the highest value possible.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_HIGHPASS

Parameter types for the FMOD_DSP_TYPE_HIGHPASS filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_HIGHPASS_CUTOFF,
    FMOD_DSP_HIGHPASS_RESONANCE
}   FMOD_DSP_HIGHPASS;
```

Values

FMOD_DSP_HIGHPASS_CUTOFF

Highpass cutoff frequency in hz. 1.0 to output 22000.0. Default = 5000.0.

FMOD_DSP_HIGHPASS_RESONANCE

Highpass resonance Q value. 1.0 to 10.0. Default = 1.0.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_ITECHO

Parameter types for the FMOD_DSP_TYPE_ITECHO filter.
?This is effectively a software based echo filter that emulates the DirectX DMO echo effect. Impulse tracker files can support this, and FMOD will produce the effect on ANY platform, not just those that support DirectX effects!
?

Enumeration

```
  typedef enum {
    FMOD_DSP_ITECHO_WETDRYMIX,
    FMOD_DSP_ITECHO_FEEDBACK,
    FMOD_DSP_ITECHO_LEFTDELAY,
    FMOD_DSP_ITECHO_RIGHTDELAY,
    FMOD_DSP_ITECHO_PANDELAY
}   FMOD_DSP_ITECHO;
```

Values

FMOD_DSP_ITECHO_WETDRYMIX

Ratio of wet (processed) signal to dry (unprocessed) signal. Must be in the range from 0.0 through 100.0 (all wet). The default value is 50.

FMOD_DSP_ITECHO_FEEDBACK

Percentage of output fed back into input, in the range from 0.0 through 100.0. The default value is 50.

FMOD_DSP_ITECHO_LEFTDELAY

Delay for left channel, in milliseconds, in the range from 1.0 through 2000.0. The default value is 500 ms.

FMOD_DSP_ITECHO_RIGHTDELAY

Delay for right channel, in milliseconds, in the range from 1.0 through 2000.0. The default value is 500 ms.

FMOD_DSP_ITECHO_PANDELAY

Value that specifies whether to swap left and right delays with each successive echo. The default value is zero, meaning no swap. Possible values are defined as 0.0 (equivalent to FALSE) and 1.0 (equivalent to TRUE). CURRENTLY NOT SUPPORTED.

Remarks

Note. Every time the delay is changed, the plugin re-allocates the echo buffer. This means the echo will dissapear at that time while it refills its new buffer.
Larger echo delays result in larger amounts of memory allocated.

As this is a stereo filter made mainly for IT playback, it is targeted for stereo signals.

With mono signals only the FMOD_DSP_ITECHO_LEFTDELAY is used.
For multichannel signals (>2) there will be no echo on those channels.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::SetParameter
- DSP::GetParameter
- FMOD_DSP_TYPE
- System::addDSP

FMOD_DSP_ITLOWPASS

Parameter types for the FMOD_DSP_TYPE_ITLOWPASS filter.
?This is different to the default FMOD_DSP_TYPE_ITLOWPASS filter in that it uses a different quality algorithm and is?the filter used to produce the correct sounding playback in .IT files.
?FMOD Ex's .IT playback uses this filter.
?

Enumeration

```
  typedef enum {
   FMOD_DSP_ITLOWPASS_CUTOFF,
   FMOD_DSP_ITLOWPASS_RESONANCE
}  FMOD_DSP_ITLOWPASS;
```

Values

FMOD_DSP_ITLOWPASS_CUTOFF

Lowpass cutoff frequency in hz. 1.0 to 22000.0. Default = 5000.0/

FMOD_DSP_ITLOWPASS_RESONANCE

Lowpass resonance Q value. 0.0 to 127.0. Default = 1.0.

Remarks

Note! This filter actually has a limited cutoff frequency below the specified maximum, due to its limited design, so for a more open range filter use FMOD_DSP_LOWPASS or if you don't mind not having resonance, FMOD_DSP_LOWPASS_SIMPLE.
The effective maximum cutoff is about 8060hz.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_LOWPASS

Parameter types for the FMOD_DSP_TYPE_LOWPASS filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_LOWPASS_CUTOFF,
    FMOD_DSP_LOWPASS_RESONANCE
}   FMOD_DSP_LOWPASS;
```

Values

FMOD_DSP_LOWPASS_CUTOFF

Lowpass cutoff frequency in hz. 10.0 to 22000.0. Default = 5000.0.

FMOD_DSP_LOWPASS_RESONANCE

Lowpass resonance Q value. 1.0 to 10.0. Default = 1.0.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_LOWPASS_SIMPLE

Parameter types for the FMOD_DSP_TYPE_LOWPASS_SIMPLE filter.
?This is a very simple low pass filter, based on two single-pole RC time-constant modules.?The emphasis is on speed rather than accuracy, so this should not be used for task requiring critical filtering.
?

Enumeration

```
  typedef enum {
    FMOD_DSP_LOWPASS_SIMPLE_CUTOFF
} FMOD_DSP_LOWPASS_SIMPLE;
```

Values

FMOD_DSP_LOWPASS_SIMPLE_CUTOFF

Lowpass cutoff frequency in hz. 10.0 to 22000.0. Default = 5000.0

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_NORMALIZE

Parameter types for the FMOD_DSP_TYPE_NORMALIZE filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_NORMALIZE_FADETIME,
    FMOD_DSP_NORMALIZE_THRESHHOLD,
    FMOD_DSP_NORMALIZE_MAXAMP
}   FMOD_DSP_NORMALIZE;
```

Values

FMOD_DSP_NORMALIZE_FADETIME

Time to ramp the silence to full in ms. 0.0 to 20000.0. Default = 5000.0.

FMOD_DSP_NORMALIZE_THRESHHOLD

Lower volume range threshold to ignore. 0.0 to 1.0. Default = 0.1. Raise higher to stop amplification of very quiet signals.

FMOD_DSP_NORMALIZE_MAXAMP

Maximum amplification allowed. 1.0 to 100000.0. Default = 20.0. 1.0 = no amplifaction, higher values allow more boost.

Remarks

Normalize amplifies the sound based on the maximum peaks within the signal.
For example if the maximum peaks in the signal were 50% of the bandwidth, it would scale the whole sound by 2.
The lower threshold value makes the normalizer ignores peaks below a certain point, to avoid over-amplification if a loud signal suddenly came in, and also to avoid amplifying to maximum things like background hiss.

Because FMOD is a realtime audio processor, it doesn't have the luxury of knowing the peak for the whole sound (ie it can't see into the future), so it has to process data as it comes in.
To avoid very sudden changes in volume level based on small samples of new data, fmod fades towards the desired amplification which makes for smooth gain control. The fadetime parameter can control this.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_OSCILLATOR

Parameter types for the FMOD_DSP_TYPE_OSCILLATOR filter.?

Enumeration

```
  typedef enum {
   FMOD_DSP_OSCILLATOR_TYPE,
   FMOD_DSP_OSCILLATOR_RATE
}  FMOD_DSP_OSCILLATOR
```

Values

FMOD_DSP_OSCILLATOR_TYPE

Waveform type. 0 = sine. 1 = square. 2 = sawup. 3 = sawdown. 4 = triangle. 5 = noise.

FMOD_DSP_OSCILLATOR_RATE

Frequency of the sinewave in hz. 1.0 to 22000.0. Default = 220.0.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_PARAMEQ

Parameter types for the FMOD_DSP_TYPE_PARAMEQ filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_PARAMEQ_CENTER,
    FMOD_DSP_PARAMEQ_BANDWIDTH,
    FMOD_DSP_PARAMEQ_GAIN
} FMOD_DSP_PARAMEQ;
```

Values

FMOD_DSP_PARAMEQ_CENTER

Frequency center. 20.0 to 22000.0. Default = 8000.0.

FMOD_DSP_PARAMEQ_BANDWIDTH

Octave range around the center frequency to filter. 0.2 to 5.0. Default = 1.0.

FMOD_DSP_PARAMEQ_GAIN

Frequency Gain. 0.05 to 3.0. Default = 1.0.

Remarks

Parametric EQ is a bandpass filter that attenuates or amplifies a selected frequency and its neighbouring frequencies.

To create a multi-band EQ create multiple FMOD_DSP_TYPE_PARAMEQ units and set each unit to different frequencies, for example 1000hz, 2000hz, 4000hz, 8000hz, 16000hz with a range of 1 octave each.

When a frequency has its gain set to 1.0, the sound will be unaffected and represents the original signal exactly.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_PITCHSHIFT

Parameter types for the FMOD_DSP_TYPE_PITCHSHIFT filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_PITCHSHIFT_PITCH,
    FMOD_DSP_PITCHSHIFT_FFTSIZE,
    FMOD_DSP_PITCHSHIFT_OVERLAP,
    FMOD_DSP_PITCHSHIFT_MAXCHANNELS
}   FMOD_DSP_PITCHSHIFT;
```

Values

FMOD_DSP_PITCHSHIFT_PITCH

Pitch value. 0.5 to 2.0. Default = 1.0. 0.5 = one octave down, 2.0 = one octave up. 1.0 does not change the pitch.

FMOD_DSP_PITCHSHIFT_FFTSIZE

FFT window size. 256, 512, 1024, 2048, 4096. Default = 1024. Increase this to reduce 'smearing'. This effect is a warbling sound similar to when an mp3 is encoded at very low bitrates.

FMOD_DSP_PITCHSHIFT_OVERLAP

Removed. Do not use. FMOD now uses 4 overlaps and cannot be changed.

FMOD_DSP_PITCHSHIFT_MAXCHANNELS

Maximum channels supported. 0 to 16. 0 = same as fmod's default output polyphony, 1 = mono, 2 = stereo etc. See remarks for more. Default = 0. It is suggested to leave at 0!

Remarks

This pitch shifting unit can be used to change the pitch of a sound without speeding it up or slowing it down.
It can also be used for time stretching or scaling, for example if the pitch was doubled, and the frequency of the sound was halved, the pitch of the sound would sound correct but it would be twice as slow.

Warning! This filter is very computationally expensive! Similar to a vocoder, it requires several overlapping FFT and IFFT's to produce smooth output, and can require around 440mhz for 1 stereo 48khz signal using the default settings.
Reducing the signal to mono will half the cpu usage.
Reducing this will lower audio quality, but what settings to use are largely dependant on the sound being played. A noisy polyphonic signal will need higher fft size compared to a speaking voice for example.

This pitch shifter is based on the pitch shifter code at http://www.dspdimension.com, written by Stephan M. Bernsee.
The original code is COPYRIGHT 1999-2003 Stephan M. Bernsee .

'*maxchannels*' dictates the amount of memory allocated. By default, the maxchannels value is 0. If FMOD is set to

stereo, the pitch shift unit will allocate enough memory for 2 channels. If it is 5.1, it will allocate enough memory for a 6 channel pitch shift, etc.
If the pitch shift effect is only ever applied to the global mix (ie it was added with System::addDSP), then 0 is the value to set as it will be enough to handle all speaker modes.
When the pitch shift is added to a channel (ie Channel::addDSP) then the channel count that comes in could be anything from 1 to 8 possibly. It is only in this case where you might want to increase the channel count above the output's channel count.
If a channel pitch shift is set to a lower number than the sound's channel count that is coming in, it will not pitch shift the sound.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_RESAMPLER

List of interpolation types that the FMOD Ex software mixer supports.?

Enumeration

```
  typedef enum {
    FMOD_DSP_RESAMPLER_NOINTERP,
    FMOD_DSP_RESAMPLER_LINEAR,
    FMOD_DSP_RESAMPLER_CUBIC,
    FMOD_DSP_RESAMPLER_SPLINE,
    FMOD_DSP_RESAMPLER_MAX
}   FMOD_DSP_RESAMPLER;
```

Values

FMOD_DSP_RESAMPLER_NOINTERP

No interpolation. High frequency aliasing hiss will be audible depending on the sample rate of the sound.

FMOD_DSP_RESAMPLER_LINEAR

Linear interpolation (default method). Fast and good quality, causes very slight lowpass effect on low frequency sounds.

FMOD_DSP_RESAMPLER_CUBIC

Cubic interoplation. Slower than linear interpolation but better quality.

FMOD_DSP_RESAMPLER_SPLINE

5 point spline interoplation. Slowest resampling method but best quality.

FMOD_DSP_RESAMPLER_MAX

Maximum number of resample methods supported.

Remarks

The default resampler type is FMOD_DSP_RESAMPLER_LINEAR.
Use System::setSoftwareFormat to tell FMOD the resampling quality you require for FMOD_SOFTWARE based sounds.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setSoftwareFormat
- System::getSoftwareFormat

FMOD_DSP_REVERB

Parameter types for the FMOD_DSP_TYPE_REVERB filter.?

Enumeration

```
  typedef enum {
    FMOD_DSP_REVERB_ROOMSIZE,
    FMOD_DSP_REVERB_DAMP,
    FMOD_DSP_REVERB_WETMIX,
    FMOD_DSP_REVERB_DRYMIX,
    FMOD_DSP_REVERB_WIDTH,
    FMOD_DSP_REVERB_MODE
} FMOD_DSP_REVERB;
```

Values

FMOD_DSP_REVERB_ROOMSIZE

Roomsize. 0.0 to 1.0. Default = 0.5

FMOD_DSP_REVERB_DAMP

Damp. 0.0 to 1.0. Default = 0.5

FMOD_DSP_REVERB_WETMIX

Wet mix. 0.0 to 1.0. Default = 0.33

FMOD_DSP_REVERB_DRYMIX

Dry mix. 0.0 to 1.0. Default = 0.66

FMOD_DSP_REVERB_WIDTH

Stereo width. 0.0 to 1.0. Default = 1.0

FMOD_DSP_REVERB_MODE

Mode. 0 (normal), 1 (freeze). Default = 0

Remarks

Based on freeverb by Jezar at Dreampoint - http://www.dreampoint.co.uk.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- DSP::setParameter
- DSP::getParameter
- FMOD_DSP_TYPE

FMOD_DSP_SFXREVERB

Parameter types for the FMOD_DSP_TYPE_SFXREVERB unit.
?

Enumeration

```
  typedef enum {
    FMOD_DSP_SFXREVERB_DRYLEVEL,
    FMOD_DSP_SFXREVERB_ROOM,
    FMOD_DSP_SFXREVERB_ROOMHF,
    FMOD_DSP_SFXREVERB_ROOMROLLOFFFACTOR,
    FMOD_DSP_SFXREVERB_DECAYTIME,
    FMOD_DSP_SFXREVERB_DECAYHFRATIO,
    FMOD_DSP_SFXREVERB_REFLECTIONSLEVEL,
    FMOD_DSP_SFXREVERB_REFLECTIONSDELAY,
    FMOD_DSP_SFXREVERB_REVERBLEVEL,
    FMOD_DSP_SFXREVERB_REVERBDELAY,
    FMOD_DSP_SFXREVERB_DIFFUSION,
    FMOD_DSP_SFXREVERB_DENSITY,
    FMOD_DSP_SFXREVERB_HFREFERENCE,
    FMOD_DSP_SFXREVERB_ROOMLF,
    FMOD_DSP_SFXREVERB_LFREFERENCE
} FMOD_DSP_SFXREVERB
```

Values

FMOD_DSP_SFXREVERB_DRYLEVEL

Dry Level : Mix level of dry signal in output in mB. Ranges from -10000.0 to 0.0. Default is 0.

FMOD_DSP_SFXREVERB_ROOM

Room : Room effect level at low frequencies in mB. Ranges from -10000.0 to 0.0. Default is 0.0.

FMOD_DSP_SFXREVERB_ROOMHF

Room HF : Room effect high-frequency level re. low frequency level in mB. Ranges from -10000.0 to 0.0. Default is 0.0.

FMOD_DSP_SFXREVERB_ROOMROLLOFFFACTOR

Room Rolloff : Like DS3D flRolloffFactor but for room effect. Ranges from 0.0 to 10.0. Default is 10.0

FMOD_DSP_SFXREVERB_DECAYTIME

Decay Time : Reverberation decay time at low-frequencies in seconds. Ranges from 0.1 to 20.0. Default is 1.0.

FMOD_DSP_SFXREVERB_DECAYHFRATIO

Decay HF Ratio : High-frequency to low-frequency decay time ratio. Ranges from 0.1 to 2.0. Default is 0.5.

FMOD_DSP_SFXREVERB_REFLECTIONSLEVEL

Reflections : Early reflections level relative to room effect in mB. Ranges from -10000.0 to 1000.0. Default is -10000.0.

FMOD_DSP_SFXREVERB_REFLECTIONSDELAY

Reflect Delay : Delay time of first reflection in seconds. Ranges from 0.0 to 0.3. Default is 0.02.

FMOD_DSP_SFXREVERB_REVERBLEVEL

Reverb : Late reverberation level relative to room effect in mB. Ranges from -10000.0 to 2000.0. Default is 0.0.

FMOD_DSP_SFXREVERB_REVERBDELAY

Reverb Delay : Late reverberation delay time relative to first reflection in seconds. Ranges from 0.0 to 0.1. Default is 0.04.

FMOD_DSP_SFXREVERB_DIFFUSION

Diffusion : Reverberation diffusion (echo density) in percent. Ranges from 0.0 to 100.0. Default is 100.0.

FMOD_DSP_SFXREVERB_DENSITY

Density : Reverberation density (modal density) in percent. Ranges from 0.0 to 100.0. Default is 100.0.

FMOD_DSP_SFXREVERB_HFREFERENCE

HF Reference : Reference high frequency in Hz. Ranges from 20.0 to 20000.0. Default is 5000.0.

FMOD_DSP_SFXREVERB_ROOMLF

Room LF : Room effect low-frequency level in mB. Ranges from -10000.0 to 0.0. Default is 0.0.

FMOD_DSP_SFXREVERB_LFREFERENCE

LF Reference : Reference low-frequency in Hz. Ranges from 20.0 to 1000.0. Default is 250.0.

Remarks

This is a high quality I3DL2 based reverb which improves greatly on FMOD_DSP_REVERB.
On top of the I3DL2 property set, "Dry Level" is also included to allow the dry mix to be changed.

Currently FMOD_DSP_SFXREVERB_REFLECTIONSLEVEL, FMOD_DSP_SFXREVERB_REFLECTIONSDELAY and FMOD_DSP_SFXREVERB_REVERBDELAY are not enabled but will come in future versions.

These properties can be set with presets in FMOD_REVERB_PRESETS.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii

See Also

- DSP::SetParameter
- DSP::GetParameter
- FMOD_DSP_TYPE
- System::addDSP
- FMOD_REVERB_PRESETS

FMOD_DSP_TYPE

These definitions can be used for creating FMOD defined special effects or DSP units.?

Enumeration

```
  typedef enum {
    FMOD_DSP_TYPE_UNKNOWN,
    FMOD_DSP_TYPE_MIXER,
    FMOD_DSP_TYPE_OSCILLATOR,
    FMOD_DSP_TYPE_LOWPASS,
    FMOD_DSP_TYPE_ITLOWPASS,
    FMOD_DSP_TYPE_HIGHPASS,
    FMOD_DSP_TYPE_ECHO,
    FMOD_DSP_TYPE_FLANGE,
    FMOD_DSP_TYPE_DISTORTION,
    FMOD_DSP_TYPE_NORMALIZE,
    FMOD_DSP_TYPE_PARAMEQ,
    FMOD_DSP_TYPE_PITCHSHIFT,
    FMOD_DSP_TYPE_CHORUS,
    FMOD_DSP_TYPE_REVERB,
    FMOD_DSP_TYPE_VSTPLUGIN,
    FMOD_DSP_TYPE_WINAMPPLUGIN,
    FMOD_DSP_TYPE_ITECHO,
    FMOD_DSP_TYPE_COMPRESSOR,
    FMOD_DSP_TYPE_SFXREVERB,
    FMOD_DSP_TYPE_LOWPASS_SIMPLE
}   FMOD_DSP_TYPE;
```

Values

FMOD_DSP_TYPE_UNKNOWN

This unit was created via a non FMOD plugin so has an unknown purpose.

FMOD_DSP_TYPE_MIXER

This unit does nothing but take inputs and mix them together then feed the result to the soundcard unit.

FMOD_DSP_TYPE_OSCILLATOR

This unit generates sine/square/saw/triangle or noise tones.

FMOD_DSP_TYPE_LOWPASS

This unit filters sound using a high quality, resonant lowpass filter algorithm but consumes more CPU time.

FMOD_DSP_TYPE_ITLOWPASS

This unit filters sound using a resonant lowpass filter algorithm that is used in Impulse Tracker, but with limited cutoff range (0 to 8060hz).

FMOD_DSP_TYPE_HIGHPASS

This unit filters sound using a resonant highpass filter algorithm.

FMOD_DSP_TYPE_ECHO

This unit produces an echo on the sound and fades out at the desired rate.

FMOD_DSP_TYPE_FLANGE

This unit produces a flange effect on the sound.

FMOD_DSP_TYPE_DISTORTION

This unit distorts the sound.

FMOD_DSP_TYPE_NORMALIZE

This unit normalizes or amplifies the sound to a certain level.

FMOD_DSP_TYPE_PARAMEQ

This unit attenuates or amplifies a selected frequency range.

FMOD_DSP_TYPE_PITCHSHIFT

This unit bends the pitch of a sound without changing the speed of playback.

FMOD_DSP_TYPE_CHORUS

This unit produces a chorus effect on the sound.

FMOD_DSP_TYPE_REVERB

This unit produces a reverb effect on the sound.

FMOD_DSP_TYPE_VSTPLUGIN

This unit allows the use of Steinberg VST plugins

FMOD_DSP_TYPE_WINAMPPLUGIN

This unit allows the use of Nullsoft Winamp plugins

FMOD_DSP_TYPE_ITECHO

This unit produces an echo on the sound and fades out at the desired rate as is used in Impulse Tracker.

FMOD_DSP_TYPE_COMPRESSOR

This unit implements dynamic compression (linked multichannel, wideband)

FMOD_DSP_TYPE_SFXREVERB

This unit implements SFX reverb

FMOD_DSP_TYPE_LOWPASS_SIMPLE

This unit filters sound using a simple lowpass with no resonance, but has flexible cutoff and is fast.

Remarks

To get them to be active, first create the unit, then add it somewhere into the DSP network, either at the front of the network near the soundcard unit to affect the global output (by using System::getDSPHead), or on a single channel (using Channel::getDSPHead).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- System::createDSPByType

FMOD_OPENSTATE

These values describe what state a sound is in after FMOD_NONBLOCKING has been used to open it.?

Enumeration

```
typedef enum {
  FMOD_OPENSTATE_READY,
  FMOD_OPENSTATE_LOADING,
  FMOD_OPENSTATE_ERROR,
  FMOD_OPENSTATE_CONNECTING,
  FMOD_OPENSTATE_BUFFERING,
  FMOD_OPENSTATE_SEEKING,
  FMOD_OPENSTATE_MAX
} FMOD_OPENSTATE;
```

Values

FMOD_OPENSTATE_READY

Opened and ready to play.

FMOD_OPENSTATE_LOADING

Initial load in progress.

FMOD_OPENSTATE_ERROR

Failed to open - file not found, out of memory etc. See return value of Sound::getOpenState for what happened.

FMOD_OPENSTATE_CONNECTING

Connecting to remote host (internet sounds only).

FMOD_OPENSTATE_BUFFERING

Buffering data.

FMOD_OPENSTATE_SEEKING

Seeking to subsound and re-flushing stream buffer.

FMOD_OPENSTATE_MAX

Maximum number of open state types.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getOpenState
- FMOD_MODE

FMOD_OUTPUTTYPE

These output types are used with System::setOutput/System::getOutput, to choose which output method to use.?

Enumeration

```
  typedef enum {
    FMOD_OUTPUTTYPE_AUTODETECT,
    FMOD_OUTPUTTYPE_UNKNOWN,
    FMOD_OUTPUTTYPE_NOSOUND,
    FMOD_OUTPUTTYPE_WAVWRITER,
    FMOD_OUTPUTTYPE_NOSOUND_NRT,
    FMOD_OUTPUTTYPE_WAVWRITER_NRT,
    FMOD_OUTPUTTYPE_DSOUND,
    FMOD_OUTPUTTYPE_WINMM,
    FMOD_OUTPUTTYPE_OPENAL,
    FMOD_OUTPUTTYPE_WASAPI,
    FMOD_OUTPUTTYPE_ASIO,
    FMOD_OUTPUTTYPE_OSS,
    FMOD_OUTPUTTYPE_ALSA,
    FMOD_OUTPUTTYPE_ESD,
    FMOD_OUTPUTTYPE_SOUNDMANAGER,
    FMOD_OUTPUTTYPE_COREAUDIO,
    FMOD_OUTPUTTYPE_XBOX,
    FMOD_OUTPUTTYPE_PS2,
    FMOD_OUTPUTTYPE_PS3,
    FMOD_OUTPUTTYPE_GC,
    FMOD_OUTPUTTYPE_XBOX360,
    FMOD_OUTPUTTYPE_PSP,
    FMOD_OUTPUTTYPE_WII,
    FMOD_OUTPUTTYPE_MAX
} FMOD_OUTPUTTYPE;
```

Values

FMOD_OUTPUTTYPE_AUTODETECT

Picks the best output mode for the platform. This is the default.

FMOD_OUTPUTTYPE_UNKNOWN

All - 3rd party plugin, unknown. This is for use with System::getOutput only.

FMOD_OUTPUTTYPE_NOSOUND

All - All calls in this mode succeed but make no sound.

FMOD_OUTPUTTYPE_WAVWRITER

All - Writes output to fmodoutput.wav by default. Use the 'extradriverdata' parameter in System::init, by simply passing the filename as a string, to set the wav filename.

FMOD_OUTPUTTYPE_NOSOUND_NRT

All - Non-realtime version of FMOD_OUTPUTTYPE_NOSOUND. User can drive mixer with System::update at whatever rate they want.

FMOD_OUTPUTTYPE_WAVWRITER_NRT

All - Non-realtime version of FMOD_OUTPUTTYPE_WAVWRITER. User can drive mixer with System::update at whatever rate they want.

FMOD_OUTPUTTYPE_DSOUND

Win32/Win64 - DirectSound output. Use this to get hardware accelerated 3d audio and EAX Reverb support. (Default on Windows)

FMOD_OUTPUTTYPE_WINMM

Win32/Win64 - Windows Multimedia output.

FMOD_OUTPUTTYPE_OPENAL

Win32/Win64 - OpenAL 1.1 output. Use this for lower CPU overhead than FMOD_OUTPUTTYPE_DSOUND, and also Vista H/W support with Creative Labs cards.

FMOD_OUTPUTTYPE_WASAPI

Win32 - Windows Audio Session API. (Default on Windows Vista)

FMOD_OUTPUTTYPE_ASIO

Win32 - Low latency ASIO driver.

FMOD_OUTPUTTYPE_OSS

Linux - Open Sound System output. (Default on Linux)

FMOD_OUTPUTTYPE_ALSA

Linux - Advanced Linux Sound Architecture output.

FMOD_OUTPUTTYPE_ESD

Linux - Enlightment Sound Daemon output.

FMOD_OUTPUTTYPE_SOUNDMANAGER

Mac - Macintosh SoundManager output. (Default on Mac carbon library)

FMOD_OUTPUTTYPE_COREAUDIO

Mac - Macintosh CoreAudio output. (Default on Mac OSX library)

FMOD_OUTPUTTYPE_XBOX

Xbox - Native hardware output. (Default on Xbox)

FMOD_OUTPUTTYPE_PS2

PS2 - Native hardware output. (Default on PS2)

FMOD_OUTPUTTYPE_PS3

PS3 - Native hardware output. (Default on PS3)

FMOD_OUTPUTTYPE_GC

GameCube - Native hardware output. (Default on GameCube)

FMOD_OUTPUTTYPE_XBOX360

Xbox 360 - Native hardware output. (Default on Xbox 360)

FMOD_OUTPUTTYPE_PSP

PSP - Native hardware output. (Default on PSP)

FMOD_OUTPUTTYPE_WII

Wii - Native hardware output. (Default on Wii)

FMOD_OUTPUTTYPE_MAX

Maximum number of output types supported.

Remarks

To pass information to the driver when initializing fmod use the extradriverdata parameter in System::init for the following reasons.

- FMOD_OUTPUTTYPE_WAVWRITER - extradriverdata is a pointer to a char * filename that the wav writer will output to.
- FMOD_OUTPUTTYPE_WAVWRITER_NRT - extradriverdata is a pointer to a char * filename that the wav writer will output to.
- FMOD_OUTPUTTYPE_DSOUND - extradriverdata is a pointer to a HWND so that FMOD can set the focus on the audio for a particular window.
- FMOD_OUTPUTTYPE_GC - extradriverdata is a pointer to a FMOD_GC_INFO struct. This can be found in fmodgc.h.
- FMOD_OUTPUTTYPE_ALSA - extradriverdata is a pointer to a char * argument if required by the chosen ALSA driver. Currently these are the only FMOD drivers that take extra information. Other unknown plugins may have different requirements.

Note! If FMOD_OUTPUTTYPE_WAVWRITER_NRT or FMOD_OUTPUTTYPE_NOSOUND_NRT are used, and if the System::update function is being called very quickly (ie for a non realtime decode) it may be being called too quickly for the FMOD streamer thread to respond to. The result will be a skipping/stuttering output in the captured audio.
To remedy this, disable the FMOD Ex streamer thread, and use FMOD_INIT_STREAM_FROM_UPDATE can be used to avoid skipping in the output stream, as it will lock the mixer and the streamer together in the same thread.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setOutput
- System::getOutput
- System::setSoftwareFormat
- System::getSoftwareFormat
- System::init
- System::update
- FMOD_INITFLAGS

FMOD_PLUGINTYPE

These are plugin types defined for use with the System::getNumPlugins,?System::getPluginInfo and System::unloadPlugin functions.?

Enumeration

```
typedef enum {
  FMOD_PLUGINTYPE_OUTPUT,
  FMOD_PLUGINTYPE_CODEC,
  FMOD_PLUGINTYPE_DSP,
  FMOD_PLUGINTYPE_MAX
} FMOD_PLUGINTYPE;
```

Values

FMOD_PLUGINTYPE_OUTPUT

The plugin type is an output module. FMOD mixed audio will play through one of these devices

FMOD_PLUGINTYPE_CODEC

The plugin type is a file format codec. FMOD will use these codecs to load file formats for playback.

FMOD_PLUGINTYPE_DSP

The plugin type is a DSP unit. FMOD will use these plugins as part of its DSP network to apply effects to output or generate sound in realtime.

FMOD_PLUGINTYPE_MAX

Maximum number of plugin types supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::getNumPlugins
- System::getPluginInfo
- System::unloadPlugin

FMOD_RESULT

error codes. Returned from every function.?

Enumeration

```
typedef enum {
  FMOD_OK,
  FMOD_ERR_ALREADYLOCKED,
  FMOD_ERR_BADCOMMAND,
  FMOD_ERR_CDDA_DRIVERS,
  FMOD_ERR_CDDA_INIT,
  FMOD_ERR_CDDA_INVALID_DEVICE,
  FMOD_ERR_CDDA_NOAUDIO,
  FMOD_ERR_CDDA_NODEVICES,
  FMOD_ERR_CDDA_NODISC,
  FMOD_ERR_CDDA_READ,
  FMOD_ERR_CHANNEL_ALLOC,
  FMOD_ERR_CHANNEL_STOLEN,
  FMOD_ERR_COM,
  FMOD_ERR_DMA,
  FMOD_ERR_DSP_CONNECTION,
  FMOD_ERR_DSP_FORMAT,
  FMOD_ERR_DSP_NOTFOUND,
  FMOD_ERR_DSP_RUNNING,
  FMOD_ERR_DSP_TOOMANYCONNECTIONS,
  FMOD_ERR_FILE_BAD,
  FMOD_ERR_FILE_COULDNOTSEEK,
  FMOD_ERR_FILE_DISKEJECTED,
  FMOD_ERR_FILE_EOF,
  FMOD_ERR_FILE_NOTFOUND,
  FMOD_ERR_FILE_UNWANTED,
  FMOD_ERR_FORMAT,
  FMOD_ERR_HTTP,
  FMOD_ERR_HTTP_ACCESS,
  FMOD_ERR_HTTP_PROXY_AUTH,
  FMOD_ERR_HTTP_SERVER_ERROR,
  FMOD_ERR_HTTP_TIMEOUT,
  FMOD_ERR_INITIALIZATION,
  FMOD_ERR_INITIALIZED,
  FMOD_ERR_INTERNAL,
  FMOD_ERR_INVALID_ADDRESS,
  FMOD_ERR_INVALID_FLOAT,
  FMOD_ERR_INVALID_HANDLE,
  FMOD_ERR_INVALID_PARAM,
  FMOD_ERR_INVALID_SPEAKER,
  FMOD_ERR_INVALID_VECTOR,
  FMOD_ERR_IRX,
  FMOD_ERR_MAXAUDIBLE,
  FMOD_ERR_MEMORY,
  FMOD_ERR_MEMORY_IOP,
  FMOD_ERR_MEMORY_SRAM,
  FMOD_ERR_MEMORY_CANTPOINT,
  FMOD_ERR_NEEDS2D,
  FMOD_ERR_NEEDS3D,
  FMOD_ERR_NEEDSHARDWARE,
  FMOD_ERR_NEEDSSOFTWARE,
  FMOD_ERR_NET_CONNECT,
    FMOD_ERR_NET_SOCKET_ERROR,
    FMOD_ERR_NET_URL,
    FMOD_ERR_NET_WOULD_BLOCK,
    FMOD_ERR_NOTREADY,
    FMOD_ERR_OUTPUT_ALLOCATED,
    FMOD_ERR_OUTPUT_CREATEBUFFER,
    FMOD_ERR_OUTPUT_DRIVERCALL,
    FMOD_ERR_OUTPUT_ENUMERATION,
    FMOD_ERR_OUTPUT_FORMAT,
    FMOD_ERR_OUTPUT_INIT,
    FMOD_ERR_OUTPUT_NOHARDWARE,
    FMOD_ERR_OUTPUT_NOSOFTWARE,
    FMOD_ERR_PAN,
    FMOD_ERR_PLUGIN,
    FMOD_ERR_PLUGIN_MISSING,
    FMOD_ERR_PLUGIN_RESOURCE,
    FMOD_ERR_PLUGIN_INSTANCES,
    FMOD_ERR_RECORD,
    FMOD_ERR_REVERB_INSTANCE,
    FMOD_ERR_SUBSOUNDS,
    FMOD_ERR_SUBSOUND_ALLOCATED,
    FMOD_ERR_TAGNOTFOUND,
    FMOD_ERR_TOOMANYCHANNELS,
    FMOD_ERR_UNIMPLEMENTED,
    FMOD_ERR_UNINITIALIZED,
    FMOD_ERR_UNSUPPORTED,
    FMOD_ERR_UPDATE,
    FMOD_ERR_VERSION,
    FMOD_ERR_EVENT_FAILED,
    FMOD_ERR_EVENT_INTERNAL,
    FMOD_ERR_EVENT_INFOONLY,
    FMOD_ERR_EVENT_MAXSTREAMS,
    FMOD_ERR_EVENT_MISMATCH,
    FMOD_ERR_EVENT_NAMECONFLICT,
    FMOD_ERR_EVENT_NOTFOUND
} FMOD_RESULT;
```

Values

FMOD_OK

No errors.

FMOD_ERR_ALREADYLOCKED

Tried to call lock a second time before unlock was called.

FMOD_ERR_BADCOMMAND

Tried to call a function on a data type that does not allow this type of functionality (ie calling Sound::lock on a streaming sound).

FMOD_ERR_CDDA_DRIVERS

Neither NTSCSI nor ASPI could be initialised.

FMOD_ERR_CDDA_INIT

An error occurred while initialising the CDDA subsystem.

FMOD_ERR_CDDA_INVALID_DEVICE

Couldn't find the specified device.

FMOD_ERR_CDDA_NOAUDIO

No audio tracks on the specified disc.

FMOD_ERR_CDDA_NODEVICES

No CD/DVD devices were found.

FMOD_ERR_CDDA_NODISC

No disc present in the specified drive.

FMOD_ERR_CDDA_READ

A CDDA read error occurred.

FMOD_ERR_CHANNEL_ALLOC

Error trying to allocate a channel.

FMOD_ERR_CHANNEL_STOLEN

The specified channel has been reused to play another sound.

FMOD_ERR_COM

A Win32 COM related error occured. COM failed to initialize or a QueryInterface failed meaning a Windows codec or driver was not installed properly.

FMOD_ERR_DMA

DMA Failure. See debug output for more information.

FMOD_ERR_DSP_CONNECTION

DSP connection error. Connection possibly caused a cyclic dependancy.

FMOD_ERR_DSP_FORMAT

DSP Format error. A DSP unit may have attempted to connect to this network with the wrong format.

FMOD_ERR_DSP_NOTFOUND

DSP connection error. Couldn't find the DSP unit specified.

FMOD_ERR_DSP_RUNNING

DSP error. Cannot perform this operation while the network is in the middle of running. This will most likely happen if a connection or disconnection is attempted in a DSP callback.

FMOD_ERR_DSP_TOOMANYCONNECTIONS

DSP connection error. The unit being connected to or disconnected should only have 1 input or output.

FMOD_ERR_FILE_BAD

Error loading file.

FMOD_ERR_FILE_COULDNOTSEEK

Couldn't perform seek operation. This is a limitation of the medium (ie netstreams) or the file format.

FMOD_ERR_FILE_DISKEJECTED

Media was ejected while reading.

FMOD_ERR_FILE_EOF

End of file unexpectedly reached while trying to read essential data (truncated data?).

FMOD_ERR_FILE_NOTFOUND

File not found.

FMOD_ERR_FILE_UNWANTED

Unwanted file access occured.

FMOD_ERR_FORMAT

Unsupported file or audio format.

FMOD_ERR_HTTP

A HTTP error occurred. This is a catch-all for HTTP errors not listed elsewhere.

FMOD_ERR_HTTP_ACCESS

The specified resource requires authentication or is forbidden.

FMOD_ERR_HTTP_PROXY_AUTH

Proxy authentication is required to access the specified resource.

FMOD_ERR_HTTP_SERVER_ERROR

A HTTP server error occurred.

FMOD_ERR_HTTP_TIMEOUT

The HTTP request timed out.

FMOD_ERR_INITIALIZATION

FMOD was not initialized correctly to support this function.

FMOD_ERR_INITIALIZED

Cannot call this command after System::init.

FMOD_ERR_INTERNAL

An error occured that wasn't supposed to. Contact support.

FMOD_ERR_INVALID_ADDRESS

On Xbox 360, this memory address passed to FMOD must be physical, (ie allocated with XPhysicalAlloc.)

FMOD_ERR_INVALID_FLOAT

Value passed in was a NaN, Inf or denormalized float.

FMOD_ERR_INVALID_HANDLE

An invalid object handle was used.

FMOD_ERR_INVALID_PARAM

An invalid parameter was passed to this function.

FMOD_ERR_INVALID_SPEAKER

An invalid speaker was passed to this function based on the current speaker mode.

FMOD_ERR_INVALID_VECTOR

The vectors passed in are not unit length, or perpendicular.

FMOD_ERR_IRX

PS2 only. fmodex.irx failed to initialize. This is most likely because you forgot to load it.

FMOD_ERR_MAXAUDIBLE

Reached maximum audible playback count for this sound's soundgroup.

FMOD_ERR_MEMORY

Not enough memory or resources.

FMOD_ERR_MEMORY_IOP

PS2 only. Not enough memory or resources on PlayStation 2 IOP ram.

FMOD_ERR_MEMORY_SRAM

Not enough memory or resources on console sound ram.

FMOD_ERR_MEMORY_CANTPOINT

Can't use FMOD_OPENMEMORY_POINT on non PCM source data, or non mp3/xma/adpcm data if FMOD_CREATECOMPRESSEDSAMPLE was used.

FMOD_ERR_NEEDS2D

Tried to call a command on a 3d sound when the command was meant for 2d sound.

FMOD_ERR_NEEDS3D

Tried to call a command on a 2d sound when the command was meant for 3d sound.

FMOD_ERR_NEEDSHARDWARE

Tried to use a feature that requires hardware support. (ie trying to play a VAG compressed sound in software on PS2).

FMOD_ERR_NEEDSSOFTWARE

Tried to use a feature that requires the software engine. Software engine has either been turned off, or command was executed on a hardware channel which does not support this feature.

FMOD_ERR_NET_CONNECT

Couldn't connect to the specified host.

FMOD_ERR_NET_SOCKET_ERROR

A socket error occurred. This is a catch-all for socket-related errors not listed elsewhere.

FMOD_ERR_NET_URL

The specified URL couldn't be resolved.

FMOD_ERR_NET_WOULD_BLOCK

Operation on a non-blocking socket could not complete immediately.

FMOD_ERR_NOTREADY

Operation could not be performed because specified sound is not ready.

FMOD_ERR_OUTPUT_ALLOCATED

Error initializing output device, but more specifically, the output device is already in use and cannot be reused.

FMOD_ERR_OUTPUT_CREATEBUFFER

Error creating hardware sound buffer.

FMOD_ERR_OUTPUT_DRIVERCALL

A call to a standard soundcard driver failed, which could possibly mean a bug in the driver or resources were missing or exhausted.

FMOD_ERR_OUTPUT_ENUMERATION

Error enumerating the available driver list. List may be inconsistent due to a recent device addition or removal.

FMOD_ERR_OUTPUT_FORMAT

Soundcard does not support the minimum features needed for this soundsystem (16bit stereo output).

FMOD_ERR_OUTPUT_INIT

Error initializing output device.

FMOD_ERR_OUTPUT_NOHARDWARE

FMOD_HARDWARE was specified but the sound card does not have the resources nescessary to play it.

FMOD_ERR_OUTPUT_NOSOFTWARE

Attempted to create a software sound but no software channels were specified in System::init.

FMOD_ERR_PAN

Panning only works with mono or stereo sound sources.

FMOD_ERR_PLUGIN

An unspecified error has been returned from a 3rd party plugin.

FMOD_ERR_PLUGIN_MISSING

A requested output, dsp unit type or codec was not available.

FMOD_ERR_PLUGIN_RESOURCE

A resource that the plugin requires cannot be found. (ie the DLS file for MIDI playback)

FMOD_ERR_PLUGIN_INSTANCES

The number of allowed instances of a plugin has been exceeded.

FMOD_ERR_RECORD

An error occured trying to initialize the recording device.

FMOD_ERR_REVERB_INSTANCE

Specified Instance in FMOD_REVERB_PROPERTIES couldn't be set. Most likely because another application has locked the EAX4 FX slot.

FMOD_ERR_SUBSOUNDS

The error occured because the sound referenced contains subsounds. (ie you cannot play the parent sound as a static sample, only its subsounds.)

FMOD_ERR_SUBSOUND_ALLOCATED

This subsound is already being used by another sound, you cannot have more than one parent to a sound. Null out the other parent's entry first.

FMOD_ERR_TAGNOTFOUND

The specified tag could not be found or there are no tags.

FMOD_ERR_TOOMANYCHANNELS

The sound created exceeds the allowable input channel count. This can be increased using the maxinputchannels parameter in System::setSoftwareFormat.

FMOD_ERR_UNIMPLEMENTED

Something in FMOD hasn't been implemented when it should be! contact support!

FMOD_ERR_UNINITIALIZED

This command failed because System::init or System::setDriver was not called.

FMOD_ERR_UNSUPPORTED

A command issued was not supported by this object. Possibly a plugin without certain callbacks specified.

FMOD_ERR_UPDATE

An error caused by System::update occured.

FMOD_ERR_VERSION

The version number of this file format is not supported.

FMOD_ERR_EVENT_FAILED

An Event failed to be retrieved, most likely due to 'just fail' being specified as the max playbacks behavior.

FMOD_ERR_EVENT_INTERNAL

An error occured that wasn't supposed to. See debug log for reason.

FMOD_ERR_EVENT_INFOONLY

Can't execute this command on an EVENT_INFOONLY event.

FMOD_ERR_EVENT_MAXSTREAMS

Event failed because 'Max streams' was hit when FMOD_INIT_FAIL_ON_MAXSTREAMS was specified.

FMOD_ERR_EVENT_MISMATCH

FSB mismatches the FEV it was compiled with or FEV was built for a different platform.

FMOD_ERR_EVENT_NAMECONFLICT

A category with the same name already exists.

FMOD_ERR_EVENT_NOTFOUND

The requested event, event group, event category or event property could not be found.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

FMOD_SOUNDGROUP_BEHAVIOR

These flags are used with SoundGroup::setMaxAudibleBehavior to determine what happens when more sounds?are played than are specified with SoundGroup::setMaxAudible.?

Enumeration

```
  typedef enum {
    FMOD_SOUNDGROUP_BEHAVIOR_FAIL,
    FMOD_SOUNDGROUP_BEHAVIOR_MUTE,
    FMOD_SOUNDGROUP_BEHAVIOR_STEALLOWEST,
    FMOD_SOUNDGROUP_BEHAVIOR_MAX
}   FMOD_SOUNDGROUP_BEHAVIOR
```

Values

FMOD_SOUNDGROUP_BEHAVIOR_FAIL

Any sound played that puts the sound count over the SoundGroup::setMaxAudible setting, will simply fail during System::playSound.

FMOD_SOUNDGROUP_BEHAVIOR_MUTE

Any sound played that puts the sound count over the SoundGroup::setMaxAudible setting, will be silent, then if another sound in the group stops the sound that was silent before becomes audible again.

FMOD_SOUNDGROUP_BEHAVIOR_STEALLOWEST

Any sound played that puts the sound count over the SoundGroup::setMaxAudible setting, will steal the quietest / least important sound playing in the group.

FMOD_SOUNDGROUP_BEHAVIOR_MAX

Maximum number of open state types.

Remarks

When using FMOD_SOUNDGROUP_BEHAVIOR_MUTE, SoundGroup::setMuteFadeSpeed can be used to stop a sudden transition. Instead, the time specified will be used to cross fade between the sounds that go silent and the ones that become audible.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii, Solaris

See Also

- SoundGroup::setMaxAudibleBehavior
- SoundGroup::getMaxAudibleBehavior
- SoundGroup::setMaxAudible
- SoundGroup::getMaxAudible
- SoundGroup::setMuteFadeSpeed
- SoundGroup::getMuteFadeSpeed

FMOD_SOUND_FORMAT

These definitions describe the native format of the hardware or software buffer that will be used.?

Enumeration

```
  typedef enum {
    FMOD_SOUND_FORMAT_NONE,
    FMOD_SOUND_FORMAT_PCM8,
    FMOD_SOUND_FORMAT_PCM16,
    FMOD_SOUND_FORMAT_PCM24,
    FMOD_SOUND_FORMAT_PCM32,
    FMOD_SOUND_FORMAT_PCMFLOAT,
    FMOD_SOUND_FORMAT_GCADPCM,
    FMOD_SOUND_FORMAT_IMAADPCM,
    FMOD_SOUND_FORMAT_VAG,
    FMOD_SOUND_FORMAT_XMA,
    FMOD_SOUND_FORMAT_MPEG,
    FMOD_SOUND_FORMAT_MAX
} FMOD_SOUND_FORMAT
```

Values

FMOD_SOUND_FORMAT_NONE

Unitialized / unknown.

FMOD_SOUND_FORMAT_PCM8

8bit integer PCM data.

FMOD_SOUND_FORMAT_PCM16

16bit integer PCM data.

FMOD_SOUND_FORMAT_PCM24

24bit integer PCM data.

FMOD_SOUND_FORMAT_PCM32

32bit integer PCM data.

FMOD_SOUND_FORMAT_PCMFLOAT

32bit floating point PCM data.

FMOD_SOUND_FORMAT_GCADPCM

Compressed GameCube DSP data.

FMOD_SOUND_FORMAT_IMAADPCM

Compressed IMA ADPCM / Xbox ADPCM data.

FMOD_SOUND_FORMAT_VAG

Compressed PlayStation 2 / PlayStation Portable ADPCM data.

FMOD_SOUND_FORMAT_XMA

Compressed Xbox360 data.

FMOD_SOUND_FORMAT_MPEG

Compressed MPEG layer 2 or 3 data.

FMOD_SOUND_FORMAT_MAX

Maximum number of sound formats supported.

Remarks

This is the format the native hardware or software buffer will be or is created in.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::createSound
- Sound::getFormat

FMOD_SOUND_TYPE

These definitions describe the type of song being played.?

Enumeration

```
  typedef enum {
    FMOD_SOUND_TYPE_UNKNOWN,
    FMOD_SOUND_TYPE_AAC,
    FMOD_SOUND_TYPE_AIFF,
    FMOD_SOUND_TYPE_ASF,
    FMOD_SOUND_TYPE_AT3,
    FMOD_SOUND_TYPE_CDDA,
    FMOD_SOUND_TYPE_DLS,
    FMOD_SOUND_TYPE_FLAC,
    FMOD_SOUND_TYPE_FSB,
    FMOD_SOUND_TYPE_GCADPCM,
    FMOD_SOUND_TYPE_IT,
    FMOD_SOUND_TYPE_MIDI,
    FMOD_SOUND_TYPE_MOD,
    FMOD_SOUND_TYPE_MPEG,
    FMOD_SOUND_TYPE_OGGVORBIS,
    FMOD_SOUND_TYPE_PLAYLIST,
    FMOD_SOUND_TYPE_RAW,
    FMOD_SOUND_TYPE_S3M,
    FMOD_SOUND_TYPE_SF2,
    FMOD_SOUND_TYPE_USER,
    FMOD_SOUND_TYPE_WAV,
    FMOD_SOUND_TYPE_XM,
    FMOD_SOUND_TYPE_XMA,
    FMOD_SOUND_TYPE_VAG,
    FMOD_SOUND_TYPE_MAX
} FMOD_SOUND_TYPE;
```

Values

FMOD_SOUND_TYPE_UNKNOWN

3rd party / unknown plugin format.

FMOD_SOUND_TYPE_AAC

AAC. Currently unsupported.

FMOD_SOUND_TYPE_AIFF

AIFF.

FMOD_SOUND_TYPE_ASF

Microsoft Advanced Systems Format (ie WMA/ASF/WMV).

FMOD_SOUND_TYPE_AT3

Sony ATRAC 3 format

FMOD_SOUND_TYPE_CDDA

Digital CD audio.

FMOD_SOUND_TYPE_DLS

Sound font / downloadable sound bank.

FMOD_SOUND_TYPE_FLAC

FLAC lossless codec.

FMOD_SOUND_TYPE_FSB

FMOD Sample Bank.

FMOD_SOUND_TYPE_GCADPCM

GameCube ADPCM

FMOD_SOUND_TYPE_IT

Impulse Tracker.

FMOD_SOUND_TYPE_MIDI

MIDI.

FMOD_SOUND_TYPE_MOD

Protracker / Fasttracker MOD.

FMOD_SOUND_TYPE_MPEG

MP2/MP3 MPEG.

FMOD_SOUND_TYPE_OGGVORBIS

Ogg vorbis.

FMOD_SOUND_TYPE_PLAYLIST

Information only from ASX/PLS/M3U/WAX playlists

FMOD_SOUND_TYPE_RAW

Raw PCM data.

FMOD_SOUND_TYPE_S3M

ScreamTracker 3.

FMOD_SOUND_TYPE_SF2

Sound font 2 format.

FMOD_SOUND_TYPE_USER

User created sound.

FMOD_SOUND_TYPE_WAV

Microsoft WAV.

FMOD_SOUND_TYPE_XM

FastTracker 2 XM.

FMOD_SOUND_TYPE_XMA

Xbox360 XMA

FMOD_SOUND_TYPE_VAG

PlayStation 2 / PlayStation Portable adpcm VAG format.

FMOD_SOUND_TYPE_MAX

Maximum number of sound types supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getFormat

FMOD_SPEAKER

These are speaker types defined for use with the Channel::setSpeakerLevels command.?It can also be used for speaker placement in the System::set3DSpeakerPosition command.?

Enumeration

```
  typedef enum {
    FMOD_SPEAKER_FRONT_LEFT,
    FMOD_SPEAKER_FRONT_RIGHT,
    FMOD_SPEAKER_FRONT_CENTER,
    FMOD_SPEAKER_LOW_FREQUENCY,
    FMOD_SPEAKER_BACK_LEFT,
    FMOD_SPEAKER_BACK_RIGHT,
    FMOD_SPEAKER_SIDE_LEFT,
    FMOD_SPEAKER_SIDE_RIGHT,
    FMOD_SPEAKER_MAX,
    FMOD_SPEAKER_MONO,
    FMOD_SPEAKER_NULL,
    FMOD_SPEAKER_SBL,
    FMOD_SPEAKER_SBR
}   FMOD_SPEAKER;
```

Values

FMOD_SPEAKER_FRONT_LEFT

FMOD_SPEAKER_FRONT_RIGHT

FMOD_SPEAKER_FRONT_CENTER

FMOD_SPEAKER_LOW_FREQUENCY

FMOD_SPEAKER_BACK_LEFT

FMOD_SPEAKER_BACK_RIGHT

FMOD_SPEAKER_SIDE_LEFT

FMOD_SPEAKER_SIDE_RIGHT

FMOD_SPEAKER_MAX

Maximum number of speaker types supported.

FMOD_SPEAKER_MONO

For use with FMOD_SPEAKERMODE_MONO and Channel::SetSpeakerLevels. Mapped to same value as FMOD_SPEAKER_FRONT_LEFT.

FMOD_SPEAKER_NULL

A non speaker. Use this to send.

FMOD_SPEAKER_SBL

For use with FMOD_SPEAKERMODE_7POINT1 on PS3 where the extra speakers are surround back inside of side speakers.

FMOD_SPEAKER_SBR

For use with FMOD_SPEAKERMODE_7POINT1 on PS3 where the extra speakers are surround back inside of side speakers.

Remarks

If you are using FMOD_SPEAKERMODE_RAW and speaker assignments are meaningless, just cast a raw integer value to this type.
For example (FMOD_SPEAKER)7 would use the 7th speaker (also the same as FMOD_SPEAKER_SIDE_RIGHT).
Values higher than this can be used if an output system has more than 8 speaker types / output channels. 15 is the current maximum.

NOTE: On Playstation 3 in 7.1, the extra 2 speakers are not side left/side right, they are 'surround back left'/'surround back right' which locate the speakers behind the listener instead of to the sides like on PC. FMOD_SPEAKER_SBL/ FMOD_SPEAKER_SBR are provided to make it clearer what speaker is being addressed on that platform.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_SPEAKERMODE
- Channel::setSpeakerLevels
- Channel::getSpeakerLevels
- System::set3DSpeakerPosition
- System::get3DSpeakerPosition

FMOD_SPEAKERMAPTYPE

When creating a multichannel sound, FMOD will pan them to their default speaker locations, for example a 6 channel sound will default to one channel per 5.1 output speaker.
?Another example is a stereo sound. It will default to left = front left, right = front right.
?
?This is for sounds that are not 'default'. For example you might have a sound that is 6 channels but actually made up of 3 stereo pairs, that should all be located in front left, front right only.?

Enumeration

```
  typedef enum {
    FMOD_SPEAKERMAPTYPE_DEFAULT,
    FMOD_SPEAKERMAPTYPE_ALLMONO,
    FMOD_SPEAKERMAPTYPE_ALLSTEREO
}   FMOD_SPEAKERMAPTYPE;
```

Values

FMOD_SPEAKERMAPTYPE_DEFAULT

This is the default, and just means FMOD decides which speakers it puts the source channels.

FMOD_SPEAKERMAPTYPE_ALLMONO

This means the sound is made up of all mono sounds. All voices will be panned to the front center by default in this case.

FMOD_SPEAKERMAPTYPE_ALLSTEREO

This means the sound is made up of all stereo sounds. All voices will be panned to front left and front right alternating every second channel.

Remarks

For full flexibility of speaker assignments, use Channel::setSpeakerLevels. This functionality is cheaper, uses less memory and easier to use.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- FMOD_CREATESOUNDEXINFO

- Channel::setSpeakerLevels

FMOD_SPEAKERMODE

These are speaker types defined for use with the System::setSpeakerMode or System::getSpeakerMode command.?

Enumeration

```
  typedef enum {
   FMOD_SPEAKERMODE_RAW,
   FMOD_SPEAKERMODE_MONO,
   FMOD_SPEAKERMODE_STEREO,
   FMOD_SPEAKERMODE_QUAD,
   FMOD_SPEAKERMODE_SURROUND,
   FMOD_SPEAKERMODE_5POINT1,
   FMOD_SPEAKERMODE_7POINT1,
   FMOD_SPEAKERMODE_PROLOGIC,
   FMOD_SPEAKERMODE_MAX
} FMOD_SPEAKERMODE;
```

Values

FMOD_SPEAKERMODE_RAW

There is no specific speakermode. Sound channels are mapped in order of input to output. Use System::setSoftwareFormat to specify speaker count. See remarks for more information.

FMOD_SPEAKERMODE_MONO

The speakers are monaural.

FMOD_SPEAKERMODE_STEREO

The speakers are stereo (DEFAULT).

FMOD_SPEAKERMODE_QUAD

4 speaker setup. This includes front left, front right, rear left, rear right.

FMOD_SPEAKERMODE_SURROUND

5 speaker setup. This includes front left, front right, center, rear left, rear right.

FMOD_SPEAKERMODE_5POINT1

5.1 speaker setup. This includes front left, front right, center, rear left, rear right and a subwoofer.

FMOD_SPEAKERMODE_7POINT1

7.1 speaker setup. This includes front left, front right, center, rear left, rear right, side left, side right and a subwoofer.

FMOD_SPEAKERMODE_PROLOGIC

Stereo output, but data is encoded in a way that is picked up by a Prologic/Prologic2 decoder and split into a 5.1 speaker setup.

FMOD_SPEAKERMODE_MAX

Maximum number of speaker modes supported.

Remarks

These are important notes on speaker modes in regards to sounds created with FMOD_SOFTWARE.
Note below the phrase 'sound channels' is used. These are the subchannels inside a sound, they are not related and have nothing to do with the FMOD class "Channel".
For example a mono sound has 1 sound channel, a stereo sound has 2 sound channels, and an AC3 or 6 channel wav file have 6 "sound channels".

FMOD_SPEAKERMODE_RAW

This mode is for output devices that are not specifically mono/stereo/quad/surround/5.1 or 7.1, but are multichannel.
Use System::setSoftwareFormat to specify the number of speakers you want to address, otherwise it will default to 2 (stereo).
Sound channels map to speakers sequentially, so a mono sound maps to output speaker 0, stereo sound maps to output speaker 0 ?
The user assumes knowledge of the speaker order. FMOD_SPEAKER enumerations may not apply, so raw channel indices should be used.
Multichannel sounds map input channels to output channels 1:1.
Channel::setPan and Channel::setSpeakerMix do not work.
Speaker levels must be manually set with Channel::setSpeakerLevels.

FMOD_SPEAKERMODE_MONO

This mode is for a 1 speaker arrangement.
Panning does not work in this speaker mode.
Mono, stereo and multichannel sounds have each sound channel played on the one speaker unity.
Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
Channel::setSpeakerMix does not work.

FMOD_SPEAKERMODE_STEREO

This mode is for 2 speaker arrangements that have a left and right speaker.

- Mono sounds default to an even distribution between left and right. They can be panned with Channel::setPan.
- Stereo sounds default to the middle, or full left in the left speaker and full right in the right speaker.
- They can be cross faded with Channel::setPan.
- Multichannel sounds have each sound channel played on each speaker at unity.
- Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
- Channel::setSpeakerMix works but only front left and right parameters are used, the rest are ignored.

FMOD_SPEAKERMODE_QUAD

This mode is for 4 speaker arrangements that have a front left, front right, rear left and a rear right speaker.

- Mono sounds default to an even distribution between front left and front right. They can be panned with Channel::setPan.
- Stereo sounds default to the left sound channel played on the front left, and the right sound channel played on the front right.

- They can be cross faded with Channel::setPan.
- Multichannel sounds default to all of their sound channels being played on each speaker in order of input.
- Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
- Channel::setSpeakerMix works but side left, side right, center and lfe are ignored.

FMOD_SPEAKERMODE_SURROUND

This mode is for 5 speaker arrangements that have a left/right/center/rear left/rear right.
- Mono sounds default to the center speaker. They can be panned with Channel::setPan.
- Stereo sounds default to the left sound channel played on the front left, and the right sound channel played on the front right.
- They can be cross faded with Channel::setPan.
- Multichannel sounds default to all of their sound channels being played on each speaker in order of input.
- Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
- Channel::setSpeakerMix works but side left / side right are ignored.

FMOD_SPEAKERMODE_5POINT1

This mode is for 5.1 speaker arrangements that have a left/right/center/rear left/rear right and a subwoofer speaker.
- Mono sounds default to the center speaker. They can be panned with Channel::setPan.
- Stereo sounds default to the left sound channel played on the front left, and the right sound channel played on the front right.
- They can be cross faded with Channel::setPan.
- Multichannel sounds default to all of their sound channels being played on each speaker in order of input.
- Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
- Channel::setSpeakerMix works but side left / side right are ignored.

FMOD_SPEAKERMODE_7POINT1

This mode is for 7.1 speaker arrangements that have a left/right/center/rear left/rear right/side left/side right and a subwoofer speaker.
- Mono sounds default to the center speaker. They can be panned with Channel::setPan.
- Stereo sounds default to the left sound channel played on the front left, and the right sound channel played on the front right.
- They can be cross faded with Channel::setPan.
- Multichannel sounds default to all of their sound channels being played on each speaker in order of input.
- Mix behavior for multichannel sounds can be set with Channel::setSpeakerLevels.
- Channel::setSpeakerMix works and every parameter is used to set the balance of a sound in any speaker.

FMOD_SPEAKERMODE_PROLOGIC

This mode is for mono, stereo, 5.1 and 7.1 speaker arrangements, as it is backwards and forwards compatible with stereo, but to get a surround effect a Dolby Prologic or Prologic 2 hardware decoder / amplifier is needed.
Pan behavior is the same as FMOD_SPEAKERMODE_5POINT1.

If this function is called the numoutputchannels setting in System::setSoftwareFormat is overwritten.

For 3D sounds, panning is determined at runtime by the 3D subsystem based on the speaker mode to determine which speaker the sound should be placed in.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setSpeakerMode
- System::getSpeakerMode
- System::getDriverCaps
- System::setSoftwareFormat
- Channel::setSpeakerLevels

FMOD_SYSTEM_CALLBACKTYPE

These callback types are used with System::setCallback.?

Enumeration

```
  typedef enum {
    FMOD_SYSTEM_CALLBACKTYPE_DEVICELISTCHANGED,
    FMOD_SYSTEM_CALLBACKTYPE_MEMORYALLOCATIONFAILED,
    FMOD_SYSTEM_CALLBACKTYPE_MAX
} FMOD_SYSTEM_CALLBACKTYPE;
```

Values

FMOD_SYSTEM_CALLBACKTYPE_DEVICELISTCHANGED

Called from System::update when the enumerated list of devices has changed.

FMOD_SYSTEM_CALLBACKTYPE_MEMORYALLOCATIONFAILED

Called directly when a memory allocation fails somewhere in FMOD.

FMOD_SYSTEM_CALLBACKTYPE_MAX

Maximum number of callback types supported.

Remarks

Each callback has commanddata parameters passed as int unique to the type of callback.
See reference to FMOD_SYSTEM_CALLBACK to determine what they might mean for each type of callback.

Note! Using FMOD_SYSTEM_CALLBACKTYPE_MEMORYALLOCATIONFAILED will override any other FMOD::System object registered for this callback.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- System::setCallback
- FMOD_SYSTEM_CALLBACK

- System::update

Version 4.12.03 Built on Feb 18, 2008

FMOD_TAGDATATYPE

List of data types that can be returned by Sound::getTag?

Enumeration

```
  typedef enum {
    FMOD_TAGDATATYPE_BINARY,
    FMOD_TAGDATATYPE_INT,
    FMOD_TAGDATATYPE_FLOAT,
    FMOD_TAGDATATYPE_STRING,
    FMOD_TAGDATATYPE_STRING_UTF16,
    FMOD_TAGDATATYPE_STRING_UTF16BE,
    FMOD_TAGDATATYPE_STRING_UTF8,
    FMOD_TAGDATATYPE_CDTOC,
    FMOD_TAGDATATYPE_MAX
} FMOD_TAGDATATYPE;
```

Values

FMOD_TAGDATATYPE_BINARY

FMOD_TAGDATATYPE_INT

FMOD_TAGDATATYPE_FLOAT

FMOD_TAGDATATYPE_STRING

FMOD_TAGDATATYPE_STRING_UTF16

FMOD_TAGDATATYPE_STRING_UTF16BE

FMOD_TAGDATATYPE_STRING_UTF8

FMOD_TAGDATATYPE_CDTOC

FMOD_TAGDATATYPE_MAX

Maximum number of tag datatypes supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getTag

FMOD_TAGTYPE

List of tag types that could be stored within a sound. These include id3 tags, metadata from netstreams and vorbis/asf data.?

Enumeration

```
  typedef enum {
    FMOD_TAGTYPE_UNKNOWN,
    FMOD_TAGTYPE_ID3V1,
    FMOD_TAGTYPE_ID3V2,
    FMOD_TAGTYPE_VORBISCOMMENT,
    FMOD_TAGTYPE_SHOUTCAST,
    FMOD_TAGTYPE_ICECAST,
    FMOD_TAGTYPE_ASF,
    FMOD_TAGTYPE_MIDI,
    FMOD_TAGTYPE_PLAYLIST,
    FMOD_TAGTYPE_FMOD,
    FMOD_TAGTYPE_USER,
    FMOD_TAGTYPE_MAX
}   FMOD_TAGTYPE;
```

Values

FMOD_TAGTYPE_UNKNOWN

FMOD_TAGTYPE_ID3V1

FMOD_TAGTYPE_ID3V2

FMOD_TAGTYPE_VORBISCOMMENT

FMOD_TAGTYPE_SHOUTCAST

FMOD_TAGTYPE_ICECAST

FMOD_TAGTYPE_ASF

FMOD_TAGTYPE_MIDI

FMOD_TAGTYPE_PLAYLIST

FMOD_TAGTYPE_FMOD

FMOD_TAGTYPE_USER

FMOD_TAGTYPE_MAX

Maximum number of tag types supported.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Solaris

See Also

- Sound::getTag

Firelight Technologies FMOD Ex

C++ Reference

C++ interfaces

EventSystem Interface

EventSystem::createReverb

Creates a 'virtual reverb' object. This object reacts to 3d location and morphs the reverb environment based on how close it is to the reverb object's center.
?Multiple reverb objects can be created to achieve a multi-reverb environment.?

Syntax

```
  FMOD_RESULT EventSystem::createReverb(
  EventReverb **  reverb
);
```

Parameters

reverb

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Use EventSystem::setReverbAmbientProperties to set a 'background' default reverb environment. This is a reverb that will be morphed to if the listener is not within any virtual reverb zones.
By default the ambient reverb is set to 'off'.
Creating multiple reverb objects does not impact performance. These are 'virtual reverbs'. There will still be only 1 physical reverb DSP running that just morphs between the different virtual reverbs.
EventSystem::setReverbProperties can still be used in conjunction with the 3d based virtual reverb system. This allows 2d sounds to have reverb. If this call is used at the same time virtual reverb objects are active, 2 physical reverb dsps will be used, incurring a small memory and cpu hit.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::setReverbAmbientProperties
- EventSystem::getReverbAmbientProperties
- EventSystem::setReverbProperties
- EventSystem::getReverbProperties
- EventReverb::release

EventSystem::get3DListenerAttributes

This retrieves the position, velocity and orientation of the specified 3D sound listener.?

Syntax

```
 FMOD_RESULT EventSystem::get3DListenerAttributes(
 int  listener,
  FMOD_VECTOR *  pos,
  FMOD_VECTOR *  vel,
  FMOD_VECTOR *  forward,
  FMOD_VECTOR * up
);
```

Parameters

listener

Listener ID in a multi-listener environment. Specify 0 if there is only 1 listener.

pos

Address of a variable that receives the position of the listener in world space, measured in distance units. Optional. Specify 0 to ignore.

vel

Address of a variable that receives the velocity of the listener measured in distance units **per second**. Optional. Specify 0 to ignore.

forward

Address of a variable that receives the forwards orientation of the listener. Optional. Specify 0 to ignore.

up

Address of a variable that receives the upwards orientation of the listener. Optional. Specify 0 to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable,

PlayStation 3, Wii

See Also

- EventSystem::set3DListenerAttributes
- FMOD_VECTOR

EventSystem::get3DNumListeners

Retrieves the number of 3D listeners.?

Syntax

```
  FMOD_RESULT EventSystem::get3DNumListeners(
  int * numlisteners
);
```

Parameters

numlisteners

Address of a variable that receives the current number of 3D listeners in the 3D scene.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::set3DNumListeners

EventSystem::getCategory

Retrieve an event category object by name.?

Syntax

```
  FMOD_RESULT EventSystem::getCategory(
  const char *  name,
  EventCategory **  category
);
```

Parameters

name

The name of an event category within this event system. Specify "master" to retrieve the master event category.

category

Address of a variable to receive the selected event category within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Sub-categories can be retrieved by specifying their full path e.g. "vehicles/cars/racers".

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategoryByIndex
- EventSystem::getNumCategories

EventSystem::getCategoryByIndex

Retrieve an event category object by index.?

Syntax

```
  FMOD_RESULT EventSystem::getCategoryByIndex(
  int index,
  EventCategory ** category
);
```

Parameters

index

The index of an event category within this event system object. Indices are 0 based. Specify -1 to retrieve the master event category.

category

Address of a variable to receive the selected event category within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategory
- EventSystem::getNumCategories

EventSystem::getEvent

Retrieve an event object by name.?

Syntax

```
  FMOD_RESULT EventSystem::getEvent(
  const char *  name,
   FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

name

The name of an event within this event system. Note: name must include full path including project name and any event group names e.g. "myproject/group1/group2/myevent"

mode

event

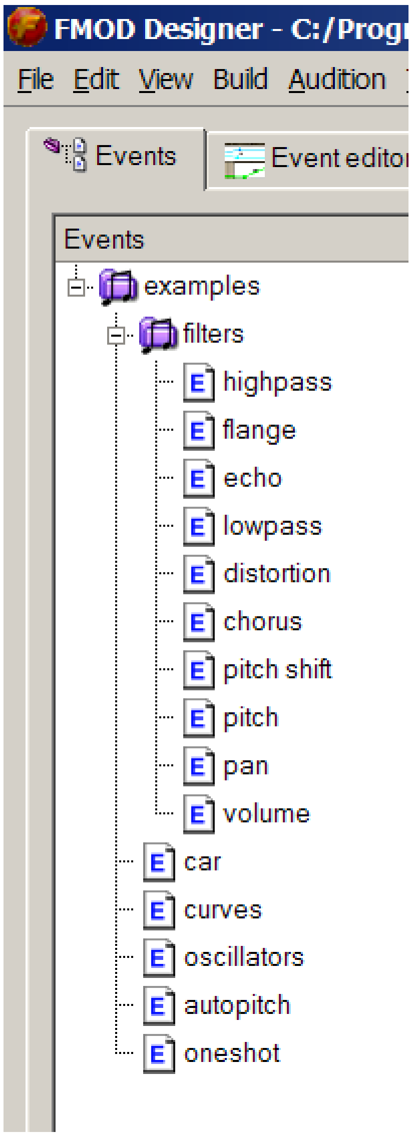
Address of a variable to receive the selected event within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with "**echo**" as the name parameter.

If the programmer does not know which events are available, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.

Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an invent handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getGroup
- FMOD_EVENT_MODE

EventSystem::getEventBySystemID

Retrieve an event object by system wide unique identifier. All loaded events can be enumerated with this function and EventSystem::getNumEvents.?

Syntax

```
FMOD_RESULT EventSystem::getEventBySystemID(
  unsigned int systemid,
  FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

systemid

mode

event

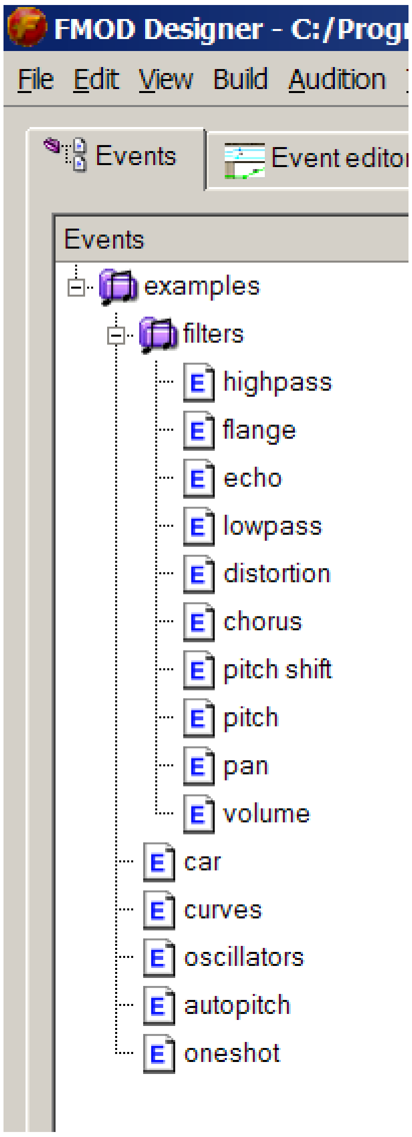
Address of a variable to receive the selected event within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with "**echo**" as the name parameter.

If the programmer does not know which events are available, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.

Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an invent handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getNumEvents
- FMOD_EVENT_MODE

EventSystem::getGroup

Retrieves an event group object by name.?

Syntax

```
  FMOD_RESULT EventSystem::getGroup(
  const char *  name,
   bool  cacheevents,
  EventGroup **  group
);
```

Parameters

name

The name of an event group that belongs to this event system. Note: name must include full path including project name and any event group names e.g. "myproject/group1/group2"

cacheevents

If cacheevents is true then all event instances within this event group will be pre-allocated so that there are no memory allocs when getEvent is called.

group

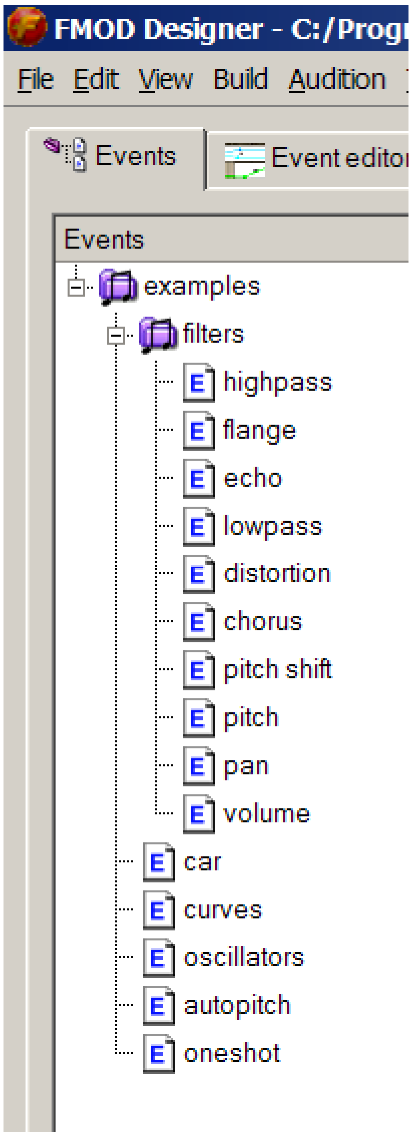
Address of a variable to receive the selected event group within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event group?
An event group is a "folder" that stores events or sub-folders. With these folders a hierarchical tree can be built to store events in a more logical manner.



In this case we are retrieving an event group from another **event group**, so if this event group object was "**examples**" we could then get the event group "**filters**" with "**filters**" as the name parameter.
In this example "**filters**" is the only sub-group below "**examples**" so no other sub-groups are available here.

If the programmer does not know which sub-groups are available or which sub-group index matches which sub-group name, the sound designer tool can output a programmer report that lists the group's sub-groups with the appropriate names and indices listed alongside them.
The only benefit of retrieving an object by index is that it is slightly faster to do so than to retrieve it by name.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventSystem::getInfo

Retrieves information about the event system.?

Syntax

```
  FMOD_RESULT EventSystem::getInfo(
    FMOD_EVENT_SYSTEMINFO *  info
);
```

Parameters

info

Address of an FMOD_EVENT_SYSTEMINFO structure to receive event system information.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_EVENT_SYSTEMINFO

EventSystem::getNumCategories

Retrieve the number of categories for the event system.?

Syntax

```
  FMOD_RESULT EventSystem::getNumCategories(
  int * numcategories
);
```

Parameters

numcategories

Address of a variable to receive the number of categories for this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategoryByIndex
- EventSystem::getCategory

EventSystem::getNumEvents

Gets the total number of unique events loaded into the system at once, including those from different EventProjects. ?Mainly used for enumeration of all events loaded into the system. This can be done in conjunction with EventSystem::getEventBySystemID.?

Syntax

```
  FMOD_RESULT EventSystem::getNumEvents(
  int *  numevents
);
```

Parameters

numevents

Pointer to a integer to retrieve the current number of unique events.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getEventBySystemID

EventSystem::getNumProjects

Retrieve the number of event projects within the top level event system.?

Syntax

```
  FMOD_RESULT EventSystem::getNumProjects(
  int *  numprojects
);
```

Parameters

numprojects

Address of a variable to receive the number of event projects within the top level event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getProjectByIndex

EventSystem::getNumReverbPresets

Retrieve the number of reverb presets defined by the sound designer.?

Syntax

```
  FMOD_RESULT EventSystem::getNumReverbPresets(
  int *  numpresets
);
```

Parameters

numpresets

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use this in conjunction with EventSystem::getReverbPresetByIndex to enumerate all sound designer specified reverb presets.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getReverbPresetByIndex
- EventSystem::getReverbPreset

EventSystem::getProject

Retrieve an event project object by name.?

Syntax

```
  FMOD_RESULT EventSystem::getProject(
  const char *  name,
  EventProject **  project
);
```

Parameters

name

The name of an event project within this event system.

project

Address of a variable to receive the selected event project within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getProjectByIndex
- EventSystem::load

EventSystem::getProjectByIndex

Retrieve an event project object by index.?

Syntax

```
  FMOD_RESULT EventSystem::getProjectByIndex(
  int index,
  EventProject ** project
);
```

Parameters

index

The index of an event project within this event system object. Indices are 0 based.

project

Address of a variable to receive the selected event project within this event system.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getProject
- EventSystem::getNumProjects
- EventSystem::load

EventSystem::getReverbAmbientProperties

Retrieves the default reverb envrionment for the virtual reverb system.?

Syntax

```
  FMOD_RESULT EventSystem::getReverbAmbientProperties(
    FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a pointer to a FMOD_REVERB_PROPERTIES to receieve the settings for the current ambient reverb setting.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default the ambient reverb is set to 'off'. This is the same as FMOD_REVERB_PRESET_OFF.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable

See Also

- FMOD_REVERB_PROPERTIES
- EventSystem::setReverbAmbientProperties
- EventSystem::createReverb

EventSystem::getReverbPreset

Retrieves a reverb property structure containing a reverb preset created by the sound designer, by name.?

Syntax

```
  FMOD_RESULT EventSystem::getReverbPreset(
  const char *  name,
   FMOD_REVERB_PROPERTIES  *  prop,
  int * index
);
```

Parameters

name

The name of an event reverb within this event system.

prop

Address of a variable to receive a FMOD_REVERB_PROPERTIES containing the desired reverb preset.

index

Address of a variable to receive the index of the preset in the reverb list. Optional. This index is the index used in EventSystem::getReverbPresetByIndex. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use the retrieved FMOD_REVERB_PROPERTIES structure to pass to EventSystem::setReverbProperties, EventSystem::setReverbAmbientProperties or EventReverb::setProperties.
Reverb presets can also be retrieved with EventSystem::getReverbPresetByIndex.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable

See Also

- FMOD_REVERB_PROPERTIES
- EventSystem::getReverbPresetByIndex

- EventSystem::setReverbProperties
- EventSystem::setReverbAmbientProperties
- EventReverb::setProperties
- EventReverb::release

EventSystem::getReverbPresetByIndex

Retrieves a reverb property structure containing a reverb preset created by the sound designer, by index instead of name.?

Syntax

```
FMOD_RESULT EventSystem::getReverbPresetByIndex(
  const int index,
   FMOD_REVERB_PROPERTIES *  prop,
  char **  name
);
```

Parameters

index

The index of an event reverb within this event system object. Indices are 0 based.

prop

Address of a variable to receive a FMOD_REVERB_PROPERTIES containing the desired reverb preset.

name

Address of a variable to receive a pointer to the name of the preset. Optional. This index is the name used in EventSystem::getReverbPreset. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
Use the retrieved FMOD_REVERB_PROPERTIES structure to pass to EventSystem::setReverbProperties, EventSystem::setReverbAmbientProperties or EventReverb::setProperties.

Remarks

All reverbs can be enumerated by using this function in conjunction with EventSystem::getNumReverbPresets.
Reverb presets can also be retrieved with EventSystem::getReverbPreset.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable

See Also

- FMOD_REVERB_PROPERTIES
- EventSystem::getReverbPreset
- EventSystem::getNumReverbPresets
- EventReverb::release

EventSystem::getReverbProperties

Retrieves the current reverb environment.?

Syntax

```
  FMOD_RESULT EventSystem::getReverbProperties(
    FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a variable that receives the current reverb environment description.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::setReverbProperties
- Event::setReverbProperties
- Event::getReverbProperties

EventSystem::getSystemObject

Retrieve the event system's internal FMOD::System object for the low level FMOD Ex API.?

Syntax

```
  FMOD_RESULT EventSystem::getSystemObject(
    FMOD::System ** system
);
```

Parameters

system

Address of a pointer to receive the FMOD::System pointer.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note! This should generally not be used unless you are trying to add features that the sound designer cannot provide! The aim of this API is to give the sound designer the control over the sound behaviour. If there are things missing from the EventSystem API that could be included contact FMOD support at support@fmod.org and it will be considered for addition.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

EventSystem::getUserData

Retrieves the user value that that was set by calling the EventSystem::setUserData function.?

Syntax

```
  FMOD_RESULT EventSystem::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventSystem::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::setUserData

EventSystem::getVersion

Returns the current version of the event system being used.?

Syntax

```
  FMOD_RESULT EventSystem::getVersion(
  unsigned int *  version
);
```

Parameters

version

Address of a variable that receives the current event system version.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The version is a 32bit hexadecimal value formated as 16:8:8, with the upper 16bits being the major version, the middle 8bits being the minor version and the bottom 8bits being the development version. For example a value of 00010106h is equal to 1.01.06.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::init

EventSystem::init

Initializes the event system object, FMOD system object and the sound device. This has to be called at the start of the user's program.
?**You must create an event system object with EventSystem_Create.**?

Syntax

```
  FMOD_RESULT EventSystem::init(
  int  maxchannels,
   FMOD_INITFLAGS   flags,
   void *  extradriverdata,
   FMOD_EVENT_INITFLAGS   eventflags
);
```

Parameters

maxchannels

The maximum number of channels to be used in FMOD. They are also called 'virtual channels' as you can play as many of these as you want, even if you only have a small number of hardware or software voices. See remarks for more.

flags

See FMOD_INITFLAGS. This can be a selection of flags bitwise OR'ed together to change the behaviour of FMOD at initialization time.

extradriverdata

Driver specific data that can be passed to the output plugin. For example the filename for the wav writer plugin. See FMOD_OUTPUTTYPE for what each output mode might take here. Optional. Specify 0 to ignore.

eventflags

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

See FMOD Ex documentation for details on FMOD_INITFLAGS etc.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem_Create

EventSystem::load

Loads an event file (.fev).?

Syntax

```
  FMOD_RESULT EventSystem::load(
  const char *  name_or_data,
   FMOD_EVENT_LOADINFO *  loadinfo,
  EventProject **  project
);
```

Parameters

name_or_data

Filename of the event file to be loaded or pointer to memory block if FMOD_EVENT_LOADINFO is provided and has a non-zero "loadfrommemory_length" field.

loadinfo

Pointer to an FMOD_EVENT_LOADINFO structure which lets the user provide extended information about loading the file. Optional. Specify 0 or NULL to ignore.

project

Address of a variable to receive the specified project object. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Loading the event data file will not open any FSB files or allocate any memory for events. This is done with EventGroup::loadEventData (to load FSB data) and EventProject::getGroup / EventProject::getGroupByIndex / EventGroup::getGroup / EventGroup::getGroupByIndex to allocate memory for the event instances so that they can be played.
To load a .fev file from memory, pass a pointer to the memory block in "name_or_data" and provide an FMOD_EVENT_LOADINFO structure with the "loadfrommemory_length" field set to the length of the memory block. The memory block can be freed immediately after this function returns.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::setMediaPath
- EventGroup::loadEventData
- EventSystem::getProject
- EventSystem::getProjectByIndex
- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventSystem::registerMemoryFSB

For users that want to pre-load static sample FSB files into memory, this function can be used to stop FMOD from loading any FSB with the same filename from disk, and it will instead point to this memory instead.?

Syntax

```
  FMOD_RESULT EventSystem::registerMemoryFSB(
  const char * filename,
   void * fsbdata,
  unsigned int fsbdatalen,
   bool loadintorsx
);
```

Parameters

filename

File name that FMOD event system would use to load. FMOD will compare this string against media path + fsb filename or just fsb filename by itself.

fsbdata

Pointer to in memory FSB file.

fsbdatalen

Length of in memory FSB file data in bytes.

loadintorsx

PS3 only. When set to true, data pointed to by 'fsbdata' will be copied to the RSX memory pool specified at System::init.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When a bank is opened as streamed, these in memory FSB files are not referenced. If 'loadintorsx' is set to true, data pointed to by 'fsbdata' will be copied to the RSX memory pool specified at System::init. It is safe to free 'fsbdata' when this is the case. NOTE: There is a performance penalty when using RSX memory. Please refer to the PS3 section of the FMOD documentation for more details.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::unregisterMemoryFSB

EventSystem::release

Closes and frees an event system object.?

Syntax

```
FMOD_RESULT EventSystem::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This will free the event system object and everything created under it.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem_Create
- EventSystem::init

EventSystem::set3DListenerAttributes

This updates the position, velocity and orientation of the specified 3D sound listener.?

Syntax

```
  FMOD_RESULT EventSystem::set3DListenerAttributes(
  int  listener,
  const FMOD_VECTOR *  pos,
  const FMOD_VECTOR *  vel,
  const FMOD_VECTOR *  forward,
  const FMOD_VECTOR * up
);
```

Parameters

listener

Listener ID in a multi-listener environment. Specify 0 if there is only 1 listener.

pos

Address of a variable that receives the position of the listener in world space, measured in distance units. You can specify 0 or NULL to not update the position.

vel

Address of a variable that receives the velocity of the listener measured in distance units **per second**. You can specify 0 or NULL to not update the velocity of the listener.

forward

Address of a variable that receives the forwards orientation of the listener. This vector must be of unit length and perpendicular to the up vector. You can specify 0 or NULL to not update the forwards orientation of the listener.

up

Address of a variable that receives the upwards orientation of the listener. This vector must be of unit length and perpendicular to the forwards vector. You can specify 0 or NULL to not update the upwards orientation of the listener.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

By default, FMOD uses a left-handed co-ordinate system. This means +X is right, +Y is up, and +Z is forwards.
To change this to a right-handed coordinate system, use FMOD_INIT_3D_RIGHTHANDED. This means +X is left, +Y is up, and +Z is forwards.

To map to another coordinate system, flip/negate and exchange these values.

Orientation vectors are expected to be of UNIT length. This means the magnitude of the vector should be 1.0.

A 'distance unit' is specified by the sound designer in the FMOD Designer tool. By default this is set to meters which is a distance scale of 1.0.

Always remember to use **units per second**, *not* units per frame as this is a common mistake and will make the doppler effect sound wrong.
For example, Do not just use (pos - lastpos) from the last frame's data for velocity, as this is not correct. You need to time compensate it so it is given in units per **second**.
You could alter your pos - lastpos calculation to something like this.

```
vel = (pos-lastpos) / time_taken_since_last_frame_in_seconds.
```

I.e. at 60fps the formula would look like this vel = (pos-lastpos) / 0.0166667.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::get3DListenerAttributes
- FMOD_INITFLAGS
- FMOD_VECTOR

EventSystem::set3DNumListeners

Sets the number of 3D 'listeners' in the 3D sound scene. This function is useful mainly for split-screen game purposes.?

Syntax

```
FMOD_RESULT EventSystem::set3DNumListeners(
  int  numlisteners
);
```

Parameters

numlisteners

Number of listeners in the scene. Valid values are from 1-4 inclusive. Default = 1.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If the number of listeners is set to more than 1, then panning and doppler are turned off. All sound effects will be mono.
FMOD uses a 'closest sound to the listener' method to determine what should be heard in this case.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::get3DNumListeners
- EventSystem::set3DListenerAttributes

EventSystem::setMediaPath

Specify a base search path for media files so they can be placed somewhere other than the directory of the main executable.?

Syntax

```
  FMOD_RESULT EventSystem::setMediaPath(
  const char *  path
);
```

Parameters

path

A character string containing a correctly formatted path to load media files from. NOTE: Must contain a trailing slash/backslash if filesystem requires it!

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

FMOD used to add a slash/backslash seperator between the path provided here and the files that it needed to load. This caused inconsistent and inflexible behaviour so this function has been changed to expect a path that already contains a trailing slash/backslash. FMOD will no longer add any slash/backslash seperators between the path specified here and the files that it needs to load. This allows the user to provide the correct seperator for the filesystem in use - which may actually be the user's own filesystem which may or may not use a seperator at all.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::load

EventSystem::setPluginPath

Specify a base search path for plugins so they can be placed somewhere other than the directory of the main executable.?

Syntax

```
  FMOD_RESULT EventSystem::setPluginPath(
  const char *  path
);
```

Parameters

path

A character string containing a correctly formatted path to load plugins from. You can specify 0 or NULL to tell FMOD not to load any plugins.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::load

EventSystem::setReverbAmbientProperties

Sets a 'background' default reverb environment for the virtual reverb system. This is a reverb preset that will be morphed to if the listener is not within any virtual reverb zones.
?By default the ambient reverb is set to 'off'.?

Syntax

```
  FMOD_RESULT EventSystem::setReverbAmbientProperties(
    FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of a FMOD_REVERB_PROPERTIES structure containing the settings for the desired ambient reverb setting.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

There is one reverb DSP dedicated to providing a 3D reverb effect. This DSP's properties are a weighted sum of all the contributing virtual reverbs.
The default 3d reverb properties specify the reverb properties in the 3D volumes which has no virtual reverbs defined.

EventSystem::getReverbPreset and EventSystem::getReverbPresetByIndex can be used to retrieve sound designer defined presets, or it can be set programatically.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_REVERB_PROPERTIES
- EventSystem::getReverbAmbientProperties
- EventSystem::createReverb

- EventSystem::getReverbPreset
- EventSystem::getReverbPresetByIndex

EventSystem::setReverbProperties

Sets parameters for the global reverb environment.
?Reverb parameters can be set manually, or automatically using the pre-defined presets given in the fmod.h header.?

Syntax

```
  FMOD_RESULT EventSystem::setReverbProperties(
  const FMOD_REVERB_PROPERTIES *  prop
);
```

Parameters

prop

Address of an FMOD_REVERB_PROPERTIES structure which defines the attributes for the reverb.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

EventSystem::getReverbPreset and EventSystem::getReverbPresetByIndex can be used to retrieve sound designer defined presets, or it can be set programatically.
With FMOD_HARDWARE on Windows using EAX, the reverb will only work on FMOD_3D based sounds. FMOD_SOFTWARE does not have this problem and works on FMOD_2D and FMOD_3D based sounds.

On PlayStation 2, the reverb is limited to only a few reverb types that are not configurable. Use the FMOD_PRESET_PS2_xxx presets.
On Xbox, it is possible to apply reverb to FMOD_2D and FMOD_HARDWARE based voices using this function. By default reverb is turned off for FMOD_2D hardware based voices.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_REVERB_PROPERTIES
- EventSystem::getReverbProperties

- Event::setReverbProperties
- Event::getReverbProperties
- EventSystem::getReverbPreset
- EventSystem::getReverbPresetByIndex

EventSystem::setUserData

Sets a user value that the EventSystem object will store internally. Can be retrieved with EventSystem::getUserData.?

Syntax

```
  FMOD_RESULT EventSystem::setUserData(
   void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventSystem object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventSystem::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getUserData

EventSystem::unload

Unloads all loaded event projects?

Syntax

```
FMOD_RESULT EventSystem::unload();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::load

EventSystem::unregisterMemoryFSB

De-register an in memory FSB from the system.?

Syntax

```
  FMOD_RESULT EventSystem::unregisterMemoryFSB(
  const char *  filename
);
```

Parameters

filename

FSB filename to unregister.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Any references to this filename by the EventSystem internally will be loaded from disk as normal if this FSB is unregistered as an in memory FSB. If 'loadintorsx' was used in EventSystem::registerMemoryFSB, this data is freed from the RSX pool.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::registerMemoryFSB

EventSystem::update

Updates the event system. This should be called once per 'game' tick, or once per frame in your application.?

Syntax

```
FMOD_RESULT EventSystem::update();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::init

EventProject Interface

EventProject::getEvent

Retrieve a an event object by name.?

Syntax

```
  FMOD_RESULT EventProject::getEvent(
  const char *  name,
   FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

name

The name of an event within this event project. Note: name must include full path including any event group names e.g. "group1/group2/myevent"

mode

event

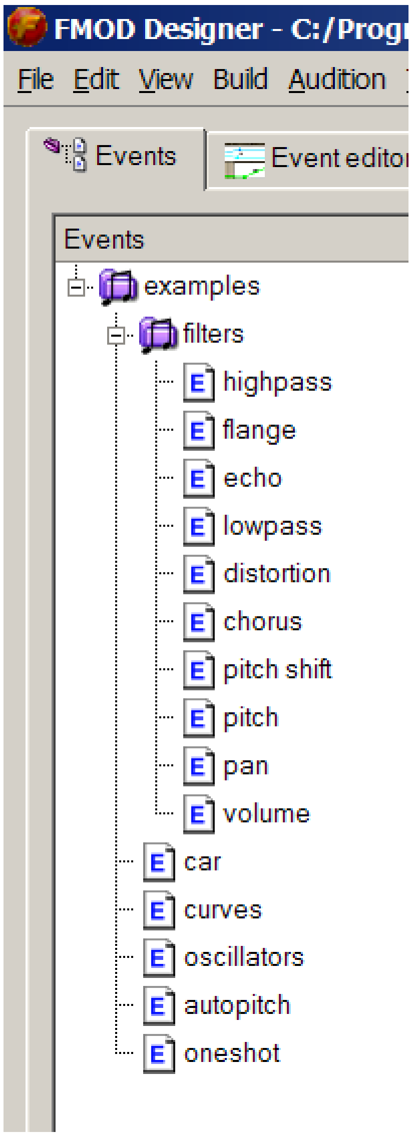
Address of a variable to receive the selected event within this event project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with "**echo**" as the name parameter.

If the programmer does not know which events are available, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.

Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an event handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getEventByProjectID
- EventGroup::getEventByIndex
- EventSystem::getGroup
- EventGroup::getGroup
- FMOD_EVENT_MODE

EventProject::getEventByProjectID

Retrieve an event handle by a project unique id.?

Syntax

```
  FMOD_RESULT EventProject::getEventByProjectID(
  unsigned int  projectid,
   FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

projectid

The project id of an event within this event project. Unique ids can be found in the programmer report generated when the project is built, and the C header.

mode

event

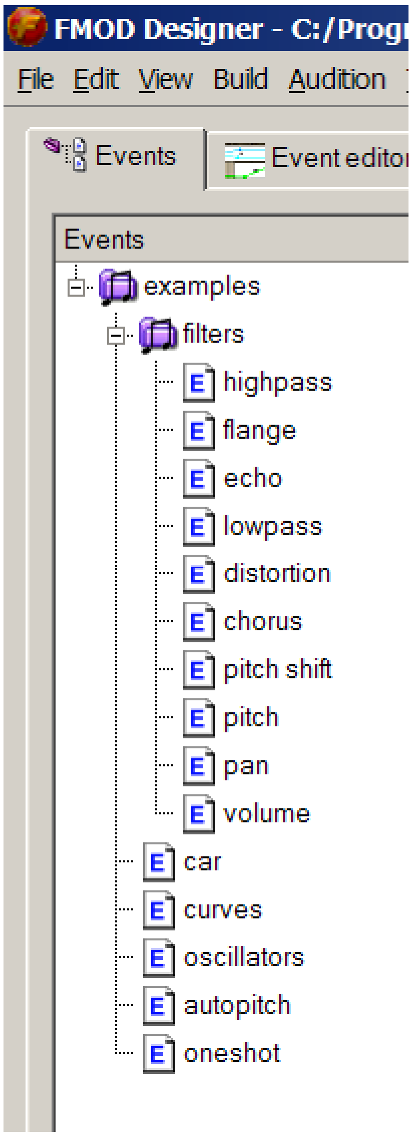
Address of a variable to receive the selected event within this event project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with "**echo**" as the name parameter.

If the programmer does not know which events are available, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.

Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an event handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getEvent
- EventProject::getNumEvents
- EventGroup::getEventByIndex
- EventSystem::getGroup
- EventGroup::getGroup
- FMOD_EVENT_MODE

EventProject::getGroup

Retrieves an event group object by name.?

Syntax

```
  FMOD_RESULT EventProject::getGroup(
  const char *  name,
   bool  cacheevents,
  EventGroup **  group
);
```

Parameters

name

The name of an event group that belongs to this event project.

cacheevents

If cacheevents is true then all event instances within this event group will be pre-allocated so that there are no memory allocs when getEvent is called.

group

Address of a variable to receive the selected event group within this event project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getGroupByIndex
- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventProject::getGroupByIndex

Retrieves an event group object by index.?

Syntax

```
FMOD_RESULT EventProject::getGroupByIndex(
  int  index,
  bool  cacheevents,
  EventGroup **  group
);
```

Parameters

index

The index of an event group within this event project. Indices are 0 based.

cacheevents

If cacheevents is true then all event instances within this event group will be pre-allocated so that there are no memory allocs when getEvent is called.

group

Address of a variable to receive the selected event group within this event project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getNumGroups
- EventProject::getGroup
- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventProject::getInfo

Retrieve information about this event project.?

Syntax

```
  FMOD_RESULT EventProject::getInfo(
  int * index,
  char ** name
);
```

Parameters

index

Address of a variable to receive the event project index.

name

Address of a variable to receive the event project name.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getInfo
- Event::getInfo

EventProject::getNumEvents

Returns the total number of events for this project only.?

Syntax

```
  FMOD_RESULT EventProject::getNumEvents(
  int *  numevents
);
```

Parameters

numevents

Address of a variable to receive the number of events in this project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getEventByProjectID

EventProject::getNumGroups

Retrieves the number of event groups stored within this event project.?

Syntax

```
  FMOD_RESULT EventProject::getNumGroups(
  int *  numgroups
);
```

Parameters

numgroups

Adress of a variable to receive the number of groups within this event project.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getGroupByIndex

EventProject::getUserData

Retrieves the user value that that was set by calling the EventProject::setUserData function.?

Syntax

```
  FMOD_RESULT EventProject::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventProject::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::setUserData

EventProject::release

Release this event project and all the events/eventgroups that it contains.?

Syntax

```
FMOD_RESULT EventProject::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getProject
- EventSystem::getProjectByIndex

EventProject::setUserData

Sets a user value that the EventProject object will store internally. Can be retrieved with EventProject::getUserData.?

Syntax

```
  FMOD_RESULT EventProject::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventProject object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventProject::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventProject::getUserData

EventGroup Interface

EventGroup::freeEventData

Free the resources for an EventGroup and all subgroups under it or for just a single specified event.?

Syntax

```
  FMOD_RESULT EventGroup::freeEventData(
  Event * event,
   bool waituntilready
);
```

Parameters

event

Single event to free resources for. Specify 0 or NULL to ignore.

waituntilready

If TRUE, this function will block until all pending asynchronous loads have completed before freeing the event data. If FALSE, this function will return FMOD_ERR_NOTREADY if any asynchronous loads are pending and it will NOT free any event data. Default = TRUE.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

If no event is specified then resources for all events in this EventGroup, and all EventGroups within this EventGroup, will be freed.
If an event is specified then just the resources for that event will be freed.
NOTE: This function does not completely remove events from memory, it simply frees any resources allocated by them. To completely remove events from memory, use EventSystem::unload.
Use waituntilready = false in time-critical situations to avoid blocking the main thread. Note that if FMOD_ERR_NOTREADY is returned from this function then no event data was actually freed - you will need to call this function again until it succeeds.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::loadEventData
- EventSystem::getGroup

EventGroup::getEvent

Retrieve a an event object by name.?

Syntax

```
  FMOD_RESULT EventGroup::getEvent(
  const char *  name,
   FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

name

The name of an event within this event group.

mode

event

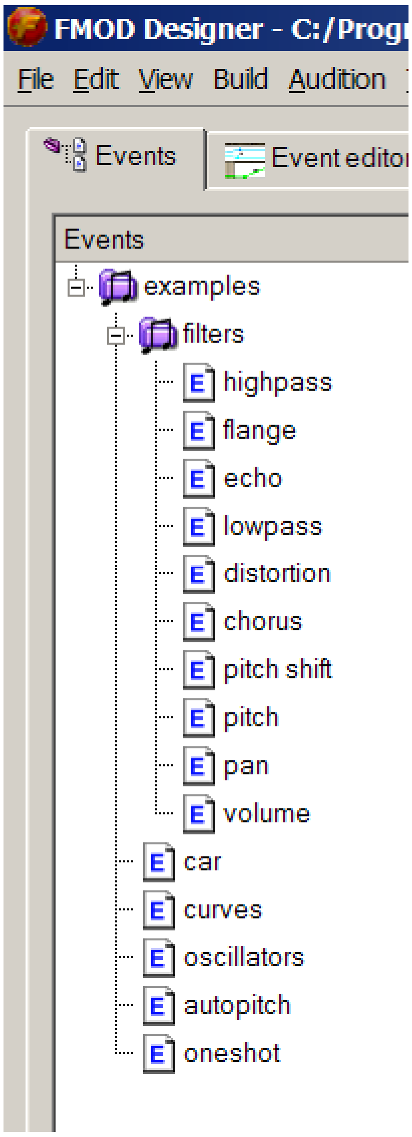
Address of a variable to receive the selected event within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with "**echo**" as the name parameter.

If the programmer does not know which events are available, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.

Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an invent handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getEventByIndex
- EventSystem::getGroup
- EventGroup::getGroup
- FMOD_EVENT_MODE

EventGroup::getEventByIndex

Retrieve an event object by index for this group.?

Syntax

```
  FMOD_RESULT EventGroup::getEventByIndex(
  int index,
   FMOD_EVENT_MODE  mode,
  Event ** event
);
```

Parameters

index

The index of an event within this event sub-group. Indices are 0 based.

mode

event

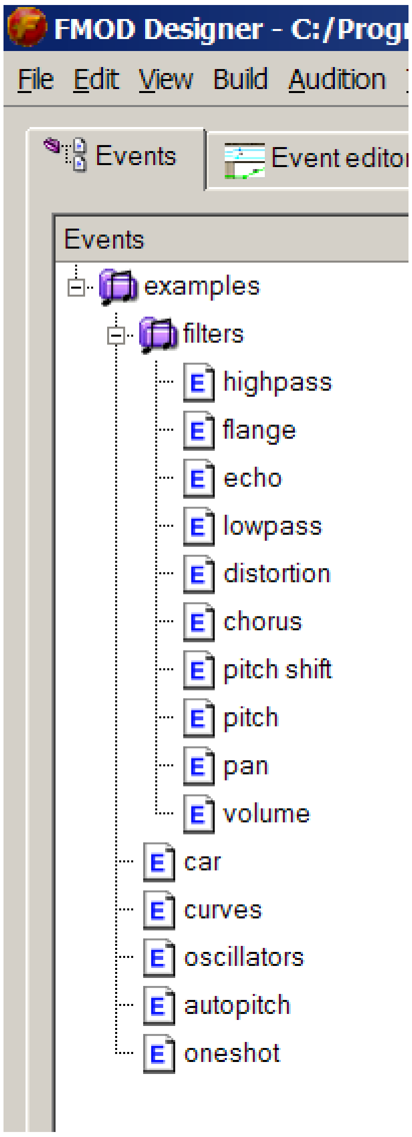
Address of a variable to receive the selected event within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event?
An event is the leaf of the event group tree. It is the actual sound to be played with complex behaviour designed by the sound designer.



In this case we are retrieving an event from an **event group**, so with the "**filters**" group we could get the echo event with **2** as the index parameter.

If the programmer does not know which events are available or which event index matches which event name, the sound designer tool can output a programmer report that lists the event group's events with the appropriate names and indices listed alongside them.
The only benefit of retrieving an object by index is that it is slightly faster to do so than to retrieve it by name.
Note!
- An event is retrieved from a pool of events (created earlier if FMOD_EVENT_CACHEEVENTS flag was set in EventSystem::getGroup / EventGroup::getGroup).
- Data may not be loaded from the disk for this event, so this event may trigger disk access. If you wish to pre-emp this use EventGroup::loadEventData first. - The pool of events has a size determined by the 'max playbacks' property in the FMOD Designer tool in the event's property sheet.
- The pointer to will be getting will be a pointer to one of these event instances.
- If you call this function more times than there are event instances, then an invent handle may be stolen, or may fail. This behaviour also determined by the sound designer. The behaviour may be to steal the oldest event in the pool, steal the quietest event in the pool, or simply fail this getEvent and return null as the event handle.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getEvent
- EventGroup::getNumEvents
- FMOD_EVENT_MODE

EventGroup::getGroup

Retrieves an event group's sub-group object by name.?

Syntax

```
  FMOD_RESULT EventGroup::getGroup(
  const char *  name,
   bool  cacheevents,
  EventGroup **  group
);
```

Parameters

name

The name of an event sub-group that belongs to this event group.

cacheevents

If cacheevents is true then all event instances within this event group will be pre-allocated so that there are no memory allocs when getEvent is called.

group

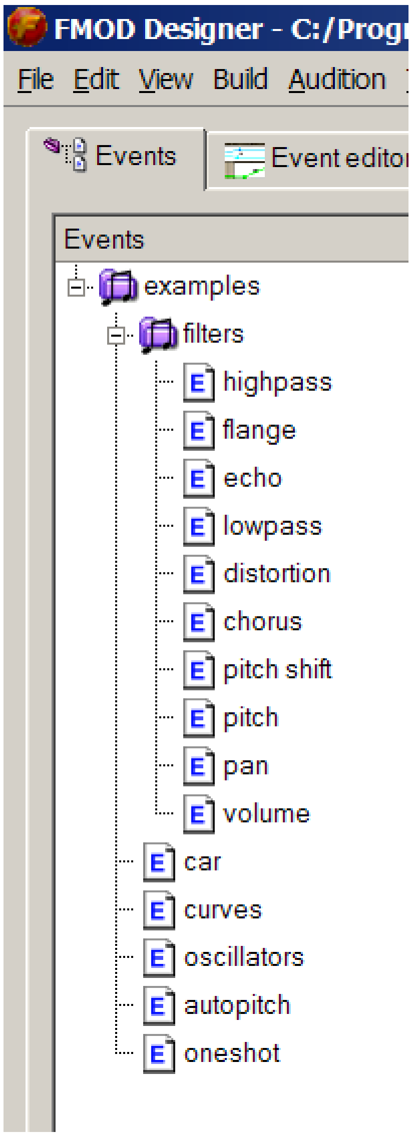
Address of a variable to receive the selected event sub-group within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event group?
An event group is a "folder" that stores events or sub-folders. With these folders a hierarchical tree can be built to store events in a more logical manner.



In this case we are retrieving an event group from another **event group**, so if this event group object was "**examples**" we could then get the event group "**filters**" with "**filters**" as the name parameter.
In this example "**filters**" is the only sub-group below "**examples**" so no other sub-groups are available here.

If the programmer does not know which sub-groups are available or which sub-group index matches which sub-group name, the sound designer tool can output a programmer report that lists the group's sub-groups with the appropriate names and indices listed alongside them.
The only benefit of retrieving an object by index is that it is slightly faster to do so than to retrieve it by name.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getGroup
- EventGroup::getGroupByIndex

EventGroup::getGroupByIndex

Retrieves an event group's sub-group object by index.?

Syntax

```
  FMOD_RESULT EventGroup::getGroupByIndex(
  int index,
   bool cacheevents,
  EventGroup ** group
);
```

Parameters

index

The index of an event sub-group within this event group. Indices are 0 based.

cacheevents

If cacheevents is true then all event instances within this event group will be pre-allocated so that there are no memory allocs when getEvent is called.

group

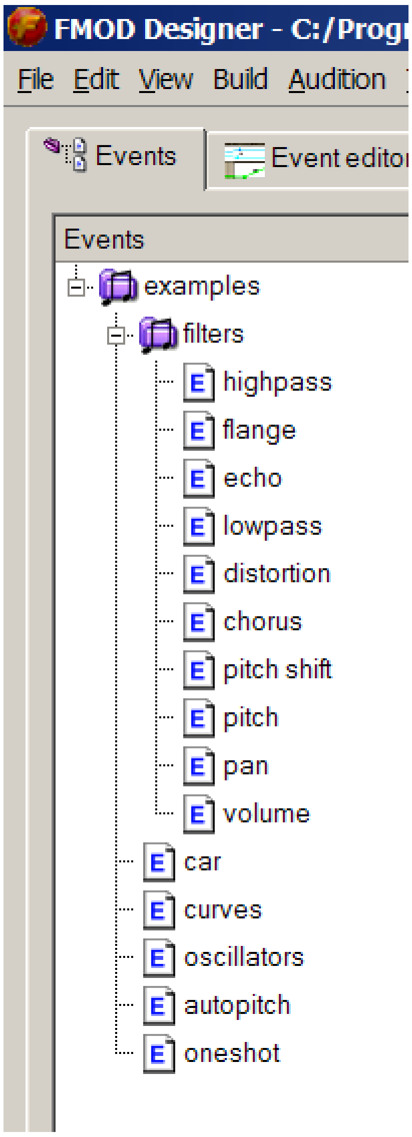
Address of a variable to receive the selected event sub-group within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

What is an event group?
An event group is a "folder" that stores events or sub-folders. With these folders a hierarchical tree can be built to store events in a more logical manner.



In this case we are retrieving an event group from another **event group**, so if this event group object was "**examples**" we could then get the event group "**filters**" with **0** as the index parameter.
In this example "**filters**" is the only sub-group below "**examples**" so no other sub-groups are available here.

If the programmer does not know which groups are available or which event group index matches which group name, the sound designer tool can output a programmer report that lists the group's sub-groups with the appropriate names and indices listed alongside them.
The only benefit of retrieving an object by index is that it is slightly faster to do so than to retrieve it by name.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getGroup
- EventGroup::getGroup
- EventGroup::getNumGroups

EventGroup::getInfo

Retrieve information about this event group.?

Syntax

```
  FMOD_RESULT EventGroup::getInfo(
  int * index,
  char ** name
);
```

Parameters

index

Address of a variable to receive the event group index.

name

Address of a variable to receive the event group name.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventParameter::getInfo
- Event::getInfo

EventGroup::getNumEvents

Retrieves the number of event events stored within this event group.?

Syntax

```
  FMOD_RESULT EventGroup::getNumEvents(
  int * numevents
);
```

Parameters

numevents

Adress of a variable to receive the number of events within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getEventByIndex

EventGroup::getNumGroups

Retrieves the number of event groups stored within this event group.?

Syntax

```
  FMOD_RESULT EventGroup::getNumGroups(
  int *  numgroups
);
```

Parameters

numgroups

Adress of a variable to receive the number of groups within this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getGroupByIndex

EventGroup::getNumProperties

Retrieve the number of properties for an event group.?

Syntax

```
  FMOD_RESULT EventGroup::getNumProperties(
  int *  numproperties
);
```

Parameters

numproperties

Address of a variable to receive the number of properties for this event group.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getPropertyByIndex

EventGroup::getParentGroup

Retrieves the eventgroup object to which this eventgroup belongs.?

Syntax

```
  FMOD_RESULT EventGroup::getParentGroup(
  EventGroup ** group
);
```

Parameters

group

Address of a variable that receives a pointer to the eventgroup's parent eventgroup

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The top level eventgroup will return a parent of 0 or NULL, as it has no parent.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventGroup::getParentProject

Retrieves the eventproject object to which this eventgroup belongs.?

Syntax

```
  FMOD_RESULT EventGroup::getParentProject(
  EventProject **  project
);
```

Parameters

project

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getGroup
- EventGroup::getGroupByIndex

EventGroup::getProperty

Retrieve an event group property by name.?

Syntax

```
  FMOD_RESULT EventGroup::getProperty(
  const char *  propertyname,
   void *  value
);
```

Parameters

propertyname

Name of the property to retrieve. This is the name that was specified in FMOD Designer.

value

Address of a variable to receive the event group property.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getPropertyByIndex
- EventGroup::getNumProperties
- EventGroup::getEvent

EventGroup::getPropertyByIndex

Retrieve an event group property by index.?

Syntax

```
  FMOD_RESULT EventGroup::getPropertyByIndex(
  int  propertyindex,
   void *  value
);
```

Parameters

propertyindex

Index of the property to retrieve.

value

Address of a variable to receive the event group property.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getProperty
- EventGroup::getNumProperties
- EventGroup::getEvent

EventGroup::getState

Retrieves the current state of an event group.?

Syntax

```
  FMOD_RESULT EventGroup::getState(
    FMOD_EVENT_STATE * state
);
```

Parameters

state

Address of a variable that receives the event group's current state.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

When FMOD_EVENT_STATE_PLAYING is true, at least one event in the group is playing. When FMOD_EVENT_STATE_PLAYING is false, no event in the group is playing.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_EVENT_STATE

EventGroup::getUserData

Retrieves the user value that that was set by calling the EventGroup::setUserData function.?

Syntax

```
  FMOD_RESULT EventGroup::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventGroup::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::setUserData

EventGroup::loadEventData

Loads the resources for all events within an event group.?

Syntax

```
FMOD_RESULT EventGroup::loadEventData(
  FMOD_EVENT_RESOURCE  resource,
  FMOD_EVENT_MODE  mode
);
```

Parameters

resource

Type of data to load. Either load samples, streams, or both. See FMOD_EVENT_RESOURCE.

mode

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use EventGroup::freeEventData to unload sample banks and close streams under this group.
Note that if another event in a different group is still using the sound data, it will not be freed until those events have had freeEventData called on them as well. (On their parent group).

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::freeEventData
- EventSystem::getGroup
- FMOD_EVENT_RESOURCE
- FMOD_EVENT_MODE

EventGroup::setUserData

Sets a user value that the EventGroup object will store internally. Can be retrieved with EventGroup::getUserData.?

Syntax

```
  FMOD_RESULT EventGroup::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventGroup object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventGroup::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getUserData

Event Interface

Event::get3DAttributes

Retrieves the position and velocity of an event.?

Syntax

```
  FMOD_RESULT Event::get3DAttributes(
    FMOD_VECTOR *  position,
    FMOD_VECTOR *  velocity,
    FMOD_VECTOR * orientation
);
```

Parameters

position

Address of a variable that receives the position in 3D space of the event. Optional. Specify 0 to ignore.

velocity

Address of a variable that receives the velocity in 'distance units per second' in 3D space of the event. See remarks. Optional. Specify 0 to ignore.

orientation

Address of a variable that receives the orientation of the event. Optional. Specify 0 to ignore. Only used for events with sound cones specified.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified in the FMOD Designer tool and are the distance units used by the game (i.e. feet, meters, inches, centimeters etc). An event has to be 3D to have its 3d position and velocity set.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::set3DAttributes

Event::get3DOcclusion

Retrieves the the EAX or software based occlusion factors for an event.?

Syntax

```
  FMOD_RESULT Event::get3DOcclusion(
   float *  directocclusion,
   float *  reverbocclusion
);
```

Parameters

directocclusion

Address of a variable that receives the occlusion factor for a voice for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

reverbocclusion

Address of a variable that receives the occlusion factor for a voice for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0. Optional. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

With EAX based sound cards, or I3DL2 based hardware accelerated voices, this will attenuate the sound and frequencies. With non EAX or I3DL2 harward accelerated voices, then the volume is attenuated by the directOcclusion factor.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::set3DOcclusion

Event::getCategory

Retrieve an event category that this event belongs to.?

Syntax

```
  FMOD_RESULT Event::getCategory(
  EventCategory **  category
);
```

Parameters

category

Address of a variable to receive the event category.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getInfo
- EventGroup::getEvent

Event::getChannelGroup

Retrieves a pointer to a lower level ChannelGroup class, mainly so that the programmer can add a custom DSP effect with ChannelGroup::addDSP.?

Syntax

```
FMOD_RESULT Event::getChannelGroup(
  FMOD::ChannelGroup ** channelgroup
);
```

Parameters

channelgroup

Address of a variable to receive a pointer to a low level ChannelGroup class.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

Event::getInfo

Retrieves information about the event.?

Syntax

```
  FMOD_RESULT Event::getInfo(
  int * index,
  char ** name,
   FMOD_EVENT_INFO * info
);
```

Parameters

index

Address of a variable to receive the event group's index. Specify 0 or NULL to ignore.

name

Address of a variable to receive the event name. Specify 0 or NULL to ignore.

info

Address of an FMOD_EVENT_INFO structure to receive extended event information. Specify 0 or NULL to ignore.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The FMOD_EVENT_INFO structure has members that need to be initialized before Event::getInfo is called. Always initialize the FMOD_EVENT_INFO structure before calling Event::getInfo!

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getEvent
- EventGroup::getEventByIndex

- EventParameter::getInfo
- FMOD_EVENT_INFO

Event::getMute

Retrieves the muted state of an event.?

Syntax

```
  FMOD_RESULT Event::getMute(
    bool * mute
);
```

Parameters

mute

Address of a variable to receive the muted state of the event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setMute

Event::getNumParameters

Retrieve the number of parameters for an event.?

Syntax

```
  FMOD_RESULT Event::getNumParameters(
  int *  numparameters
);
```

Parameters

numparameters

Address of a variable to receive the number of parameters for this event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getParameterByIndex

Event::getNumProperties

Retrieve the number of properties for an event.?

Syntax

```
  FMOD_RESULT Event::getNumProperties(
  int * numproperties
);
```

Parameters

numproperties

Address of a variable to receive the number of properties for this event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getPropertyByIndex

Event::getParameter

Retrieve an event parameter object by name.?

Syntax

```
  FMOD_RESULT Event::getParameter(
  const char *  name,
  EventParameter **  parameter
);
```

Parameters

name

The name of an event parameter that belongs to this event.

parameter

Address of a variable to receive the selected event parameter within this event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Each event will have a list of parameters to control the playback behaviour of the event. For example, if a sound designer made a car engine event, one of the parameters might be 'RPM'.
If the programmer does not know which parameters are available, the sound designer tool can output a programmer report that lists the event's parameters with the appropriate names and indices listed alongside them.

Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getParameterByIndex
- EventGroup::getEvent
- EventGroup::getEventByIndex

- FMOD_EVENT_MODE

Event::getParameterByIndex

Retrieve an event parameter by index.?

Syntax

```
  FMOD_RESULT Event::getParameterByIndex(
  int  index,
  EventParameter **  parameter
);
```

Parameters

index

The index of an event parameter within this event. Indices are 0 based. Pass -1 to retrieve this event's primary parameter.

parameter

Address of a variable to receive the event parameter object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Each event will have a list of parameters to control the playback behaviour of the event. For example, if a sound designer made a car engine event, one of the parameters might be 'RPM'.
If the programmer does not know which parameters are available or which index matches which parameter, the sound designer tool can output a programmer report that lists the event's parameters with the appropriate names and indices listed alongside them.
The only benefit of retrieving a parameter by index is that it is slightly faster to do so than to retrieve it by name.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getParameter
- Event::getNumParameters
- EventGroup::getEvent

- EventGroup::getEventByIndex

Event::getParentGroup

Retrieves the eventgroup object to which this event belongs.?

Syntax

```
  FMOD_RESULT Event::getParentGroup(
  EventGroup ** group
);
```

Parameters

group

Address of a variable that receives a pointer to the event's parent eventgroup

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::getEvent
- EventGroup::getEventByIndex

Event::getPaused

Retrieves the paused state of an event.?

Syntax

```
  FMOD_RESULT Event::getPaused(
   bool *  paused
);
```

Parameters

paused

Address of a variable to receive the paused state of the event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setPaused

Event::getPitch

Retrieves the overall pitch of an event.?

Syntax

```
  FMOD_RESULT Event::getPitch(
    float * pitch,
    FMOD_EVENT_PITCHUNITS units
);
```

Parameters

pitch

Address of a variable to receive the current pitch level of the event. 0.0 = normal pitch (default).

units

The desired units for the retrieved pitch value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setPitch
- FMOD_EVENT_PITCHUNITS

Event::getProperty

Retrieve an event property by name.?

Syntax

```
  FMOD_RESULT Event::getProperty(
  const char *  propertyname,
   void *  value,
   bool  this_instance
);
```

Parameters

propertyname

Name of the property to retrieve. This is the name that was specified in FMOD Designer.

value

Address of a variable to receive the event property.

this_instance

If TRUE then retrieve the per-instance property value, if FALSE then retrieve the parent (FMOD_EVENT_INFOONLY) event's property value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A parent, or FMOD_EVENT_INFOONLY, event is the prototype that all its event instances are based on. An event instance obtained by using any of the getEventXXX functions will be initialized with the current property values of its parent event. After an event instance is obtained, its property values may be modified using Event::setProperty and Event::setPropertyByIndex so that they differ from their parent event's properties. Use the 'this_instance' parameter to specify whether to retrieve the property value of the parent event or the specific event instance.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getPropertyByIndex
- Event::setProperty
- Event::setPropertyByIndex
- EventGroup::getEvent

Event::getPropertyByIndex

Retrieve an event property by index.?

Syntax

```
  FMOD_RESULT Event::getPropertyByIndex(
  int  propertyindex,
   void *  value,
   bool  this_instance
);
```

Parameters

propertyindex

Index of the property to retrieve. See FMOD_EVENT_PROPERTY for details.

value

Address of a variable to receive the event property.

this_instance

If TRUE then retrieve the per-instance property value, if FALSE then retrieve the parent (FMOD_EVENT_INFOONLY) event's property value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A parent, or FMOD_EVENT_INFOONLY, event is the prototype that all its event instances are based on. An event instance obtained by using any of the getEventXXX functions will be initialized with the current property values of its parent event. After an event instance is obtained, its property values may be modified using Event::setProperty and Event::setPropertyByIndex so that they differ from their parent event's properties. Use the 'this_instance' parameter to specify whether to retrieve the property value of the parent event or the specific event instance.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getProperty
- Event::getNumProperties
- Event::setProperty
- Event::setPropertyByIndex
- EventGroup::getEvent
- FMOD_EVENT_PROPERTY

Event::getReverbProperties

Retrieves the current reverb properties for this event.?

Syntax

```
  FMOD_RESULT Event::getReverbProperties(
    FMOD_REVERB_CHANNELPROPERTIES *  prop
);
```

Parameters

prop

Address of a variable to receive the FSOUND_REVERB_CHANNELPROPERTIES information.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Xbox, Xbox360, PlayStation 2, PlayStation 3

See Also

- Event::setReverbProperties
- FMOD_REVERB_CHANNELPROPERTIES

Event::getState

Retrieves the current state of an event.?

Syntax

```
  FMOD_RESULT Event::getState(
    FMOD_EVENT_STATE * state
);
```

Parameters

state

Address of a variable that receives the event's current state.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::loadEventData
- EventGroup::getEvent
- FMOD_EVENT_STATE
- FMOD_EVENT_MODE

Event::getUserData

Retrieves the user value that that was set by calling the Event::setUserData function.?

Syntax

```
  FMOD_RESULT Event::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the Event::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setUserData

Event::getVolume

Retrieves the overall volume of an event.?

Syntax

```
  FMOD_RESULT Event::getVolume(
    float *  volume
);
```

Parameters

volume

Address of a variable to receive the current volume level of the event. 0.0 = silent, 1.0 = full volume (default).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setVolume

Event::set3DAttributes

Sets the 3d position and velocity of an event.?

Syntax

```
  FMOD_RESULT Event::set3DAttributes(
  const FMOD_VECTOR *  position,
  const FMOD_VECTOR *  velocity,
  const FMOD_VECTOR * orientation
);
```

Parameters

position

Position in 3D space of the event. Specifying 0 / null will ignore this parameter.

velocity

Velocity in 'distance units per second' in 3D space of the event. See remarks. Specifying 0 / null will ignore this parameter.

orientation

Orientation of the event sound cone. Only used if the event has a cone specified to determine cone detection, otherwise just specify 0 / null. Specifying 0 / null will ignore this parameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A 'distance unit' is specified in the FMOD Designer tool and are the distance units used by the game (i.e. feet, meters, inches, centimeters etc). An event has to be 3D to have its 3d position and velocity set.

Before getting an event with 'just fail if quietest' max playbacks behaviour, this function should be called on an EVENT_INFOONLY event to allow the event system to estimate volume. The various getEvent* functions copy the 3D attributes from the EVENT_INFOONLY event to the event that is returned, so it is not necessary to set the 3D attributes again after getting a real event.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::get3DAttributes

Event::set3DOcclusion

Sets the EAX or software based occlusion factors for an event.?This function can be called to produce the same audible effects as the FMOD geometry engine, just without the built in polygon processing.?

Syntax

```
  FMOD_RESULT Event::set3DOcclusion(
    float  directocclusion,
    float  reverbocclusion
);
```

Parameters

directocclusion

Occlusion factor for a voice for the direct path. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

reverbocclusion

Occlusion factor for a voice for the reverb mix. 0.0 = not occluded. 1.0 = fully occluded. Default = 0.0.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function does not go through and overwrite the channel occlusion factors, it calls ChannelGroup::set3DOcclusion. This means that final occlusion values will be affected by both Event occlusion and geometry (if any).

With EAX based sound cards, or I3DL2 based hardware accelerated voices, this will attenuate the sound and frequencies. With non EAX or I3DL2 harward accelerated voices, then the volume is attenuated by the directOcclusion factor.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::get3DOcclusion
- Event::setReverbProperties
- ChannelGroup::set3DOcclusion

Event::setCallback

Sets a callback so that when certain event behaviours happen, they can be caught by the user.?

Syntax

```
  FMOD_RESULT Event::setCallback(
    FMOD_EVENT_CALLBACK  callback,
    void *  userdata
);
```

Parameters

callback

Pointer to a callback to be called by FMOD.

userdata

Userdata pointer to be passed to callback.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

wav sync points are supported.
These can be created by placing 'markers' in the original source wavs using a tool such as Sound Forge or Cooledit.
Callbacks will be automatically generated when these markers are encountered.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_EVENT_CALLBACK
- FMOD_EVENT_CALLBACKTYPE

Event::setMute

Mutes or unmutes an event for runtime reasons.?

Syntax

```
  FMOD_RESULT Event::setMute(
    bool mute
);
```

Parameters

mute

Mute state of the event. true = muted, false = unmuted.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
An event can have several hardware/software voices playing under it at once so this function mutes all relevant voices for this event.

This function is not to be used unless needed for runtime reasons.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getMute

Event::setPaused

Pauses or unpauses an event for runtime reasons.?

Syntax

```
  FMOD_RESULT Event::setPaused(
    bool  paused
);
```

Parameters

paused

Paused state of the event. true = paused, false = unpaused.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.
An event can have several hardware/software voices playing under it at once so this function pauses all relevant voices for this event.

This function is not to be used unless needed for runtime reasons.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getPaused

Event::setPitch

Sets the overall pitch of an event.?

Syntax

```
  FMOD_RESULT Event::setPitch(
    float  pitch,
    FMOD_EVENT_PITCHUNITS  units
);
```

Parameters

pitch

Pitch level of the event. 0.0 = normal pitch (default).

units

The units in which the new pitch level is specified.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

An event can have several hardware/software voices playing under it at once so this function scales all relevant voice pitches for this event.
This function is not to be used unless needed for runtime reasons, as the sound designer will have set the appropriate event pitch level in the FMOD Designer tool.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getPitch
- FMOD_EVENT_PITCHUNITS

Event::setProperty

Set an event property by name.?

Syntax

```
  FMOD_RESULT Event::setProperty(
  const char *  propertyname,
   void *  value,
   bool  this_instance
);
```

Parameters

propertyname

Name of the property to set. This is the name that was specified in FMOD Designer.

value

Pointer to the new value for this event property.

this_instance

If TRUE then set the per-instance property value, if FALSE then set the property value of all event instances and also the parent (FMOD_EVENT_INFOONLY) event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A parent, or FMOD_EVENT_INFOONLY, event is the prototype that all its event instances are based on. An event instance obtained by using any of the getEventXXX functions will be initialized with the current property values of its parent event. After an event instance is obtained, its property values may be modified using Event::setProperty and Event::setPropertyByIndex so that they differ from their parent event's properties. Use the 'this_instance' parameter to specify whether to set the property value of all event instances and also the parent event or just the specific event instance.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getProperty
- Event::setPropertyByIndex
- EventGroup::getEvent

Event::setPropertyByIndex

Set an event property by index.?

Syntax

```
  FMOD_RESULT Event::setPropertyByIndex(
  int  propertyindex,
   void *  value,
   bool  this_instance
);
```

Parameters

propertyindex

Index of the property to set. See FMOD_EVENT_PROPERTY for details.

value

Pointer to the new value for this event property.

this_instance

If TRUE then set the per-instance property value, if FALSE then set the property value of all event instances and also the parent (FMOD_EVENT_INFOONLY) event.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

A parent, or FMOD_EVENT_INFOONLY, event is the prototype that all its event instances are based on. An event instance obtained by using any of the getEventXXX functions will be initialized with the current property values of its parent event. After an event instance is obtained, its property values may be modified using Event::setProperty and Event::setPropertyByIndex so that they differ from their parent event's properties. Use the 'this_instance' parameter to specify whether to set the property value of all event instances and also the parent event or just the specific event instance.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getProperty
- Event::setProperty
- EventGroup::getEvent
- FMOD_EVENT_PROPERTY

Event::setReverbProperties

Sets the event specific reverb properties for sounds created with FMOD_HARDWARE, including things like wet/dry mix (room size), and things like obstruction and occlusion properties.?

Syntax

```
  FMOD_RESULT Event::setReverbProperties(
  const FMOD_REVERB_CHANNELPROPERTIES *  prop
);
```

Parameters

prop

Pointer to a FMOD_REVERB_CHANNELPROPERTIES structure definition.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Under Win32 / Win64, you must be using FMOD_OUTPUTTYPE_DSOUND as the output mode for this to work. In DSound mode, the reverb will only work if you have an EAX compatible soundcard such as the Sound Blaster, and your sound was created with the FMOD_HARDWARE and FMOD_3D flags.

On PlayStation2, the 'Room' parameter is the only parameter supported. The hardware only allows 'on' or 'off', so the reverb will be off when 'Room' is -10000 and on for every other value.

On Xbox, it is possible to apply reverb to FMOD_2D based voices using this function. By default reverb is turned off for FMOD_2D based voices.
NOTE: This function overrides values set with Event::set3DOcclusion and also overrides the "Reverb Level" property defined using the FMOD Designer tool. If you need Event::set3DOcclusion or "3D Reverb Level" functionality then factor it into your FMOD_REVERB_CHANNELPROPERTIES values.

Platforms Supported

Win32, Win64, Xbox, Xbox360, PlayStation 2, PlayStation 3

See Also

- Event::getReverbProperties
- System::setReverbProperties

- FMOD_REVERB_CHANNELPROPERTIES
- Event::set3DOcclusion

Event::setUserData

Sets a user value that the Event object will store internally. Can be retrieved with Event::getUserData.?

Syntax

```
  FMOD_RESULT Event::setUserData(
   void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the Event object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using Event::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getUserData

Event::setVolume

Sets the overall volume of an event.?

Syntax

```
  FMOD_RESULT Event::setVolume(
    float  volume
);
```

Parameters

volume

Volume level of the event. 0.0 = silent, 1.0 = full volume (default).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

An event can have several hardware/software voices playing under it at once so this function scales all relevant voice volumes for this event.
This function is not to be used unless needed for runtime reasons, as the sound designer will have set the appropriate event volume level in the FMOD Designer tool.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getVolume

Event::start

Start this event playing.?

Syntax

```
FMOD_RESULT Event::start();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Call Event::stop to halt playback of an event.

Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::stop
- EventGroup::getEvent
- EventGroup::getEventByIndex
- FMOD_EVENT_MODE

Event::stop

Stop this event playing.?

Syntax

```
  FMOD_RESULT Event::stop(
    bool immediate
);
```

Parameters

immediate

Set this to true to force the event to stop immediately, ignoring the "Fadeout time" property

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::start
- EventGroup::getEvent
- EventGroup::getEventByIndex
- FMOD_EVENT_MODE

EventParameter Interface

EventParameter::getInfo

Retrieve information about this event parameter.?

Syntax

```
  FMOD_RESULT EventParameter::getInfo(
  int * index,
  char ** name
);
```

Parameters

index

Address of a variable to receive the event parameters index into the parent event.

name

Address of a variable to receive the event parameter name.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Mainly used for display purposes, this function returns a pointer to memory containing the event's name. Do not modify or try to free this memory.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::getRange

EventParameter::getRange

Retrieve the minimum and maximum values for this event parameter.?

Syntax

```
  FMOD_RESULT EventParameter::getRange(
    float *  rangemin,
    float *  rangemax
);
```

Parameters

rangemin

Address of variable to receive the minimum value allowed for this EventParameter.

rangemax

Address of variable to receive the maximum value allowed for this EventParameter.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This parameter is defined by the sound designer, and usually has a logical meaning, such as RPM for a car engine for example.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::setValue
- EventParameter::getValue
- EventParameter::getInfo

EventParameter::getSeekSpeed

Receieves the seek velocity of an event.?

Syntax

```
  FMOD_RESULT EventParameter::getSeekSpeed(
    float *  value
);
```

Parameters

value

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::setSeekSpeed

EventParameter::getUserData

Retrieves the user value that that was set by calling the EventParameter::setUserData function.?

Syntax

```
  FMOD_RESULT EventParameter::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventParameter::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventParameter::setUserData

EventParameter::getValue

Retrieve the current value of this parameter.?

Syntax

```
  FMOD_RESULT EventParameter::getValue(
    float *  value
);
```

Parameters

value

Address of variable to receive the parameter value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This parameter is defined by the sound designer, and has a minimum and maximum value. It usually has a logical meaning, such as RPM for a car engine for example.

Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::setValue
- EventParameter::getRange
- EventParameter::getInfo

EventParameter::getVelocity

Receieves the velocity of an event.?

Syntax

```
  FMOD_RESULT EventParameter::getVelocity(
    float *  value
);
```

Parameters

value

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::setVelocity

EventParameter::keyOff

Triggers a keyoff on an event parameter that has sustain points in it. If an event parameter is currently sustaining on a sustain point,?triggering a keyoff will release it and allow the parameter to continue.?

Syntax

```
FMOD_RESULT EventParameter::keyOff();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Keyoffs can be triggered in advance of a sustain point being reached, so that they continue past the sustain point ahead of time.
Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::setVelocity
- EventParameter::getVelocity

EventParameter::setSeekSpeed

Sets the seek velocity of a parameter.?

Syntax

```
  FMOD_RESULT EventParameter::setSeekSpeed(
   float  value
);
```

Parameters

value

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that currently setting the velocity of an event parameter will set the velocity for all instances of this event. This value is normally set by the sound designer but may be used if the programmer wishes to vary it.
Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::getSeekSpeed

EventParameter::setUserData

Sets a user value that the EventParameter object will store internally. Can be retrieved with EventParameter::getUserData.?

Syntax

```
FMOD_RESULT EventParameter::setUserData(
  void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventParameter object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventParameter::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventParameter::getUserData

EventParameter::setValue

Set the 'value' of this parameter.?

Syntax

```
  FMOD_RESULT EventParameter::setValue(
    float  value
);
```

Parameters

value

Value to set this parameter to. Note! Must lie in the range described by EventParameter::getRange.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This parameter is defined by the sound designer, and has a minimum and maximum value. It usually has a logical meaning, such as RPM for a car engine for example.

Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::getValue
- EventParameter::getRange

EventParameter::setVelocity

Sets the velocity of a parameter. In most cases the velocity of a parameter will be 0, unless it is a time based event.?

Syntax

```
  FMOD_RESULT EventParameter::setVelocity(
    float  value
);
```

Parameters

value

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Note that currently setting the velocity of an event parameter will set the velocity for all instances of this event. This value is normally set by the sound designer but may be used if the programmer wishes to pause or speed up / slow down the parameter movement.
Attempting to use this function on an FMOD_EVENT_INFOONLY event will cause an FMOD_ERR_INVALID_HANDLE error to be returned.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventParameter::getVelocity

EventCategory Interface

EventCategory::getCategory

Retrieve an event category object by name.?

Syntax

```
  FMOD_RESULT EventCategory::getCategory(
  const char * name,
  EventCategory ** category
);
```

Parameters

name

The name of an event category that belongs to this event category.

category

Address of a variable to receive the selected event category within this event category.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getCategoryByIndex
- EventSystem::getCategory

EventCategory::getCategoryByIndex

Retrieve an event category object by index.?

Syntax

```
  FMOD_RESULT EventCategory::getCategoryByIndex(
  int  index,
  EventCategory  **  category
);
```

Parameters

index

The index of an event category within this event category. Indices are 0 based.

category

Address of a variable to receive the event category object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getCategory
- EventSystem::getCategory

EventCategory::getChannelGroup

Retrieves a pointer to a lower level ChannelGroup class, mainly so that the programmer can add a custom DSP effect with ChannelGroup::addDSP.?

Syntax

```
  FMOD_RESULT EventCategory::getChannelGroup(
    FMOD::ChannelGroup ** channelgroup
);
```

Parameters

channelgroup

Address of a variable to receive a pointer to a low level ChannelGroup class.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

EventCategory::getEventByIndex

Retrieve an event object by index.?

Syntax

```
  FMOD_RESULT EventCategory::getEventByIndex(
  int index,
   FMOD_EVENT_MODE eventmode,
  Event ** event
);
```

Parameters

index

The index of an event within this event category. Indices are 0 based.

eventmode

event

Address of a variable to receive the event object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategory
- EventCategory::getNumEvents

EventCategory::getInfo

Retrieve information about this event category.?

Syntax

```
  FMOD_RESULT EventCategory::getInfo(
  int * index,
  char ** name
);
```

Parameters

index

Address of a variable to receive the event category's index.

name

Address of a variable to receive the event category's name.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Mainly used for display purposes, this function returns a pointer to memory containing the event category's name. Do not modify or try to free this memory.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategory

EventCategory::getMute

Retrieves the mute state of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::getMute(
    bool * mute
);
```

Parameters

mute

Address of a variable to receive the mute state of the event category.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::setMute

EventCategory::getNumCategories

Retrieve the number of sub-categories below this event category.?

Syntax

```
  FMOD_RESULT EventCategory::getNumCategories(
  int *  numcategories
);
```

Parameters

numcategories

Address of a variable to receive the number of categories in this category.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getCategory
- EventSystem::getCategory

EventCategory::getNumEvents

Retrieve the number of events within this event category.?

Syntax

```
  FMOD_RESULT EventCategory::getNumEvents(
  int *  numevents
);
```

Parameters

numevents

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getCategory
- EventCategory::getEventByIndex

EventCategory::getPaused

Retrieves the paused state of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::getPaused(
    bool *  paused
);
```

Parameters

paused

Address of a variable to receive the paused state of the event category.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::setPaused

EventCategory::getPitch

Retrieves the overall pitch of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::getPitch(
    float * pitch,
    FMOD_EVENT_PITCHUNITS units
);
```

Parameters

pitch

Address of a variable to receive the current pitch level of the event category. 0.0 = normal pitch (default).

units

The desired units for the retrieved pitch value.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::setPitch
- FMOD_EVENT_PITCHUNITS

EventCategory::getUserData

Retrieves the user value that that was set by calling the EventCategory::setUserData function.?

Syntax

```
  FMOD_RESULT EventCategory::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventCategory::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::setUserData

EventCategory::getVolume

Retrieves the overall volume of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::getVolume(
    float *  volume
);
```

Parameters

volume

Address of a variable to receive the current volume level of the event category. 0.0 = silent, 1.0 = full volume (default).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::setVolume

EventCategory::setMute

Pauses or unpauses an event category for runtime reasons.?

Syntax

```
  FMOD_RESULT EventCategory::setMute(
    bool mute
);
```

Parameters

mute

Mute the event category. true = muted, false = unmuted.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getMute

EventCategory::setPaused

Pauses or unpauses an event category for runtime reasons.?

Syntax

```
  FMOD_RESULT EventCategory::setPaused(
   bool  paused
);
```

Parameters

paused

Paused state of the event category. true = paused, false = unpaused.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getPaused

EventCategory::setPitch

Sets the overall pitch of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::setPitch(
   float  pitch,
   FMOD_EVENT_PITCHUNITS  units
);
```

Parameters

pitch

Pitch level of the event category. 0.0 = normal pitch (default).

units

The units in which the new pitch level is specified.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getPitch
- FMOD_EVENT_PITCHUNITS

EventCategory::setUserData

Sets a user value that the EventCategory object will store internally. Can be retrieved with EventCategory::getUserData.?

Syntax

```
FMOD_RESULT EventCategory::setUserData(
  void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventCategory object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventCategory::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getUserData

EventCategory::setVolume

Sets the overall volume of an event category.?

Syntax

```
  FMOD_RESULT EventCategory::setVolume(
    float  volume
);
```

Parameters

volume

Volume level of the event category. 0.0 = silent, 1.0 = full volume (default).

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

An event category can have several events playing under it at once so this function scales all relevant event volumes for this event category.
This function is not to be used unless needed for runtime reasons, as the sound designer will have set the appropriate event category volume level in the FMOD Designer tool.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventCategory::getVolume

EventCategory::stopAllEvents

Stops all events in this category and subcategories.?

Syntax

```
FMOD_RESULT EventCategory::stopAllEvents();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

EventReverb Interface

EventReverb::get3DAttributes

Gets the 3D attributes of the event 3D reverb object?

Syntax

```
FMOD_RESULT EventReverb::get3DAttributes(
  FMOD_VECTOR *  position,
  float *  mindistance,
  float *  maxdistance
);
```

Parameters

position

pointer to a vector in 3D space where the reverb is centred

mindistance

radius within which the reverb has full effect

maxdistance

radius outside of which the reverb has zero effect

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::set3DAttributes

EventReverb::getActive

Retrieves the active state of the reverb object.?

Syntax

```
  FMOD_RESULT EventReverb::getActive(
    bool * active
);
```

Parameters

active

Address of a variable to receive the current active state of the reverb object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::setActive
- EventSystem::createReverb

EventReverb::getProperties

Retrieves the current reverb properties for this event 3d reverb object.?

Syntax

```
  FMOD_RESULT EventReverb::getProperties(
    FMOD_REVERB_PROPERTIES *  props
);
```

Parameters

props

Address of a variable to receive the FMOD_REVERB_PROPERTIES information.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::setProperties
- FMOD_REVERB_PROPERTIES

EventReverb::getUserData

Retrieves the user value that that was set by calling the EventReverb::setUserData function.?

Syntax

```
  FMOD_RESULT EventReverb::getUserData(
    void ** userdata
);
```

Parameters

userdata

Address of a pointer that receives the data specified with the EventReverb::setUserData function.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::setUserData

EventReverb::release

Release memory for an event reverb object.?

Syntax

```
FMOD_RESULT EventReverb::release();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This will release this event reverb object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::getReverbPreset
- EventSystem::getReverbPresetByIndex

EventReverb::set3DAttributes

Sets the 3D attributes of the event 3D reverb object?

Syntax

```
  FMOD_RESULT EventReverb::set3DAttributes(
  const FMOD_VECTOR *  position,
    float  mindistance,
    float  maxdistance
);
```

Parameters

position

pointer to a vector in 3D space where the reverb is centred

mindistance

radius within which the reverb has full effect

maxdistance

radius outside of which the reverb has zero effect

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

The 3D reverb object is a sphere having 3D attributes (position, minimum distance, maximum distance) and reverb properties.
The properties and 3D attributes of all reverb objects collectively determine, along with the listener's position, the settings of and input gains into a single 3D reverb DSP.
Please note that this only applies to software channels. When the listener is within the sphere of effect of one or more 3d reverbs, the listener's 3D reverb properties are a weighted combination of such 3d reverbs. When the listener is outside all of the reverbs, the 3D reverb setting is set to the default ambient reverb setting.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::get3DAttributes

EventReverb::setActive

Disables or enables a reverb object so that it does or does not contribute to the 3d scene.?

Syntax

```
  FMOD_RESULT EventReverb::setActive(
    bool active
);
```

Parameters

active

true = active, false = not active. Default = true.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::getActive
- EventSystem::createReverb

EventReverb::setProperties

Sets the reverb properties for this event 3d reverb object.?

Syntax

```
  FMOD_RESULT EventReverb::setProperties(
  const FMOD_REVERB_PROPERTIES *  props
);
```

Parameters

props

Pointer to a FMOD_REVERB_PROPERTIES structure definition.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

EventSystem::getReverbPreset and EventSystem::getReverbPresetByIndex can be used to retrieve sound designer defined presets, or it can be set programmatically.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::getProperties
- FMOD_REVERB_PROPERTIES
- EventSystem::getReverbPreset
- EventSystem::getReverbPresetByIndex
- EventSystem::createReverb

EventReverb::setUserData

Sets a user value that the EventReverb object will store internally. Can be retrieved with EventReverb::getUserData.?

Syntax

```
  FMOD_RESULT EventReverb::setUserData(
    void * userdata
);
```

Parameters

userdata

Address of user data that the user wishes stored within the EventReverb object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

This function is primarily used in case the user wishes to 'attach' data to an FMOD object.
It can be useful if an FMOD callback passes an object of this type as a parameter, and the user does not know which object it is (if many of these types of objects exist). Using EventReverb::getUserData would help in the identification of the object.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventReverb::getUserData

Functions

EventSystem_Create

EventSystem_Create

Factory function to create an EventSystem object. This must be called to create an FMOD System object before you can do anything else.
?Use this function to create 1, or multiple instances of FMOD System objects.?

Syntax

```
  FMOD_RESULT EventSystem_Create(
  EventSystem ** eventsystem
);
```

Parameters

eventsystem

Address of a pointer that receives the new FMOD EventSystem object.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Use EventSystem::release to free an eventsystem object.

It is generally recommended to only create one system object. Creating more than one can lead to excess cpu usage as it will spawn extra software mixer threads.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventSystem::release

Callbacks

FMOD_EVENT_CALLBACK

FMOD_EVENT_CALLBACK

Event callback that is called when one of the events listed in FMOD_EVENT_CALLBACKTYPE occurs.?

Syntax

```
  FMOD_RESULT F_CALLBACK  FMOD_EVENT_CALLBACK(
    FMOD_EVENT * event,
    FMOD_EVENT_CALLBACKTYPE  type,
    void *  param1,
    void *  param2,
    void *  userdata
);
```

Parameters

event

Event handle in question.

type

Type of callback being issued. see FMOD_EVENT_CALLBACKTYPE for the different reasons FMOD will generate a callback for an event.

param1

Parameter 1 for the event callback. See remarks for different uses of param1.

param2

Parameter 2 for the event callback. See remarks for different uses of param2.

userdata

User specified value set when calling Event::setCallback.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

C++ Users. Cast **FMOD_EVENT *** to **FMOD::Event *** inside the callback and use as normal.

To avoid needing to process all messages simply switch on the messages you are interested in.

When the event callback is called. 'param1' and 'param2' mean different things depending on the type of callback.

Here the contents of param1 and param2 are listed.
The parameters are void *, but should be cast to the listed C type to get the correct value.

- FMOD_EVENT_CALLBACKTYPE_SYNCPOINT - param1 = (char *) name of sync point. param2 = (unsigned int) PCM offset of sync point.
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_START - param1 = (char *) name of sound definition being started. param2 = (int) index of wave being started inside sound definition (ie for multi wave sound definitions).
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_END - param1 = (char *) name of sound definition being stopped. param2 = (int) index of wave being started inside sound definition (ie for multi wave sound definitions).
- FMOD_EVENT_CALLBACKTYPE_STOLEN - param1 = 0. param2 = 0. If the callback function returns FMOD_ERR_EVENT_FAILED, the event will **not** be stolen, and the returned value will be passed back as the return value of the getEventXXX call that triggered the steal attempt.
- FMOD_EVENT_CALLBACKTYPE_EVENTFINISHED - param1 = 0. param2 = 0.
- FMOD_EVENT_CALLBACKTYPE_NET_MODIFIED - param1 = (FMOD_EVENT_PROPERTY) which property was modified. param2 = (float) the new property value.
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_CREATE - param1 = (char *) name of sound definition. param2 [in] = (int *) pointer to index of sound definition entry. param2 [out] = (FMOD::Sound **) pointer to a valid lower level API FMOD Sound handle.
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_RELEASE - param1 = (char *) name of sound definition. param2 = (FMOD::Sound *) the FMOD sound handle that was previously created in FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_CREATE.
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_INFO - param1 = (char *) name of sound definition. param2 = (FMOD::Sound *) the FMOD sound handle that FMOD will use for this sound definition.
- FMOD_EVENT_CALLBACKTYPE_EVENTSTARTED - param1 = 0. param2 = 0.
- FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_SELECTINDEX - param1 = (char *) name of sound definition. param2 [in] = (int *) pointer to number of entries in this sound definition. *param2 [out] = (int) index of sound definition entry to select.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, XBox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- FMOD_EVENT_CALLBACKTYPE
- FMOD_EVENT_PROPERTY
- Event::setCallback

Structures

FMOD_EVENT_INFO

Structure containing extended information about an event.?

Structure

```
typedef struct {
  int memoryused;
  int positionms;
  int lengthms;
  int channelsplaying;
  int instancesactive;
  char ** wavebanknames;
  unsigned int projectid;
  unsigned int systemid;
  float audibility;
  int numinstances;
  FMOD_EVENT ** instances;
} FMOD_EVENT_INFO;
```

Members

memoryused

[out] Amount of memory (in bytes) used by this event. DISABLED. Not working currently until further notice. Use FMOD::Memory_GetStats instead.

positionms

[out] Time passed in playback of this event instance in milliseconds.

lengthms

[out] Length in milliseconds of this event. Note: lengthms will be -1 if the length of the event can't be determined i.e. if it has looping sounds.

channelsplaying

[out] Number of channels currently playing in this event instance.

instancesactive

[out] Number of event instances currently in use.

wavebanknames

[out] An array containing the names of all wave banks that are referenced by this event.

projectid

[out] The runtime 'EventProject' wide unique identifier for this event.

systemid

[out] The runtime 'EventSystem' wide unique identifier for this event. This is calculated when single or multiple projects are loaded.

audibility

[out] current audibility of event.

numinstances

[in/out] On entry, maximum number of entries in instances array. On exit, actual number of entries in instances array, or if instances is null, then it is just the number of instances of this event. Optional.

instances

[in/out] Pointer to an array that will be filled with the current reference-counted event handles of all instances of this event. Optional. Specify 0 if not needed. Must be used in conjunction with numinstances. Note: Due to reference counting, the event instance handles returned here may be different between subsequent calls to this function. If you use these event handles, make sure your code is prepared for them to be invalid!

Remarks

This structure is optional! Specify 0 or NULL in Event::getInfo if you don't need it!
This structure has members that need to be initialized before Event::getInfo is called. Always initialize this structure before calling Event::getInfo!

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- Event::getInfo

FMOD_EVENT_LOADINFO

Use this structure with EventSystem::load when more control is needed over loading.?

Structure

```
  typedef struct {
  unsigned int size;
  char * encryptionkey;
   float sounddefentrylimit;
  unsigned int  loadfrommemory_length;
}  FMOD_EVENT_LOADINFO;
```

Members

size

[in] Size of this structure. This is used so the structure can be expanded in the future and still work on older versions of FMOD Ex.

encryptionkey

[in] Optional. Specify 0 to ignore. Key, or 'password' to decrypt a bank. A sound designer may have encrypted the audio data to protect their sound data from 'rippers'.

sounddefentrylimit

[in] Optional. Specify 0 to ignore. A value between 0 -> 1 that is multiplied with the number of sound definition entries in each sound definition in the project being loaded in order to programmatically reduce the number of sound definition entries used at runtime.

loadfrommemory_length

[in] Optional. Specify 0 to ignore. Length of memory buffer pointed to by name_or_data parameter passed to EventSystem::load. If this field is non-zero then the name_or_data parameter passed to EventSystem::load will be interpreted as a pointer to a memory buffer containing the .fev data to load. If this field is zero the name_or_data parameter is interpreted as the filename of the .fev file to load.

Remarks

This structure is optional! Specify 0 or NULL in EventSystem::load if you don't need it!

Members marked with [in] mean the user sets the value before passing it to the function.
Members marked with [out] mean FMOD sets the value to be used after the function exits.
Use sounddefentrylimit to limit the number of sound definition entries - and therefore the amount of wave data - loaded for each sound definition. This feature allows the programmer to implement a "low detail" setting at runtime without needing a seperate "low detail" set of assets.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventSystem::load

FMOD_EVENT_SYSTEMINFO

Structure containing realtime information about an event system.?

Structure

```
  typedef struct {
  int  numevents;
  int  eventmemory;
  int  numinstances;
  int  instancememory;
  int  dspmemory;
  int  numwavebanks;
   FMOD_EVENT_WAVEBANKINFO  * wavebankinfo;
  int  numplayingevents;
   FMOD_EVENT **  playingevents;
}  FMOD_EVENT_SYSTEMINFO;
```

Members

numevents

[out] Total number of events in all event groups in this event system.

eventmemory

[out] Amount of memory (in bytes) used by event hierarchy classes. DISABLED. Not working currently until further notice. Use FMOD::Memory_GetStats instead.

numinstances

[out] Total number of event instances in all event groups in this event system.

instancememory

[out] Amount of memory (in bytes) used by all event instances in this event system. DISABLED. Not working currently until further notice. Use FMOD::Memory_GetStats instead.

dspmemory

[out] Amount of memory (in bytes) used by event dsp networks. DISABLED. Not working currently until further notice. Use FMOD::Memory_GetStats instead.

numwavebanks

[out] Number of wave banks known to this event system.

wavebankinfo

[out] Array of detailed information on each wave bank.

numplayingevents

[in/out] On entry, maximum number of entries in playingevents array. On exit, actual number of entries in playingevents array, or if playingevents is null, then it is just the number of currently playing events. Optional.

playingevents

[in/out] Pointer to an array that will be filled with the event handles of all playing events. Optional. Specify 0 if not needed. Must be used in conjunction with numplayingevents.

Remarks

On entry, numplayingevents should be set to the number of elements in the playingevents array. If the actual number of playing events is greater than numplayingevents then the playingevents array will be filled with numplayingevents entries and numplayingevents will be set to the actual number of playing events on exit. In short, if numplayingevents on exit > numplayingevents on entry then the playingevents array wasn't large enough and some events were unable to be added to the array.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventSystem::getInfo
- FMOD_EVENT_WAVEBANKINFO

FMOD_EVENT_WAVEBANKINFO

Structure containing realtime information about a wavebank.?

Structure

```
typedef struct {
  char * name;
  int streamrefcnt;
  int samplerefcnt;
  int numstreams;
  int maxstreams;
  int streamsinuse;
  unsigned int streammemory;
  unsigned int samplememory;
} FMOD_EVENT_WAVEBANKINFO;
```

Members

name

[out] Name of this wave bank.

streamrefcnt

[out] Number of stream references to this wave bank made by events in this event system.

samplerefcnt

[out] Number of sample references to this wave bank made by events in this event system.

numstreams

[out] Number of times this wave bank has been opened for streaming.

maxstreams

[out] Maximum number of times this wave bank will be opened for streaming.

streamsinuse

[out] Number of streams currently in use.

streammemory

[out] Amount of memory (in bytes) used by streams.

samplememory

[out] Amount of memory (in bytes) used by samples.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventSystem::getInfo
- FMOD_EVENT_SYSTEMINFO

Defines

FMOD_EVENT_INITFLAGS

Initialization flags. Use them with EventSystem::init in the eventflags parameter to change various behaviour.?

Definition

```
  #define FMOD_EVENT_INIT_NORMAL 0x00000000
 #define FMOD_EVENT_INIT_USER_ASSETMANAGER 0x00000001
 #define FMOD_EVENT_INIT_FAIL_ON_MAXSTREAMS 0x00000002
 #define FMOD_EVENT_INIT_DONTUSENAMES 0x00000004
 #define FMOD_EVENT_INIT_UPPERCASE_FILENAMES 0x00000008
 #define FMOD_EVENT_INIT_SEARCH_PLUGINS 0x00000010
```

Values

FMOD_EVENT_INIT_NORMAL

All platforms - Initialize normally

FMOD_EVENT_INIT_USER_ASSETMANAGER

All platforms - All wave data loading/freeing will be referred back to the user through the event callback

FMOD_EVENT_INIT_FAIL_ON_MAXSTREAMS

All platforms - Events will fail if "Max streams" was reached when playing streamed banks, instead of going virtual.

FMOD_EVENT_INIT_DONTUSENAMES

All platforms - All event/eventgroup/eventparameter/eventcategory/eventreverb names will be discarded on load. Use getXXXByIndex to access them. This may potentially save a lot of memory at runtime.

FMOD_EVENT_INIT_UPPERCASE_FILENAMES

All platforms - All FSB filenames will be translated to upper case before being used.

FMOD_EVENT_INIT_SEARCH_PLUGINS

All platforms - Search the current directory for dsp/codec plugins on EventSystem::init.

Platforms Supported

Win32, Win64, Linux, Linux64, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3

See Also

- EventSystem::init

Version 4.12.03 Built on Feb 18, 2008

FMOD_EVENT_MODE

Event data loading bitfields. Bitwise OR them together for controlling how event data is loaded.?

Definition

```
  #define FMOD_EVENT_DEFAULT  0x00000000
 #define FMOD_EVENT_NONBLOCKING  0x00000001
 #define FMOD_EVENT_ERROR_ON_DISKACCESS  0x00000002
 #define FMOD_EVENT_INFOONLY  0x00000004
```

Values

FMOD_EVENT_DEFAULT

FMOD_EVENT_DEFAULT specifies default loading behaviour i.e. event data for the whole group is NOT cached and the function that initiated the loading process will block until loading is complete.

FMOD_EVENT_NONBLOCKING

For loading event data asynchronously. FMOD will use a thread to load the data. Use Event::getState to find out when loading is complete.

FMOD_EVENT_ERROR_ON_DISKACCESS

For EventGroup::getEvent / EventGroup::getEventByIndex. If EventGroup::loadEventData has accidently been forgotten this flag will return an FMOD_ERR_FILE_UNWANTED if the getEvent function tries to load data.

FMOD_EVENT_INFOONLY

For EventGroup::getEvent / EventGroup::getEventByIndex. Don't allocate instances or load data, just get a handle to allow user to get information from the event.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::loadEventData
- EventGroup::getEvent
- EventGroup::getEventByIndex

FMOD_EVENT_STATE

These values describe what state an event is in.?The flags below can be combined to set multiple states at once. Use bitwise AND operations to test for these.?An example of a combined flag set would be FMOD_EVENT_STATE_READY | FMOD_EVENT_STATE_PLAYING.?

Definition

```
  #define FMOD_EVENT_STATE_READY            0x00000001
  #define FMOD_EVENT_STATE_LOADING          0x00000002
  #define FMOD_EVENT_STATE_ERROR            0x00000004
  #define FMOD_EVENT_STATE_PLAYING          0x00000008
  #define FMOD_EVENT_STATE_CHANNELSACTIVE   0x00000010
  #define FMOD_EVENT_STATE_INFOONLY         0x00000020
  #define FMOD_EVENT_STATE_STARVING         0x00000040
```

Values

FMOD_EVENT_STATE_READY

Event is ready to play.

FMOD_EVENT_STATE_LOADING

Loading in progress.

FMOD_EVENT_STATE_ERROR

Failed to open - file not found, out of memory etc. See return value of Event::getState for what happened.

FMOD_EVENT_STATE_PLAYING

Event has been started. This will still be true even if there are no sounds active. Event::stop must be called or the event must stop itself using a 'one shot and stop event' parameter mode.

FMOD_EVENT_STATE_CHANNELSACTIVE

Event has active voices. Use this if you want to detect if sounds are playing in the event or not.

FMOD_EVENT_STATE_INFOONLY

Event was loaded with the FMOD_EVENT_INFOONLY flag.

FMOD_EVENT_STATE_STARVING

Event is streaming but not being fed data in time, so may be stuttering.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getState
- FMOD_EVENT_MODE

Enumerations

FMOD_EVENT_CALLBACKTYPE

These callback types are used with FMOD_EVENT_CALLBACK.?

Enumeration

```
  typedef enum {
    FMOD_EVENT_CALLBACKTYPE_SYNCPOINT,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_START,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_END,
    FMOD_EVENT_CALLBACKTYPE_STOLEN,
    FMOD_EVENT_CALLBACKTYPE_EVENTFINISHED,
    FMOD_EVENT_CALLBACKTYPE_NET_MODIFIED,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_CREATE,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_RELEASE,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_INFO,
    FMOD_EVENT_CALLBACKTYPE_EVENTSTARTED,
    FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_SELECTINDEX
}   FMOD_EVENT_CALLBACKTYPE;
```

Values

FMOD_EVENT_CALLBACKTYPE_SYNCPOINT

Called when a syncpoint is encountered. Can be from wav file markers.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_START

Called when a sound definition inside an event is triggered.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_END

Called when a sound definition inside an event ends or is stopped.

FMOD_EVENT_CALLBACKTYPE_STOLEN

Called when a getEventXXX call steals a playing event instance.

FMOD_EVENT_CALLBACKTYPE_EVENTFINISHED

Called when an event is stopped for any reason.

FMOD_EVENT_CALLBACKTYPE_NET_MODIFIED

Called when a property of the event has been modified by a network-connected host.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_CREATE

Called when a programmer sound definition entry is loaded.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_RELEASE

Called when a programmer sound definition entry is unloaded.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_INFO

Called when a sound definition entry is loaded.

FMOD_EVENT_CALLBACKTYPE_EVENTSTARTED

Called when an event is started.

FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_SELECTINDEX

Called when a sound definition entry needs to be chosen from a "ProgrammerSelected" sound definition.

Remarks

Note! Currently the user must call EventSystem::update for these callbacks to trigger!
An FMOD_EVENT_CALLBACKTYPE_SYNCPOINT callback is generated from 'markers' embedded in .wav files. These can be created by placing 'markers' in the original source wavs using a tool such as Sound Forge or Cooledit.
The wavs are then compiled into .FSB files when compiling the audio data using the FMOD designer tool.
Callbacks will be automatically generated at the correct place in the timeline when these markers are encountered which makes it useful for synchronization, lip syncing etc.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_START callback is generated each time a sound definition is played in an event.
This happens every time a sound definition starts due to the event parameter entering the region specified in the layer created by the sound designer..
This also happens when sounds are randomly respawned using the random respawn feature in the sound definition properties in FMOD designer.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_END callback is generated when a one-shot sound definition inside an event ends, or when a looping sound definition stops due to the event parameter leaving the region specified in the layer created by the sound designer.

An FMOD_EVENT_CALLBACKTYPE_STOLEN callback is generated when a getEventXXX call needs to steal a playing event because the event's "Max playbacks" has been exceeded. This callback is called before the event is stolen and before the event is stopped. An FMOD_EVENT_CALLBACKTYPE_EVENTFINISHED callback will be generated when the stolen event is stopped i.e. **after** the FMOD_EVENT_CALLBACKTYPE_STOLEN. If the callback function returns FMOD_ERR_EVENT_FAILED, the event will **not** be stolen, and the returned value will be passed back as the return value of the getEventXXX call that triggered the steal attempt.

An FMOD_EVENT_CALLBACKTYPE_EVENTFINISHED callback is generated whenever an event is stopped for any reason including when the user calls Event::stop().

An FMOD_EVENT_CALLBACKTYPE_NET_MODIFIED callback is generated when someone has connected to your running application with FMOD Designer and changed a property within this event, for example volume or pitch.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_CREATE callback is generated when a "programmer" sound needs to be loaded.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_RELEASE callback is generated when a "programmer" sound needs to be unloaded.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_INFO callback is generated when a sound definition is loaded. It can be used to find information about the specific sound that will be played.

An FMOD_EVENT_CALLBACKTYPE_EVENTSTARTED callback is generated whenever an event is started. This callback will be called before any sounds in the event have begun to play.

An FMOD_EVENT_CALLBACKTYPE_SOUNDDEF_SELECTINDEX callback is generated when a sound definition entry needs to be chosen from a "ProgrammerSelected" sound definition.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii, Wii

See Also

- Event::setCallback
- FMOD_EVENT_CALLBACK
- EventSystem::update

FMOD_EVENT_PITCHUNITS

Pitch units for Event::setPitch and EventCategory::setPitch.?

Enumeration

```
  typedef enum {
    FMOD_EVENT_PITCHUNITS_RAW,
    FMOD_EVENT_PITCHUNITS_OCTAVES,
    FMOD_EVENT_PITCHUNITS_SEMITONES,
    FMOD_EVENT_PITCHUNITS_TONES
}   FMOD_EVENT_PITCHUNITS;
```

Values

FMOD_EVENT_PITCHUNITS_RAW

Pitch is specified in raw underlying units.

FMOD_EVENT_PITCHUNITS_OCTAVES

Pitch is specified in units of octaves.

FMOD_EVENT_PITCHUNITS_SEMITONES

Pitch is specified in units of semitones.

FMOD_EVENT_PITCHUNITS_TONES

Pitch is specified in units of tones.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::setPitch
- EventCategory::setPitch

FMOD_EVENT_PROPERTY

Property indices for Event::getPropertyByIndex.?

Enumeration

```
  typedef enum {
    FMOD_EVENTPROPERTY_NAME,
    FMOD_EVENTPROPERTY_VOLUME,
    FMOD_EVENTPROPERTY_VOLUMERANDOMIZATION,
    FMOD_EVENTPROPERTY_PITCH,
    FMOD_EVENTPROPERTY_PITCH_OCTAVES,
    FMOD_EVENTPROPERTY_PITCH_SEMITONES,
    FMOD_EVENTPROPERTY_PITCH_TONES,
    FMOD_EVENTPROPERTY_PITCHRANDOMIZATION,
    FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_OCTAVES,
    FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_SEMITONES,
    FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_TONES,
    FMOD_EVENTPROPERTY_PRIORITY,
    FMOD_EVENTPROPERTY_MAX_PLAYBACKS,
    FMOD_EVENTPROPERTY_MAX_PLAYBACKS_BEHAVIOR,
    FMOD_EVENTPROPERTY_MODE,
    FMOD_EVENTPROPERTY_3D_ROLLOFF,
    FMOD_EVENTPROPERTY_3D_MINDISTANCE,
    FMOD_EVENTPROPERTY_3D_MAXDISTANCE,
    FMOD_EVENTPROPERTY_3D_POSITION,
    FMOD_EVENTPROPERTY_3D_CONEINSIDEANGLE,
    FMOD_EVENTPROPERTY_3D_CONEOUTSIDEANGLE,
    FMOD_EVENTPROPERTY_3D_CONEOUTSIDEVOLUME,
    FMOD_EVENTPROPERTY_3D_DOPPLERSCALE,
    FMOD_EVENTPROPERTY_3D_SPEAKERSPREAD,
    FMOD_EVENTPROPERTY_3D_PANLEVEL,
    FMOD_EVENTPROPERTY_SPEAKER_L,
    FMOD_EVENTPROPERTY_SPEAKER_C,
    FMOD_EVENTPROPERTY_SPEAKER_R,
    FMOD_EVENTPROPERTY_SPEAKER_LS,
    FMOD_EVENTPROPERTY_SPEAKER_RS,
    FMOD_EVENTPROPERTY_SPEAKER_LR,
    FMOD_EVENTPROPERTY_SPEAKER_RR,
    FMOD_EVENTPROPERTY_SPEAKER_LFE,
    FMOD_EVENTPROPERTY_REVERBWETLEVEL,
    FMOD_EVENTPROPERTY_ONESHOT,
    FMOD_EVENTPROPERTY_FADEIN,
    FMOD_EVENTPROPERTY_FADEOUT,
    FMOD_EVENTPROPERTY_REVERBDRYLEVEL,
    FMOD_EVENTPROPERTY_TIMEOFFSET,
    FMOD_EVENTPROPERTY_SPAWNINTENSITY,
    FMOD_EVENTPROPERTY_SPAWNINTENSITY_RANDOMIZATION,
    FMOD_EVENTPROPERTY_WII_CONTROLLERSPEAKER,
    FMOD_EVENTPROPERTY_3D_POSRANDOMIZATION,
    FMOD_EVENTPROPERTY_USER_BASE
} FMOD_EVENT_PROPERTY;
```

Values

FMOD_EVENTPROPERTY_NAME

Type : char * - Name of event.

FMOD_EVENTPROPERTY_VOLUME

Type : float - Relative volume of event.

FMOD_EVENTPROPERTY_VOLUMERANDOMIZATION

Type : float - Random deviation in volume of event.

FMOD_EVENTPROPERTY_PITCH

Type : float - Relative pitch of event in raw underlying units.

FMOD_EVENTPROPERTY_PITCH_OCTAVES

Type : float - Relative pitch of event in octaves.

FMOD_EVENTPROPERTY_PITCH_SEMITONES

Type : float - Relative pitch of event in semitones.

FMOD_EVENTPROPERTY_PITCH_TONES

Type : float - Relative pitch of event in tones.

FMOD_EVENTPROPERTY_PITCHRANDOMIZATION

Type : float - Random deviation in pitch of event in raw underlying units.

FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_OCTAVES

Type : float - Random deviation in pitch of event in octaves.

FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_SEMITONES

Type : float - Random deviation in pitch of event in semitones.

FMOD_EVENTPROPERTY_PITCHRANDOMIZATION_TONES

Type : float - Random deviation in pitch of event in tones.

FMOD_EVENTPROPERTY_PRIORITY

Type : int - Playback priority of event.

FMOD_EVENTPROPERTY_MAX_PLAYBACKS

Type : int - Maximum simultaneous playbacks of event.

FMOD_EVENTPROPERTY_MAX_PLAYBACKS_BEHAVIOR

Type : int - 1 = steal oldest, 2 = steal newest, 3 = steal quietest, 4 = just fail, 5 = just fail if quietest.

FMOD_EVENTPROPERTY_MODE

Type : FMOD_MODE - Either FMOD_3D or FMOD_2D.

FMOD_EVENTPROPERTY_3D_ROLLOFF

Type : FMOD_MODE - Either FMOD_3D_LOGROLLOFF, FMOD_3D_LINEARROLLOFF, or none for custom rolloff.

FMOD_EVENTPROPERTY_3D_MINDISTANCE

Type : float - Minimum 3d distance of event.

FMOD_EVENTPROPERTY_3D_MAXDISTANCE

Type : float - Maximum 3d distance of event. Means different things depending on EVENTPROPERTY_3D_ROLLOFF. If event has custom rolloff, setting FMOD_EVENTPROPERTY_3D_MAXDISTANCE will scale the range of all distance parameters in this event e.g. set this property to 2.0 to double the range of all distance parameters, set it to 0.5 to halve the range of all distance parameters.

FMOD_EVENTPROPERTY_3D_POSITION

Type : FMOD_MODE - Either FMOD_3D_HEADRELATIVE or FMOD_3D_WORLDRELATIVE.

FMOD_EVENTPROPERTY_3D_CONEINSIDEANGLE

Type : float - Event cone inside angle. 0 to 360.

FMOD_EVENTPROPERTY_3D_CONEOUTSIDEANGLE

Type : float - Event cone outside angle. 0 to 360.

FMOD_EVENTPROPERTY_3D_CONEOUTSIDEVOLUME

Type : float - Event cone outside volume. 0 to 1.0.

FMOD_EVENTPROPERTY_3D_DOPPLERSCALE

Type : float - Doppler scale where 0 = no doppler, 1.0 = normal doppler, 2.0 = double doppler etc.

FMOD_EVENTPROPERTY_3D_SPEAKERSPREAD

Type : float - Angle of spread for stereo/mutlichannel source. 0 to 360.

FMOD_EVENTPROPERTY_3D_PANLEVEL

Type : float - 0 = sound pans according to speaker levels, 1 = sound pans according to 3D position.

FMOD_EVENTPROPERTY_SPEAKER_L

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_C

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_R

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_LS

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_RS

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_LR

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_RR

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_SPEAKER_LFE

Type : float - 2D event volume for front left speaker.

FMOD_EVENTPROPERTY_REVERBWETLEVEL

Type : float - Reverb gain for this event where 0 = full reverb, -60 = no reverb.

FMOD_EVENTPROPERTY_ONESHOT

Type : int - Oneshot event - stops when no channels playing

FMOD_EVENTPROPERTY_FADEIN

Type : int - Time in milliseconds over which to fade this event in when programmer starts it. 0 = no fade in.

FMOD_EVENTPROPERTY_FADEOUT

Type : int - Time in milliseconds over which to fade this event out when programmer stops it. 0 = no fade out.

FMOD_EVENTPROPERTY_REVERBDRYLEVEL

Type : float - Dry reverb gain for this event where 0 = full dry, -60 = no dry.

FMOD_EVENTPROPERTY_TIMEOFFSET

Type : float - Time offset of sound start in seconds (0 to 60.0f)

FMOD_EVENTPROPERTY_SPAWNINTENSITY

Type : float - Multiplier for spawn frequency of all sounds in this event.

FMOD_EVENTPROPERTY_SPAWNINTENSITY_RANDOMIZATION

Type : float - Random deviation in spawn intensity of event.

FMOD_EVENTPROPERTY_WII_CONTROLLERSPEAKER

Type : int - Wii only. Use 0 to 3 to specify a Wii controller speaker to play this event on, -1 to play on normal Wii speakers.

FMOD_EVENTPROPERTY_3D_POSRANDOMIZATION

Type : unsigned int - Radius of random deviation in the 3D position of event.

FMOD_EVENTPROPERTY_USER_BASE

User created events start from here onwards.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- Event::getPropertyByIndex

FMOD_EVENT_RESOURCE

Flags to pass to EventGroup::loadEventData to determine what to load at the time of calling.?

Enumeration

```
typedef enum {
  FMOD_EVENT_RESOURCE_STREAMS_AND_SAMPLES,
  FMOD_EVENT_RESOURCE_STREAMS,
  FMOD_EVENT_RESOURCE_SAMPLES
} FMOD_EVENT_RESOURCE;
```

Values

FMOD_EVENT_RESOURCE_STREAMS_AND_SAMPLES

Open all streams and load all banks into memory, under this group (recursive)

FMOD_EVENT_RESOURCE_STREAMS

Open all streams under this group (recursive). No samples are loaded.

FMOD_EVENT_RESOURCE_SAMPLES

Load all banks into memory, under this group (recursive). No streams are opened.

Platforms Supported

Win32, Win64, Linux, Macintosh, Xbox, Xbox360, PlayStation 2, GameCube, PlayStation Portable, PlayStation 3, Wii

See Also

- EventGroup::loadEventData

Firelight Technologies FMOD Ex

C++ Reference

Functions

NetEventSystem_GetVersion

Get the NetEventSystem version number.?

Syntax

```
  FMOD_RESULT NetEventSystem_GetVersion(
  unsigned int *  version
);
```

Parameters

version

A pointer to an integer to receive the version number

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Platforms Supported

See Also

- NetEventSystem_Init

NetEventSystem_Init

This function initializes the NetEventSystem and prepares it to accept incoming connections.?NOTE: This function must be called before any other NetEventSystem functions.?

Syntax

```
  FMOD_RESULT NetEventSystem_Init(
  EventSystem * eventsystem,
  unsigned short port
);
```

Parameters

eventsystem

A pointer to a user-created EventSystem object.

port

The TCP port that the NetEventSystem will use to accept incoming connections. 0 = use default port which is 17997.

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Specify 0 for the port unless you have a good reason not to. Make sure that whatever port you specify is not blocked.

Platforms Supported

See Also

- NetEventSystem_Update
- NetEventSystem_Shutdown
- NetEventSystem_GetVersion

NetEventSystem_Shutdown

Shut down the NetEventSystem.?

Syntax

```
FMOD_RESULT NetEventSystem_Shutdown();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

Call this function after you call EventSystem::release.

Platforms Supported

See Also

- NetEventSystem_Init

NetEventSystem_Update

Update the NetEventSystem.?

Syntax

```
FMOD_RESULT NetEventSystem_Update();
```

Parameters

Return Values

If the function succeeds then the return value is FMOD_OK.
If the function fails then the return value will be one of the values defined in the FMOD_RESULT enumeration.

Remarks

You must call this function once a frame just after you call EventSystem::update.

Platforms Supported

See Also

- NetEventSystem_Init
- NetEventSystem_Shutdown