FMOD

Le but de ce TD est de vous familiariser avec l'utilisation de FMOD..

Instructions :

1) Si ce n'est pas déjà fait, installez le FMOD EX Programmers API disponible ici : http://www.fmod.org/index.php/download

2) Configurez Visual pour utiliser les répertoires include(api \ inc) et lib (api \ lib) de FMOD et copiez les DLL du répertoires api dans le répertoire de votre éxécutable.

3) Allez dans le répertoire exemple de FMOD pour ouvrir la solution des sample. Etudiez les samples suivants :

   -Playsound → main.cpp
   -PlayStream → main.cpp
   -Cdplayer → main.cpp

4) A présent, Ouvrez SoundManager.uml : Votre travail sera de réaliser les classes Sound et Stream pour commencer.

5) A l'aide des sons qui vous sont fournis, ajoutez le son dans votre scène :

   -la musique demarre dès le début, il est possible de la mettre en pause ou de la stopper.
   -l'alarme sonne tant que la porte est ouverte.
   -A chaque transition de state de la porte correspond un son a lire, une seule fois et non mélangé avec le précédent.

N'oubliez pas d'initialiser le FMOD::system dans votre Game.cpp

6) A présent, réalisez la classe SoundManager présente dans l'UML. Vous ne devez plus faire appel qu'à ce singleton pour gérer le son dans votre Game,cpp

7) Pour allez plus loin, Ouvrez le sample 3D. A présent en fonction des déplacement de la porte, déplacez le listener afin que le son corresponde à un déplacement type FPS.

Informations :

- Il faut toujours commencer par vérifier la version de FMOD.
- Pour le TD, vous pouvez vous passer de l' « errcheck »

Documentation :

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.

C++ Syntax

```
FMOD_RESULT System_Create(
  FMOD::System **  system
);
```

C Syntax

```
FMOD_RESULT FMOD_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.

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.

C++ Syntax

```
FMOD_RESULT System::init(
  int  maxchannels,
  FMOD_INITFLAGS  flags,
  void *  extradriverdata
);
```

C Syntax

```
FMOD_RESULT FMOD_System_Init(
```

```
  FMOD_SYSTEM *  system,
  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.

System::release

Closes and frees a system object and its resources.

C++ Syntax

```
FMOD_RESULT System::release();
```

C Syntax

```
FMOD_RESULT FMOD_System_Release(FMOD_SYSTEM *  system);
```

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.

System::getVersion

Returns the current version of FMOD Ex being used.

C++ Syntax

```
FMOD_RESULT System::getVersion(
  unsigned int *  version
);
```

C Syntax

```
FMOD_RESULT FMOD_System_GetVersion(
  FMOD_SYSTEM *  system,
  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.

System::createSound

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

C++ Syntax

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

C Syntax

```
FMOD_RESULT FMOD_System_CreateSound(
  FMOD_SYSTEM *  system,
  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 a pointer to a preloaded sound 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.

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.

C++ Syntax

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

C Syntax

```
FMOD_RESULT FMOD_System_CreateStream(
  FMOD_SYSTEM *  system,
  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.

System::set3DSettings

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

C++ Syntax

```
FMOD_RESULT System::set3DSettings(
  float  dopplerscale,
  float  distancefactor,
  float  rolloffscale
);
```

C Syntax

```
FMOD_RESULT FMOD_System_Set3DSettings(
  FMOD_SYSTEM *  system,
  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.

Note! "rolloffscale" has no effect when using FMOD_3D_LINEARROLLOFF or FMOD_3D_CUSTOMROLLOFF.

System::set3DListenerAttributes

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

C++ Syntax

```
FMOD_RESULT System::set3DListenerAttributes(
  int  listener,
  const FMOD_VECTOR *  pos,
  const FMOD_VECTOR *  vel,
  const FMOD_VECTOR *  forward,
  const FMOD_VECTOR *  up
);
```

C Syntax

```
FMOD_RESULT FMOD_System_Set3DListenerAttributes(
  FMOD_SYSTEM *  system,
  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.

System::setSpeakerMode

Sets the speaker mode in the hardware and FMOD software mixing engine.

C++ Syntax

```
FMOD_RESULT System::setSpeakerMode(
  FMOD_SPEAKERMODE  speakermode
);
```

C Syntax

```
FMOD_RESULT FMOD_System_SetSpeakerMode(
  FMOD_SYSTEM *  system,
  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.

System::update

Updates the FMOD system. This should be called once per 'game' tick, or once per frame in your application.

C++ Syntax

```
FMOD_RESULT System::update();
```

C Syntax

```
FMOD_RESULT FMOD_System_Update(FMOD_SYSTEM *  system);
```

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. System::update must be called to get 3D positioning.
- Virtual voices. If more voices are played than there are real hardware/software voices, System::update must be called to handle the virtualization.
- *_NRT output modes. System::update must be called to drive the output for these output modes.
- FMOD_INIT_STREAM_FROM_UPDATE. System::update must be called to update the streamer if this flag has been used.
- Callbacks. System::update must be called to fire callbacks if they are specified.
- FMOD_NONBLOCKING. System::update must be called to make sounds opened with FMOD_NONBLOCKING flag to work properly.
- FMOD_SYSTEM_CALLBACKTYPE_DEVICELISTCHANGED callback. System::update must be called for this callback to trigger.
  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.

System::playSound

Plays a sound object on a particular channel.

C++ Syntax

```
FMOD_RESULT System::playSound(
  FMOD_CHANNELINDEX  channelid,
  FMOD::Sound *  sound,
  bool  paused,
  FMOD::Channel **  channel
);
```

C Syntax

```
FMOD_RESULT FMOD_System_PlaySound(
  FMOD_SYSTEM *  system,
  FMOD_CHANNELINDEX  channelid,
  FMOD_SOUND *  sound,
  FMOD_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.

Sound::getLength

Retrieves the length of the sound using the specified time unit.

C++ Syntax

```
FMOD_RESULT Sound::getLength(
  unsigned int *  length,
  FMOD_TIMEUNIT  lengthtype
);
```

C Syntax

```
FMOD_RESULT FMOD_Sound_GetLength(
  FMOD_SOUND *  sound,
  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.

Sound::getName

Retrieves the name of a sound.

C++ Syntax

```
FMOD_RESULT Sound::getName(
  char *  name,
  int  namelen
);
```

C Syntax

```
FMOD_RESULT FMOD_Sound_GetName(
  FMOD_SOUND *  sound,
  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.

Channel::getPaused

Retrieves the paused state of the channel.

C++ Syntax

```
FMOD_RESULT Channel::getPaused(
  bool *  paused
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_GetPaused(
  FMOD_CHANNEL *  channel,
  FMOD_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.

Channel::setPaused

Sets the paused state of the channel.

C++ Syntax

```
FMOD_RESULT Channel::setPaused(
  bool  paused
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_SetPaused(
  FMOD_CHANNEL *  channel,
  FMOD_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.

Channel::getPosition

Returns the current PCM offset or playback position for the specified channel.

C++ Syntax

```
FMOD_RESULT Channel::getPosition(
  unsigned int *  position,
  FMOD_TIMEUNIT  postype
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_GetPosition(
  FMOD_CHANNEL *  channel,
  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.

Channel::setPosition

Sets the current playback position for the currently playing sound to the specified PCM offset.

C++ Syntax

```
FMOD_RESULT Channel::setPosition(
  unsigned int  position,
  FMOD_TIMEUNIT  postype
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_SetPosition(
  FMOD_CHANNEL *  channel,
  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.

Channel::setVolume

Sets the volume for the channel linearly.

C++ Syntax

```
FMOD_RESULT Channel::setVolume(
  float  volume
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_SetVolume(
  FMOD_CHANNEL *  channel,
  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.

Channel::stop

Stops the channel from playing. Makes it available for re-use by the priority system.

C++ Syntax

```
FMOD_RESULT Channel::stop();
```

C Syntax

```
FMOD_RESULT FMOD_Channel_Stop(FMOD_CHANNEL *  channel);
```

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.

Channel::set3DMinMaxDistance

Sets the minimum and maximum audible distance for a channel.

C++ Syntax

```
FMOD_RESULT Channel::set3DMinMaxDistance(
  float  mindistance,
  float  maxdistance
);
```

C Syntax

```
FMOD_RESULT FMOD_Channel_Set3DMinMaxDistance(
  FMOD_CHANNEL *  channel,
  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.