

AcousticsLib

1.00 Alpha

Generated by Doxygen 1.8.11

Contents

1	AcousticsLib 1.00 Alpha Documentation	1
1.1	Table of Contents	1
1.2	Introduction	1
1.3	Getting Started	2
1.4	Example 1: Play a sound	2
1.5	Example 2: Read wave buffers	2
1.6	Example 3: Music streaming	3
1.7	Example 4: 3D Sound	3
1.8	Example 5: Recording	4
2	Class Index	7
2.1	Class List	7
3	Class Documentation	9
3.1	Ac::AudioStream Class Reference	9
3.1.1	Detailed Description	9
3.1.2	Member Function Documentation	9
3.1.2.1	Seek(double timePoint)=0	9
3.1.2.2	StreamWaveBuffer(WaveBuffer &buffer)=0	10
3.2	Ac::AudioSystem Class Reference	10
3.2.1	Detailed Description	11
3.2.2	Member Function Documentation	12
3.2.2.1	DetermineAudioFormat(std::istream &stream)	12
3.2.2.2	Load(const std::string &moduleName)	13

3.2.2.3	LoadSound(const std::string &filename, const SoundFlags::BitMask flags=0)	13
3.2.2.4	OpenAudioStream(const std::string &filename)	14
3.2.2.5	OpenAudioStream(std::unique_ptr< std::istream > &&stream)	14
3.2.2.6	Play(const std::string &filename, float volume=1.0f, float pitch=1.0f)	15
3.2.2.7	ReadWaveBuffer(const std::string &filename)	15
3.2.2.8	ReadWaveBuffer(std::istream &stream)	15
3.2.2.9	Streaming(Sound &sound, WaveBuffer &waveBuffer)	15
3.2.2.10	Streaming(Sound &sound)	16
3.2.2.11	WriteAudioBuffer(const AudioFormats format, std::ostream &stream, const WaveBuffer &waveBuffer)	16
3.3	Ac::ChannelTypes2 Struct Reference	16
3.4	Ac::ChannelTypes3 Struct Reference	17
3.5	Ac::ChannelTypes4 Struct Reference	17
3.6	Ac::ChannelTypes5 Struct Reference	17
3.7	Ac::ChannelTypes5_1 Struct Reference	17
3.7.1	Member Enumeration Documentation	18
3.7.1.1	anonymous enum	18
3.8	Ac::ChannelTypes6_1 Struct Reference	18
3.8.1	Member Enumeration Documentation	18
3.8.1.1	anonymous enum	18
3.9	Ac::ChannelTypes7_1 Struct Reference	18
3.9.1	Member Enumeration Documentation	18
3.9.1.1	anonymous enum	18
3.10	Ac::ListenerOrientation Struct Reference	19
3.10.1	Detailed Description	19
3.11	Ac::Microphone Class Reference	19
3.11.1	Detailed Description	20
3.11.2	Member Function Documentation	20
3.11.2.1	IsRecording() const =0	20
3.11.2.2	ReceivedInput()=0	20

3.11.2.3 Start(const WaveBufferFormat &waveFormat, std::size_t sampleFrames, std::size_t deviceIndex=standardDeviceIndex)=0	20
3.11.2.4 Start(const WaveBufferFormat &waveFormat, double duration, std::size_t deviceIndex=standardDeviceIndex)=0	21
3.12 Ac::MicrophoneDevice Struct Reference	21
3.12.1 Detailed Description	22
3.13 Ac::Renderer Class Reference	22
3.13.1 Detailed Description	22
3.14 Ac::Sound Class Reference	22
3.14.1 Detailed Description	24
3.14.2 Member Function Documentation	24
3.14.2.1 AttachBuffer(const WaveBuffer &waveBuffer)=0	24
3.14.2.2 Enable3D(bool enable=true)=0	24
3.14.2.3 GetProcessedQueueSize() const =0	25
3.14.2.4 GetQueueSize() const =0	25
3.14.2.5 QueueBuffer(const WaveBuffer &waveBuffer)=0	25
3.14.2.6 SetLooping(bool enable)=0	25
3.14.2.7 SetStreamSource(const std::shared_ptr< AudioStream > &streamSource)	26
3.15 Ac::SoundFlags Struct Reference	26
3.15.1 Detailed Description	26
3.15.2 Member Enumeration Documentation	26
3.15.2.1 anonymous enum	26
3.16 Ac::WaveBuffer Class Reference	27
3.16.1 Detailed Description	28
3.16.2 Member Function Documentation	28
3.16.2.1 Append(const WaveBuffer &other)	28
3.16.2.2 ForEachSample(const SampleIterationFunction &iterator, std::size_t indexBegin, std::size_t indexEnd)	29
3.16.2.3 ForEachSample(const SampleIterationFunction &iterator, double timeBegin, double timeEnd)	29
3.16.2.4 ForEachSample(const SampleIterationFunction &iterator)	29
3.16.2.5 ForEachSample(const SampleConstIterationFunction &iterator, std::size_t indexBegin, std::size_t indexEnd) const	30

3.16.2.6 ForEachSample(const SampleConstIterationFunction &iterator, double timeBegin, double timeEnd) const	30
3.16.2.7 ForEachSample(const SampleConstIterationFunction &iterator) const	30
3.16.2.8 GetIndexFromTimePoint(double timePoint) const	31
3.16.2.9 GetTimePointFromIndex(std::size_t index) const	31
3.16.2.10 ReadSample(std::size_t index, unsigned short channel) const	31
3.16.2.11 SetChannels(unsigned short channels)	32
3.16.2.12 SetFormat(const WaveBufferFormat &format)	32
3.16.2.13 SwapEndianness()	32
3.17 Ac::WaveBufferFormat Struct Reference	32
3.17.1 Detailed Description	33
3.17.2 Member Data Documentation	33
3.17.2.1 sampleRate	33
Index	35

Chapter 1

AcousticsLib 1.00 Alpha Documentation

1.1 Table of Contents

- [Introduction](#)
- [Getting Started](#)
- [Example 1: Play a sound](#)
- [Example 2: Read wave buffers](#)
- [Example 3: Music streaming](#)
- [Example 4: 3D Sound](#)
- [Example 5: Recording](#)

1.2 Introduction

Welcome to AcousticsLib, a cross-platform and open-source C++ library for real-time audio processing.

The following platforms are supported:

- **Windows Vista/ 7/ 8/ 10**
- **Mac OS X**
- **Linux**

The following audio file formats are supported:

- **WAV** (*Waveform Audio File Format*) Read/Write
- **AIFF** (*Audio Interchange File Format*) Read
- **OGG** (*Ogg-Vorbis*) Read

The following audio engines are provided:

- **OpenAL**

1.3 Getting Started

To start using AcousticsLib in your C++ project you need the following prerequisites:

- C++11 compliant compiler (i.e. **Visual C++ 2015** or later, **G++** or **clang** with `-std=c++11` option available)
- **GaussianLib** header files

To build the AcousticsLib you will also need the following prerequisites:

- **CMake** 2.8 or later to build the project files (e.g. for VisualStudio, Xcode, Code::Blocks etc.)
- **OpenAL** SDK
- **Ogg Vorbis** source files of **libogg** and **libvorbis**

After setting up all prerequisites, set the **include search path** to `<AcousticsLib-Path>/include` and add the library file `AcLib` (`AcLib.lib` for Visual C++ and `libAcLib.a` for G++ and clang) to the dependencies in your project.

1.4 Example 1: Play a sound

This is an example of playing a sound in the simplest way:

```
#include <Ac/AcLib.h>
#include <iostream>

int main()
{
    // Load audio system and play a sound from file.
    auto audioSystem = Ac::AudioSystem::Load();
    audioSystem->Play("MySound.wav");

    // Wait for user input before quit
    int i = 0;
    std::cin >> i;

    return 0;
}
```

1.5 Example 2: Read wave buffers

This is an example about reading wave buffers and creating sounds manually:

```
#include <Ac/AcLib.h>
#include <iostream>
#include <thread>
#include <chrono>

int main()
{
    try
    {
        // Load specific audio system. To find all available system,
        // use "Ac::AudioSystem::FindModules()". This must be an std::shared_ptr,
        // because the audio system keeps track of this reference with an std::weak_ptr.
        // If a new audio system is loaded, all references must be reset,
        // since only a single audio system can be loaded at a time.
        std::shared_ptr<Ac::AudioSystem> audioSystem = Ac::AudioSystem::Load("OpenAL");
    }
}
```



```

// Read wave buffer from file. The return type is "Ac::WaveBuffer".
Ac::WaveBuffer waveBuffer = audioSystem->ReadWaveBuffer("MySound.wav");

// Add some noise to the wave buffer with an amplitude of 0.1.
// The "NoiseGenerator" function will return a function object,
// which is applied to each sample.
waveBuffer.ForEachSample(Ac::Synthesizer::NoiseGenerator(0.1));

// Blur the buffer to make is sound like a robot voice.
// Time spread is 0.2, gaussian curve variance is 1.0 (standard deviation),
// and use 100 samples to compute the blur.
Ac::Synthesizer::BlurWaveBuffer(waveBuffer, 0.2, 1.0, 100);

// Create a sound with our wave buffer.
std::unique_ptr<Ac::Sound> sound = audioSystem->CreateSound(waveBuffer);

// Play the sound with a pitch (or frequency factor) of 70% and volume of 80%.
sound->SetPitch(0.7f);
sound->SetVolume(0.8f);
sound->Play();

// Wait while the sound is playing
while (sound->IsPlaying())
{
    // Sleep for 100 milliseconds to let other processes run
    // (playing the sound is done in the background).
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
}
catch (const std::exception& e)
{
    // Print out exception message if something went wrong
    // (e.g. sound file is corrupted or the like).
    std::cerr << e.what() << std::endl;
}
return 0;
}

```

1.6 Example 3: Music streaming

This is a small example about music streaming:

```

#include <Ac/AcLib.h>

int main()
{
    auto audioSystem = Ac::AudioSystem::Load();

    // Load and play Ogg-Vorbis music stream from file.
    auto sound = audioSystem->LoadSound("MyMusic.ogg");
    sound->Play();

    // Process music streaming
    // (this must currently be done manually with the "Streaming" function).
    while (sound->IsPlaying())
    {
        // Process next streaming block. This function automatically
        // checks if new streaming buffers can be queued.
        audioSystem->Streaming(*sound);
    }

    return 0;
}

```

1.7 Example 4: 3D Sound

This is a small example about music streaming:

```

#include <Ac/AcLib.h>
#include <thread>
#include <chrono>

int main()
{
    auto audioSystem = Ac::AudioSystem::Load();

    // Load sound from file and enable 3D features.
    auto sound = audioSystem->LoadSound("MySound.wav", Ac::SoundFlags::Enable3D);

    // Setup 3D position in the left-handed coordinate system.
    sound->SetPosition({ 5, 0, 2 });
    sound->Play();

    while (sound->IsPlaying())
    {
        // Move sound in 3D space.
        Gs::Vector3 deltaPos(-0.1f, 0, 0);
        sound->SetPosition(sound->GetPosition() + deltaPos);

        // Set sound velocity for the doppler effect.
        sound->SetVelocity(deltaPos);

        // Wait for a moment
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    return 0;
}

```

1.8 Example 5: Recording

This is an example about recording with a microphone device (currently only supported on the *Windows* platform):

```

#include <Ac/AcLib.h>
#include <chrono>

int main()
{
    try
    {
        auto audioSystem = Ac::AudioSystem::Load();

        // Query a microphone object and use standard device.
        std::unique_ptr<Ac::Microphone> mic = audioSystem->QueryMicrophone();

        if (mic)
        {
            // Start recording process with the following wave buffer format:
            // 44.1 kHz sample rate, 16 bits per sample, 1 channel, and sample every 0.1 seconds.
            mic->Start(Ac::WaveBufferFormat(Ac::sampleRate44kHz, 16, 1), 0.1);

            // Create an empty sound to play immediately what we've recorded.
            auto sound = audioSystem->CreateSound();

            // Record for a specific amount of time.
            auto startTime = std::chrono::system_clock::now();

            while (mic->IsRecording())
            {
                // Receiver wave buffer from microphone.
                std::unique_ptr<Ac::WaveBuffer> buffer = mic->ReceivedInput();
                if (buffer)
                {
                    // Print some information about the received buffer.
                    std::cout
                        << "Received Buffer: Sample Frames = " << buffer->GetSampleFrames() << ", "
                        << "Duration = " << buffer->GetTotalTime() << "s, "
                        << "Queue Size = " << sound->GetQueueSize() << std::endl;

                    // Modify the input buffer before adding it to the sound queue.
                    Ac::Synthesizer::BlurWaveBuffer(*buffer, 0.1, 1.0, 15);

                    // Queue the new buffer to our sound.
                    sound->QueueBuffer(*buffer);

                    // Start playing the sound when we have enough buffers in the queue.
                    if (sound->GetQueueSize() == 2)
                        sound->Play();
                }
            }
        }
    }
}

```

```
    }

    // Check if time for recording test is over (5 seconds)
    auto now = std::chrono::system_clock::now();
    if (std::chrono::duration_cast<std::chrono::seconds>(now - startTime).count() > 5)
    {
        // Stop recording process.
        mic->Stop();
    }
}

else
    std::cerr << "No microphone device available" << std::endl;
}
catch (const std::exception& e)
{
    std::cerr << e.what() << std::endl;
}
}
```


Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Ac::AudioStream	
Audio stream interface	9
Ac::AudioSystem	
Audio system interface. All coordinates or 3D sounds are meant to be in a left-handed coordinates system, i.e. positive Z values point into your monitor, and negative Z values point out of your monitor	10
Ac::ChannelTypes2	16
Ac::ChannelTypes3	17
Ac::ChannelTypes4	17
Ac::ChannelTypes5	17
Ac::ChannelTypes5_1	17
Ac::ChannelTypes6_1	18
Ac::ChannelTypes7_1	18
Ac::ListenerOrientation	
Structure for the 3D listener orientation with at- and up- vectors	19
Ac::Microphone	
Microphone interface	19
Ac::MicrophoneDevice	
Microphone device descriptor structure	21
Ac::Renderer	
Renderer interface used in conjunction with the Visualizer	22
Ac::Sound	
Sound source interface	22
Ac::SoundFlags	
Loading sound flags enumeration	26
Ac::WaveBuffer	
Data model for an audio wave buffer	27
Ac::WaveBufferFormat	
Wave buffer format descriptor structure	32

Chapter 3

Class Documentation

3.1 Ac::AudioStream Class Reference

Audio stream interface.

```
#include <AudioStream.h>
```

Public Member Functions

- virtual std::size_t [StreamWaveBuffer](#) ([WaveBuffer](#) &buffer)=0
Reads the audio data from the active stream and stores it in the wave buffer.
- virtual void [Seek](#) (double timePoint)=0
- virtual double [TotalTime](#) () const =0
Returns the total time of the stream (in seconds).
- virtual std::vector< std::string > [InfoComments](#) () const =0
Returns an optional list of strings, containing informational commentaries such as "ARTIST=John Doe".
- virtual [WaveBufferFormat](#) [GetFormat](#) () const =0
Returns the native wave buffer format of this audio stream.

3.1.1 Detailed Description

Audio stream interface.

3.1.2 Member Function Documentation

3.1.2.1 virtual void Ac::AudioStream::Seek (double *timePoint*) [pure virtual]

Sets the new time point from where to stream the audio data.

Parameters

in	<i>timePoint</i>	Specifies the new position (in seconds).
----	------------------	--

Exceptions

<code>std::runtime_exception</code>	If something went wrong.
-------------------------------------	--------------------------

3.1.2.2 virtual std::size_t Ac::AudioStream::StreamWaveBuffer (WaveBuffer & *buffer*) [pure virtual]

Reads the audio data from the active stream and stores it in the wave buffer.

Parameters

out	<i>buffer</i>	Specifies the output wave buffer.
-----	---------------	-----------------------------------

Returns

Number of bytes read from the input stream. If this is zero, the end of the stream has been reached.

Exceptions

<code>std::runtime_exception</code>	If something went wrong while reading.
-------------------------------------	--

The documentation for this class was generated from the following file:

- AudioStream.h

3.2 Ac::AudioSystem Class Reference

Audio system interface. All coordinates or 3D sounds are meant to be in a left-handed coordinates system, i.e. positive Z values point into your monitor, and negative Z values point out of your monitor.

```
#include <AudioSystem.h>
```

Public Member Functions

- **AudioSystem** (const [AudioSystem](#) &)=delete
- **AudioSystem** & **operator=** (const [AudioSystem](#) &)=delete
- const std::string & [GetName](#) () const
Returns the name of this audio system.
- virtual std::string [GetVersion](#) () const =0
Returns a descriptive version string of this audio system (e.g. "OpenAL 1.1").
- virtual std::unique_ptr< [Sound](#) > [CreateSound](#) ()=0
Creates an empty sound which can later be filled with a wave buffer.
- std::unique_ptr< [Sound](#) > [CreateSound](#) (const [WaveBuffer](#) &waveBuffer)
Creates a sound initialized with specified wave buffer.
- std::unique_ptr< [Sound](#) > [LoadSound](#) (const std::string &filename, const SoundFlags::BitMask flags=0)

- Loads the specified sound from file.*
- void [Play](#) (const std::string &filename, float volume=1.0f, float pitch=1.0f)
 - Plays directly the specified sound file.*
- void [Streaming](#) ([Sound](#) &sound, [WaveBuffer](#) &waveBuffer)
 - Performs the audio streaming process. This should be called once per frame in a real-time application.*
- void [Streaming](#) ([Sound](#) &sound)
 - Performs the audio streaming process with a default wave buffer configuration.*
- virtual void [SetListenerPosition](#) (const Gs::Vector3f &position)=0
 - Sets the listener world position. By default (0, 0, 0).*
- virtual Gs::Vector3f [GetListenerPosition](#) () const =0
 - Returns the listener world position.*
- virtual void [SetListenerVelocity](#) (const Gs::Vector3f &velocity)=0
 - Sets the listener world velocity. This is used for the "Doppler"-effect. By default (0, 0, 0).*
- virtual Gs::Vector3f [GetListenerVelocity](#) () const =0
 - Returns the listener world velocity.*
- virtual void [SetListenerOrientation](#) (const [ListenerOrientation](#) &orientation)=0
 - Sets the listener world orientation. By default { (0, 0, 0), (0, 0, 0) }.*
- virtual [ListenerOrientation](#) [GetListenerOrientation](#) () const =0
 - Returns the listener world position.*
- [WaveBuffer](#) [ReadWaveBuffer](#) (const std::string &filename)
 - Reads the audio data from the specified file and stores it in the output wave buffer.*
- [WaveBuffer](#) [ReadWaveBuffer](#) (std::istream &stream)
 - Reads the audio data from the specified stream and stores it in the output wave buffer.*
- std::unique_ptr< [AudioStream](#) > [OpenAudioStream](#) (const std::string &filename)
 - Opens a new audio stream form the specified file.*
- std::unique_ptr< [AudioStream](#) > [OpenAudioStream](#) (std::unique_ptr< std::istream > &&stream)
 - Opens a new audio stream.*
- bool [WriteAudioBuffer](#) (const AudioFormats format, std::ostream &stream, const [WaveBuffer](#) &waveBuffer)
 - Writes the audio data to the specified stream.*
- std::unique_ptr< [Microphone](#) > [QueryMicrophone](#) ()

Static Public Member Functions

- static std::vector< std::string > [FindModules](#) ()
 - Returns the list of all available audio system modules for the current platform (e.g. on Windows this might be { "OpenAL", "XAudio2" }, but on MacOS it might be only { "OpenAL" }).*
- static std::shared_ptr< [AudioSystem](#) > [Load](#) (const std::string &moduleName)
 - Loads a new audio system from the specified module.*
- static std::shared_ptr< [AudioSystem](#) > [Load](#) ()
 - Returns the first available audio system.*
- static AudioFormats [DetermineAudioFormat](#) (std::istream &stream)
 - Determines the audio format of the specified stream.*

3.2.1 Detailed Description

Audio system interface. All coordinates or 3D sounds are meant to be in a left-handed coordinates system, i.e. positive Z values point into your monitor, and negative Z values point out of your monitor.

3.2.2 Member Function Documentation

3.2.2.1 static AudioFormats Ac::AudioSystem::DetermineAudioFormat (std::istream & *stream*) [static]

Determines the audio format of the specified stream.

Parameters

<i>in, out</i>	<i>stream</i>	Specifies the input stream where the audio format is to be determined from.
----------------	---------------	---

Remarks

This function will jump to the beginning of the input stream to read the magic number and then sets the reading position back to the previous position.

Returns

Determined audio format or AudioFormats::Unknown if the audio format is not supported.

3.2.2.2 static std::shared_ptr<AudioSystem> Ac::AudioSystem::Load (const std::string & moduleName) [static]

Loads a new audio system from the specified module.

Parameters

<i>in</i>	<i>moduleName</i>	Specifies the name from which the new audio system is to be loaded. This denotes a dynamic library (*.dll-files on Windows, *.so-files on Unix systems). If compiled in debug mode, the postfix "D" is appended to the module name. Moreover, the platform dependent file extension is always added automatically as well as the prefix "AcLib_", i.e. a module name "OpenAL" will be translated to "AcLib_OpenALD.dll", if compiled on Windows in Debug mode.
-----------	-------------------	--

Remarks

Usually the return type is a std::unique_ptr, but the AcousticsLib needs to keep track of the existence of this audio system because only a single instance can be loaded at a time. So a std::weak_ptr is stored internally to check if it has been expired (see http://en.cppreference.com/w/cpp/memory/weak_ptr/expired), and this type can only refer to a std::shared_ptr.

Exceptions

<i>std::runtime_exception</i>	If loading the audio system from the specified module failed.
<i>std::runtime_exception</i>	If there is already a loaded instance of an audio system (make sure there are no more shared pointer references to the previous audio system!)

3.2.2.3 std::unique_ptr<Sound> Ac::AudioSystem::LoadSound (const std::string & filename, const SoundFlags::BitMask flags = 0)

Loads the specified sound from file.

Parameters

in	<i>flags</i>	Specifies the bit mask flags. This can be a bitwas OR combination of the values of the "SoundFlags" enumeration. By default 0.
----	--------------	--

See also

[SoundFlags](#)

3.2.2.4 `std::unique_ptr<AudioStream> Ac::AudioSystem::OpenAudioStream (const std::string & filename)`

Opens a new audio stream form the specified file.

Parameters

in	<i>filename</i>	Specifies the filename of the input file stream.
----	-----------------	--

See also

`OpenAudioStream(std::istream&)`

3.2.2.5 `std::unique_ptr<AudioStream> Ac::AudioSystem::OpenAudioStream (std::unique_ptr<std::istream> && stream)`

Opens a new audio stream.

Parameters

in	<i>stream</i>	Specifies the input stream to read from. This stream must be opened in binary mode!
----	---------------	---

Returns

New [AudioStream](#) object or null if 'format' is invalid.

Remarks

The input stream must be a unique pointer, so that the returned audio stream object can take care of the input stream to read from.

Exceptions

<code>std::runtime_exception</code>	If something went wrong while opening the stream.
-------------------------------------	---

3.2.2.6 void Ac::AudioSystem::Play (const std::string & *filename*, float *volume* = 1.0f, float *pitch* = 1.0f)

Plays directly the specified sound file.

Parameters

in	<i>filename</i>	Specifies the sound file to play.
in	<i>volume</i>	Specifies the volume. By default 1.
in	<i>pitch</i>	Specifies the pitch (or frequency multiplier). By default 1.

3.2.2.7 WaveBuffer Ac::AudioSystem::ReadWaveBuffer (const std::string & *filename*)

Reads the audio data from the specified file and stores it in the output wave buffer.

Parameters

in	<i>filename</i>	Specifies the filename of the input file stream.
----	-----------------	--

See also

ReadWaveBuffer(const AudioFormats, std::istream&, WaveBuffer&)

3.2.2.8 WaveBuffer Ac::AudioSystem::ReadWaveBuffer (std::istream & *stream*)

Reads the audio data from the specified stream and stores it in the output wave buffer.

Parameters

in, out	<i>stream</i>	Specifies the input stream to read from. This stream must be opened in binary mode!
---------	---------------	---

Exceptions

<i>std::runtime_exception</i>	If something went wrong while reading.
-------------------------------	--

3.2.2.9 void Ac::AudioSystem::Streaming (Sound & *sound*, WaveBuffer & *waveBuffer*)

Performs the audio streaming process. This should be called once per frame in a real-time application.

Parameters

in, out	<i>sound</i>	Specifies the sound for which the streaming is to be performed.
in, out	<i>waveBuffer</i>	Specifies the wave buffer for buffering during the streaming process. If the buffer is empty, an appropriate buffer will be configured for streaming.

Remarks

If the specified sound does not have a source stream, this function call has no effect.

See also

Sound::SetSourceStream

3.2.2.10 void Ac::AudioSystem::Streaming (Sound & *sound*)

Performs the audio streaming process with a default wave buffer configuration.

See also

[Streaming\(Sound&, WaveBuffer&\)](#)

3.2.2.11 bool Ac::AudioSystem::WriteAudioBuffer (const AudioFormats *format*, std::ostream & *stream*, const WaveBuffer & *waveBuffer*)

Writes the audio data to the specified stream.

Parameters

in, out	<i>stream</i>	Specifies the output stream to write to. This stream must be opened in binary mode!
out	<i>waveBuffer</i>	Specifies the input wave buffer.

Returns

True if the stream has been written successfully.

Exceptions

<i>std::runtime_exception</i>	If something went wrong while writing.
-------------------------------	--

The documentation for this class was generated from the following file:

- AudioSystem.h

3.3 Ac::ChannelTypes2 Struct Reference

Public Types

- enum : unsigned short { **Left** = 0, **Right** }

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.4 Ac::ChannelTypes3 Struct Reference

Public Types

- enum : unsigned short { **Left** = 0, **Center**, **Right** }

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.5 Ac::ChannelTypes4 Struct Reference

Public Types

- enum : unsigned short { **FrontLeft** = 0, **FrontRight**, **RearLeft**, **RearRight** }

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.6 Ac::ChannelTypes5 Struct Reference

Public Types

- enum : unsigned short {
 FrontLeft = 0, **FrontCenter**, **FrontRight**, **RearLeft**,
 RearRight }

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.7 Ac::ChannelTypes5_1 Struct Reference

Public Types

- enum : unsigned short {
 FrontLeft = 0, **FrontCenter**, **FrontRight**, **RearLeft**,
 RearRight, **LFE** }

3.7.1 Member Enumeration Documentation

3.7.1.1 anonymous enum : unsigned short

Enumerator

LFE Low-Frequency-Effects.

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.8 Ac::ChannelTypes6_1 Struct Reference

Public Types

- enum : unsigned short {
 FrontLeft = 0, **FrontCenter**, **FrontRight**, **SideLeft**,
 SideRight, **RearCenter**, [LFE](#) }

3.8.1 Member Enumeration Documentation

3.8.1.1 anonymous enum : unsigned short

Enumerator

LFE Low-Frequency-Effects.

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.9 Ac::ChannelTypes7_1 Struct Reference

Public Types

- enum : unsigned short {
 FrontLeft = 0, **FrontCenter**, **FrontRight**, **SideLeft**,
 SideRight, **RearLeft**, **RearRight**, [LFE](#) }

3.9.1 Member Enumeration Documentation

3.9.1.1 anonymous enum : unsigned short

Enumerator

LFE Low-Frequency-Effects.

The documentation for this struct was generated from the following file:

- ChannelTypes.h

3.10 Ac::ListenerOrientation Struct Reference

Structure for the 3D listener orientation with at- and up- vectors.

```
#include <AudioSystem.h>
```

Public Attributes

- `Gs::Vector3f atVector`
- `Gs::Vector3f upVector`

3.10.1 Detailed Description

Structure for the 3D listener orientation with at- and up- vectors.

The documentation for this struct was generated from the following file:

- `AudioSystem.h`

3.11 Ac::Microphone Class Reference

[Microphone](#) interface.

```
#include <Microphone.h>
```

Public Member Functions

- **Microphone** (const [Microphone](#) &)=delete
- **Microphone** & **operator=** (const [Microphone](#) &)=delete
- virtual std::vector< [MicrophoneDevice](#) > [QueryDevices](#) () const =0
Returns a list of all available microphone devices.
- virtual std::unique_ptr< [WaveBuffer](#) > [ReceivedInput](#) ()=0
Returns the received audio input from this microphone.
- virtual void [Start](#) (const [WaveBufferFormat](#) &waveFormat, std::size_t sampleFrames, std::size_t device↵
Index=[standardDeviceIndex](#))=0
Starts the recording process.
- virtual void [Start](#) (const [WaveBufferFormat](#) &waveFormat, double duration, std::size_t device↵
Index=[standardDeviceIndex](#))=0
Starts the recording process.
- virtual void [Stop](#) ()=0
Stops the recording process.
- virtual bool [IsRecording](#) () const =0
Returns true if the recording process is currently running.

Static Public Attributes

- static const std::size_t [standardDeviceIndex](#) = std::size_t(-1)
Standard audio input device index.

3.11.1 Detailed Description

[Microphone](#) interface.

3.11.2 Member Function Documentation

3.11.2.1 virtual bool Ac::Microphone::IsRecording () const [pure virtual]

Returns true if the recording process is currently running.

Remarks

The recording process can be started with the "Start" function.

See also

[Start](#)
[Stop](#)

3.11.2.2 virtual std::unique_ptr<WaveBuffer> Ac::Microphone::ReceivedInput () [pure virtual]

Returns the received audio input from this microphone.

Returns

Unique pointer to the new wave buffer or null if there is currently no more data.

Remarks

The recording process must be started with the "Start" function, before anything can be recorded. Example usage:

```
// Start recording
mic->Start();

// Process microphone input until there is no more to process
WaveBuffer inputBuffer, outputBuffer;
while (mic->ProcessInput(inputBuffer))
{
    // Append input buffer to output buffer
    outputBuffer.Append(inputBuffer);

    // Stop recording when user presses a key for instance
    if (...)
        mic->Stop();
}
```

See also

[Start](#)
[Stop](#)

3.11.2.3 virtual void Ac::Microphone::Start (const WaveBufferFormat & waveFormat, std::size_t sampleFrames, std::size_t deviceIndex = standardDeviceIndex) [pure virtual]

Starts the recording process.

Parameters

in	<i>waveFormat</i>	Specifies the wave buffer format which is to be used for the receiver buffer, which can be acquired with the "ReceivedInput" function.
in	<i>sampleFrames</i>	Specifies how many sample frames shall be received at once. The larger this value, the larger the latency.
in	<i>deviceIndex</i>	Specifies the input device index (beginning with 0). By default the standard device is used.

Remarks

Before a new recording process can be started, the previous one must be stopped. To select an appropriate device index, use the "QueryDevices" function, to query all available input devices.

See also

[Stop
IsRecording](#)

3.11.2.4 `virtual void Ac::Microphone::Start (const WaveBufferFormat & waveFormat, double duration, std::size_t deviceIndex = standardDeviceIndex) [pure virtual]`

Starts the recording process.

Remarks

Same as the other "Start" function but here the duration (in seconds) specifies the latency instead of the sample count.

See also

`Start(const WaveBufferFormat&, std::size_t)`

The documentation for this class was generated from the following file:

- `Microphone.h`

3.12 Ac::MicrophoneDevice Struct Reference

[Microphone](#) device descriptor structure.

```
#include <Microphone.h>
```

Public Attributes

- std::string [name](#)
Name of the microphone device.
- std::vector< [WaveBufferFormat](#) > [formats](#)
List of all supported standard formats.

3.12.1 Detailed Description

[Microphone](#) device descriptor structure.

The documentation for this struct was generated from the following file:

- [Microphone.h](#)

3.13 Ac::Renderer Class Reference

[Renderer](#) interface used in conjunction with the Visualizer.

```
#include <Renderer.h>
```

Public Member Functions

- virtual void **BeginDrawing** (const Gs::Vector2i &size)=0
- virtual void **EndDrawing** ()=0
- virtual void **DrawLineList** (const std::vector< Gs::Vector2i > &vertices)=0
- virtual void **DrawLineStrip** (const std::vector< Gs::Vector2i > &vertices)=0

3.13.1 Detailed Description

[Renderer](#) interface used in conjunction with the Visualizer.

See also

[Visualizer](#)

The documentation for this class was generated from the following file:

- [Renderer.h](#)

3.14 Ac::Sound Class Reference

[Sound](#) source interface.

```
#include <Sound.h>
```

Public Member Functions

- **Sound** (const [Sound](#) &)=delete
- **Sound & operator=** (const [Sound](#) &)=delete
- virtual void [Play](#) ()=0
Starts the sound playback.
- virtual void [Pause](#) ()=0
Pauses the sound playback at the current position.
- virtual void [Stop](#) ()=0
Stops the sound playback.
- virtual void [SetLooping](#) (bool enable)=0
Enables or disables sound looping.
- virtual bool [GetLooping](#) () const =0
Returns true if sound looping is enabled.
- virtual void [SetVolume](#) (float volume)=0
Sets the sound volume in the range [0, +inf). By default 1.
- virtual float [GetVolume](#) () const =0
Returns the sound volume.
- virtual void [SetPitch](#) (float pitch)=0
Sets the sound pitch (or frequency ratio) in the range (0, +inf). By default 1.
- virtual float [GetPitch](#) () const =0
Returns the sound pitch.
- virtual bool [IsPlaying](#) () const =0
Returns true if the sound is currently being played.
- virtual bool [IsPaused](#) () const =0
Returns true if the sound is currently being played but is pause mode.
- virtual void [SetSeek](#) (double position)=0
Seeks the current playback position (in seconds) in the range [0, +inf).
- virtual double [GetSeek](#) () const =0
Returns the current playback position (in seconds).
- virtual double [TotalTime](#) () const =0
Returns the total time (in seconds) this sound takes to be played.
- virtual void [AttachBuffer](#) (const [WaveBuffer](#) &waveBuffer)=0
Attaches the specified wave buffer to this sound.
- virtual void [QueueBuffer](#) (const [WaveBuffer](#) &waveBuffer)=0
Appends the specified buffer at the end of the buffer queue of this sound.
- virtual std::size_t [GetQueueSize](#) () const =0
Returns the current size of the buffer queue.
- virtual std::size_t [GetProcessedQueueSize](#) () const =0
Returns the number of the processed buffer in the queue.
- void [SetStreamSource](#) (const std::shared_ptr< [AudioStream](#) > &streamSource)
Connects this sound with the specified stream source.
- const std::shared_ptr< [AudioStream](#) > & [GetStreamSource](#) () const
Returns the previously connected stream source.
- virtual void [Enable3D](#) (bool enable=true)=0
Enables or disables the 3D sound feature. By default disabled.
- virtual bool [Is3DEnabled](#) () const =0
Returns true if 3D sound effect is enabled.
- virtual void [SetPosition](#) (const Gs::Vector3f &position)=0
Sets the world position of this 3D sound. By default (0, 0, 0).
- virtual Gs::Vector3f [GetPosition](#) () const =0

- Returns the world position of this 3D sound.*
- virtual void [SetVelocity](#) (const Gs::Vector3f &velocity)=0
Sets the world velocity of this 3D sound. The velocity is used for the "Doppler"-effect. By default (0, 0, 0).
- virtual Gs::Vector3f [GetVelocity](#) () const =0
Returns the world velocity of this 3D sound.
- virtual void [SetSpaceRelative](#) (bool enable)=0
Specifies whether to make the coordinate space of this sound relative to the listener or not. By default false.
- virtual bool [GetSpaceRelative](#) () const =0
Returns true if the coordinate space of this sound is relative to the listener or not.

3.14.1 Detailed Description

[Sound](#) source interface.

3.14.2 Member Function Documentation

3.14.2.1 virtual void Ac::Sound::AttachBuffer (const WaveBuffer & *waveBuffer*) [pure virtual]

Attaches the specified wave buffer to this sound.

Remarks

If this function is used, only a single buffer can be added to the sound, and the previous buffer will be removed.

See also

[QueueBuffer](#)

3.14.2.2 virtual void Ac::Sound::Enable3D (bool *enable* = true) [pure virtual]

Enables or disables the 3D sound feature. By default disabled.

Remarks

All 3D sound functions (for position, velocity, and relative space) have no effect until this sound was enabled to be a 3D sound. Moreover, all 3D attributes (position, velocity, and relative space) are reset whenever this function is called!

See also

[SetPosition](#)
[SetVelocity](#)
[SetSpaceRelative](#)

3.14.2.3 virtual std::size_t Ac::Sound::GetProcessedQueueSize () const [pure virtual]

Returns the number of the processed buffer in the queue.

Remarks

This can be used for manual audio streaming if the "AudioSystem::Streaming" function is not used. Example usage:

```
// Initialize buffer queue with 10 buffers
for (int i = 0; i < 10 && audioStream->StreamWaveBuffer(waveBuffer) > 0; ++i)
    sound->QueueBuffer(waveBuffer);

// Start continuous streaming
while (sound->IsPlaying())
{
    while (sound->GetProcessedQueueSize() > 0)
    {
        if (audioStream->StreamWaveBuffer(waveBuffer) > 0)
            sound->QueueBuffer(waveBuffer);
        else
            break;
    }
}
```

See also

[AudioSystem::Streaming](#)

3.14.2.4 virtual std::size_t Ac::Sound::GetQueueSize () const [pure virtual]

Returns the current size of the buffer queue.

See also

[QueueBuffer](#)
[GetProcessedQueueSize](#)

3.14.2.5 virtual void Ac::Sound::QueueBuffer (const WaveBuffer & waveBuffer) [pure virtual]

Appends the specified buffer at the end of the buffer queue of this sound.

Remarks

If this function is used, the sound will be managed for audio streaming.

See also

[AttachBuffer](#)

3.14.2.6 virtual void Ac::Sound::SetLooping (bool enable) [pure virtual]

Enables or disables sound looping.

Parameters

<code>in</code>	<code>enable</code>	If true, the sound will be played from the beginning, after the end was reached. By default false.
-----------------	---------------------	--

3.14.2.7 void Ac::Sound::SetStreamSource (const std::shared_ptr< AudioStream > & *streamSource*) [inline]

Connects this sound with the specified stream source.

Remarks

This is used for the audio system, to perform continous streaming automatically. If you do the streaming manually, you don't necessarily need this function.

The documentation for this class was generated from the following file:

- Sound.h

3.15 Ac::SoundFlags Struct Reference

Loading sound flags enumeration.

```
#include <AudioSystem.h>
```

Public Types

- enum { [AlwaysCreateSound](#) = (1 << 0), [Enable3D](#) = (1 << 1) }
- using **BitMask** = unsigned int

3.15.1 Detailed Description

Loading sound flags enumeration.

3.15.2 Member Enumeration Documentation

3.15.2.1 anonymous enum

Enumerator

AlwaysCreateSound Indicates that "LoadSound" shall always return a valid [Sound](#) object, even if the sound file could not be loaded.

Enable3D Indicates that "LoadSound" shall return a [Sound](#) object which is prepared for the 3D sound features.

See also

[Sound::Enable3D](#)

The documentation for this struct was generated from the following file:

- AudioSystem.h

3.16 Ac::WaveBuffer Class Reference

Data model for an audio wave buffer.

```
#include <WaveBuffer.h>
```

Public Member Functions

- **WaveBuffer** (const [WaveBuffer](#) &)=default
- **WaveBuffer** & **operator=** (const [WaveBuffer](#) &)=default
- **WaveBuffer** (const [WaveBufferFormat](#) &format)
- **WaveBuffer** ([WaveBuffer](#) &&other)
- **WaveBuffer** & **operator=** ([WaveBuffer](#) &&other)
- std::size_t [GetSampleFrames](#) () const
Returns the number of samples per channel.
- void [SetSampleFrames](#) (std::size_t sampleFrames)
Sets the new number of samples per channel.
- double [GetTotalTime](#) () const
Returns the total time (in seconds) which is required to play this entire wave buffer.
- void [SetTotalTime](#) (double duration)
Resizes the buffer to the specified total time (in seconds).
- double [ReadSample](#) (std::size_t index, unsigned short channel) const
Returns the sample at the specified index of the specified channel.
- void [WriteSample](#) (std::size_t index, unsigned short channel, double sample)
Sets the sample at the specified index of the specified channel.
- double [ReadSample](#) (double timePoint, unsigned short channel) const
Returns the sample at the specified time point of the specified channel.
- void [WriteSample](#) (double timePoint, unsigned short channel, double sample)
Sets the sample at the specified time point of the specified channel.
- std::size_t [GetIndexFromTimePoint](#) (double timePoint) const
Determines the sample index for the specified time point (in seconds).
- double [GetTimePointFromIndex](#) (std::size_t index) const
Determines the time point (in seconds) for the specified sample index.
- void [SetFormat](#) (const [WaveBufferFormat](#) &format)
Sets the new wave buffer format.
- void [SetChannels](#) (unsigned short channels)
Sets the new number of channels. By default 2.
- void [SwapEndianness](#) ()
Swaps the endianness (byte order) of each sample between little-endian and big-endian.
- void [ForEachSample](#) (const SampleIterationFunction &iterator, std::size_t indexBegin, std::size_t indexEnd)
Iterates over all samples of this wave buffer within the specified range.
- void [ForEachSample](#) (const SampleIterationFunction &iterator, double timeBegin, double timeEnd)
Iterates over all samples of this wave buffer within the specified time range.
- void [ForEachSample](#) (const SampleIterationFunction &iterator)
Iterates over all samples of this wave buffer.
- void [ForEachSample](#) (const SampleConstIterationFunction &iterator, std::size_t indexBegin, std::size_t indexEnd) const
Iterates over all samples of this wave buffer within the specified range with a constant iterator.
- void [ForEachSample](#) (const SampleConstIterationFunction &iterator, double timeBegin, double timeEnd) const

- Iterates over all samples of this wave buffer within the specified time range with a constant iterator.*
- void [ForEachSample](#) (const SampleConstIterationFunction &iterator) const
Iterates over all samples of this wave buffer with a constant iterator.
- void [Append](#) (const [WaveBuffer](#) &other)
Appends the specified wave buffer to this buffer.
- std::size_t [BufferSize](#) () const
Returns the actual PCM buffer size (in bytes).
- char * [Data](#) ()
Returns a raw pointer to the PCM buffer data.
- const char * [Data](#) () const
Returns a constant raw pointer to the PCM buffer data.
- const [WaveBufferFormat](#) & [GetFormat](#) () const
Returns the format description of this wave buffer.

3.16.1 Detailed Description

Data model for an audio wave buffer.

Remarks

This class manages the PCM (Pulse Code Modulation) buffer by abstracting the underlying audio samples (8 or 16 bit, signed or unsigned) to double precision floating-points in the normalized range [-1, 1]. Here is a usage example:

```
// Create wave buffer with 44 kHz sample rate, 16-bit samples, and two channels
Ac::WaveBuffer buffer(WaveBufferFormat(Ac::Synthesizer::sampleRate44kHz, 16, 2));

// Allocate internal buffer to store samples for 4.5 seconds.
buffer.SetTotalTime(4.5);

// Generate samples for the first half of the wave buffer (here a sine wave of 350 Hz)
buffer.ForEachSample(Ac::Synthesizer::SineGenerator(0.3, 0.0, 350.0), 0.0, 2.25);

// Generate sample for the second half of the wave buffer (here with a custom function)
buffer.ForEachSample(
    [](double& sample, unsigned short channel, std::size_t index, double timePoint)
    {
        //sample += ...
    },
    2.25, 4.5
);

// Now create sound with our buffer
auto sound = audioSystem->CreateSound(buffer);
sound->Play();
```

3.16.2 Member Function Documentation

3.16.2.1 void Ac::WaveBuffer::Append (const WaveBuffer & other)

Appends the specified wave buffer to this buffer.

Parameters

in	<i>buffer</i>	Specifies the new wave buffer which is to be appended to this buffer. The format of the input buffer will be set to the format of this buffer (if they are unequal).
----	---------------	--

Remarks

The input buffer must not be the same as this buffer, i.e. 'buffer.Append(buffer)' is an invalid operation and the behavior is undefined!

See also

[SetFormat](#)

3.16.2.2 void Ac::WaveBuffer::ForEachSample (const SampleIterationFunction & *iterator*, std::size_t *indexBegin*, std::size_t *indexEnd*)

Iterates over all samples of this wave buffer within the specified range.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
in	<i>indexBegin</i>	Specifies the first sample index.
in	<i>indexEnd</i>	Specifies the last sample index. The ending is inclusive, i.e. the iteration range is [indexBegin, indexEnd].

See also

SampleIterationFunction

3.16.2.3 void Ac::WaveBuffer::ForEachSample (const SampleIterationFunction & *iterator*, double *timeBegin*, double *timeEnd*)

Iterates over all samples of this wave buffer within the specified time range.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
in	<i>timeBegin</i>	Specifies the beginning time point (in seconds). This will be clamped to [0, +inf).
in	<i>timeEnd</i>	Specifies the ending time point (in seconds). This will be clamped to [timeBegin, +inf). The ending is inclusive, i.e. the iteration range is [timeBegin, timeEnd].

See also

SampleIterationFunction

3.16.2.4 void Ac::WaveBuffer::ForEachSample (const SampleIterationFunction & *iterator*)

Iterates over all samples of this wave buffer.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
----	-----------------	---

See also

SampleIterationFunction

3.16.2.5 void Ac::WaveBuffer::ForEachSample (const SampleConstIterationFunction & *iterator*, std::size_t *indexBegin*, std::size_t *indexEnd*) const

Iterates over all samples of this wave buffer within the specified range with a constant iterator.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
in	<i>indexBegin</i>	Specifies the first sample index.
in	<i>indexEnd</i>	Specifies the last sample index. The ending is inclusive, i.e. the iteration range is [indexBegin, indexEnd].

See also

SampleIterationFunction

3.16.2.6 void Ac::WaveBuffer::ForEachSample (const SampleConstIterationFunction & *iterator*, double *timeBegin*, double *timeEnd*) const

Iterates over all samples of this wave buffer within the specified time range with a constant iterator.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
in	<i>timeBegin</i>	Specifies the beginning time point (in seconds). This will be clamped to [0, +inf).
in	<i>timeEnd</i>	Specifies the ending time point (in seconds). This will be clamped to [timeBegin, +inf). The ending is inclusive, i.e. the iteration range is [timeBegin, timeEnd].

See also

SampleIterationFunction

3.16.2.7 void Ac::WaveBuffer::ForEachSample (const SampleConstIterationFunction & *iterator*) const

Iterates over all samples of this wave buffer with a constant iterator.

Parameters

in	<i>iterator</i>	Specifies the sample iteration callback function. This function will be used to modify each sample.
----	-----------------	---

See also

[SampleIterationFunction](#)

3.16.2.8 std::size_t Ac::WaveBuffer::GetIndexFromTimePoint (double *timePoint*) const

Determines the sample index for the specified time point (in seconds).

See also

[ReadSample\(std::size_t, unsigned short\) const](#)
[WriteSample\(std::size_t, unsigned short, double\)](#)
[GetTimePointFromIndex](#)

3.16.2.9 double Ac::WaveBuffer::GetTimePointFromIndex (std::size_t *index*) const

Determines the time point (in seconds) for the specified sample index.

See also

[GetIndexFromTimePoint](#)

3.16.2.10 double Ac::WaveBuffer::ReadSample (std::size_t *index*, unsigned short *channel*) const

Returns the sample at the specified index of the specified channel.

Parameters

in	<i>index</i>	Specifies the sample index. One can use the "GetIndexFromTimePoint" function, to determine the index by the time point (in seconds).
in	<i>channel</i>	Specifies the channel from which to read the sample. This will be clamped to the range [0, GetFormat().channels).

Returns

The read sample in the range [-1, 1].

See also

[GetIndexFromTimePoint](#)

3.16.2.11 void Ac::WaveBuffer::SetChannels (unsigned short *channels*)

Sets the new number of channels. By default 2.

Remarks

This is a shortcut for the following behavior:

```
auto format = this->GetFormat();
format.channels = channels;
this->SetFormat(format);
```

See also

[SetFormat](#)

3.16.2.12 void Ac::WaveBuffer::SetFormat (const WaveBufferFormat & *format*)

Sets the new wave buffer format.

Parameters

in	<i>format</i>	Specifies the new wave buffer format. If this is equal to the previous buffer, this function has no effect.
----	---------------	---

Remarks

This function may take some computational overhead, since the entire PCM buffer needs to be resampled.

3.16.2.13 void Ac::WaveBuffer::SwapEndianness ()

Swaps the endianness (byte order) of each sample between little-endian and big-endian.

Remarks

Per default, all data is read in little endian format on an x86 (IA-32) and x64 (AMD64) processor. Therefore, calling this function swaps the byte order to big-endian. This is used for the AIFF reader for instance. If the bits per sample is 8 (or lower), then this function has no effect, since byte order does not matter for a single byte.

The documentation for this class was generated from the following file:

- WaveBuffer.h

3.17 Ac::WaveBufferFormat Struct Reference

Wave buffer format descriptor structure.

```
#include <WaveBufferFormat.h>
```

Public Member Functions

- **WaveBufferFormat** (const [WaveBufferFormat](#) &)=default
- **WaveBufferFormat** & **operator=** (const [WaveBufferFormat](#) &)=default
- **WaveBufferFormat** (unsigned int [sampleRate](#), unsigned short [bitsPerSample](#), unsigned short [channels](#))
- std::size_t [BytesPerFrame](#) () const
Returns the size (in bytes) for each sample frame (or rather sample block alignment) which is computed as follows: $(\text{channels} * \text{bitsPerSample}) / 8$.
- std::size_t [BytesPerSecond](#) () const
Returns the number of bytes this buffer format requires for each second: $\text{sampleRate} * \text{BytesPerFrame}()$.
- double [TotalTime](#) (std::size_t bufferSize) const
Returns the total time (in seconds) a PCM buffer with the specified size (in bytes) requires to play with this wave buffer format.
- bool [IsSigned](#) () const
Returns true if this is a signed format. This is true if $(\text{bitsPerSample} > 8)$ holds true.

Public Attributes

- unsigned int [sampleRate](#) = 44100
Number of samples per second (in Hz). Default value is 44100.
- unsigned short [bitsPerSample](#) = 16
Number of bits per sample. Typical values are 8, 12, 16, 24, and 32. Default value is 16.
- unsigned short [channels](#) = 1
Number of channels. 1 for mono and 2 for stereo. Default value is 1.

3.17.1 Detailed Description

Wave buffer format descriptor structure.

3.17.2 Member Data Documentation

3.17.2.1 unsigned int Ac::WaveBufferFormat::sampleRate = 44100

Number of samples per second (in Hz). Default value is 44100.

Remarks

The commonly used sample rates are: 8 kHz, 11.025 kHz, 22.05 kHz, and 44.1 kHz.

The documentation for this struct was generated from the following file:

- WaveBufferFormat.h

Index

- Ac::AudioStream, 9
 - Seek, 9
 - StreamWaveBuffer, 10
- Ac::AudioSystem, 10
 - DetermineAudioFormat, 12
 - Load, 13
 - LoadSound, 13
 - OpenAudioStream, 14
 - Play, 14
 - ReadWaveBuffer, 15
 - Streaming, 15, 16
 - WriteAudioBuffer, 16
- Ac::ChannelTypes2, 16
- Ac::ChannelTypes3, 17
- Ac::ChannelTypes4, 17
- Ac::ChannelTypes5, 17
- Ac::ChannelTypes5_1, 17
 - LFE, 18
- Ac::ChannelTypes6_1, 18
 - LFE, 18
- Ac::ChannelTypes7_1, 18
 - LFE, 18
- Ac::ListenerOrientation, 19
- Ac::Microphone, 19
 - IsRecording, 20
 - ReceivedInput, 20
 - Start, 20, 21
- Ac::MicrophoneDevice, 21
- Ac::Renderer, 22
- Ac::Sound, 22
 - AttachBuffer, 24
 - Enable3D, 24
 - GetProcessedQueueSize, 24
 - GetQueueSize, 25
 - QueueBuffer, 25
 - SetLooping, 25
 - SetStreamSource, 26
- Ac::SoundFlags, 26
 - AlwaysCreateSound, 26
 - Enable3D, 26
- Ac::WaveBuffer, 27
 - Append, 28
 - ForEachSample, 29, 30
 - GetIndexFromTimePoint, 31
 - GetTimePointFromIndex, 31
 - ReadSample, 31
 - SetChannels, 31
 - SetFormat, 32
 - SwapEndianness, 32
- Ac::WaveBufferFormat, 32
 - sampleRate, 33
- AlwaysCreateSound
 - Ac::SoundFlags, 26
- Append
 - Ac::WaveBuffer, 28
- AttachBuffer
 - Ac::Sound, 24
- DetermineAudioFormat
 - Ac::AudioSystem, 12
- Enable3D
 - Ac::Sound, 24
 - Ac::SoundFlags, 26
- ForEachSample
 - Ac::WaveBuffer, 29, 30
- GetIndexFromTimePoint
 - Ac::WaveBuffer, 31
- GetProcessedQueueSize
 - Ac::Sound, 24
- GetQueueSize
 - Ac::Sound, 25
- GetTimePointFromIndex
 - Ac::WaveBuffer, 31
- IsRecording
 - Ac::Microphone, 20
- LFE
 - Ac::ChannelTypes5_1, 18
 - Ac::ChannelTypes6_1, 18
 - Ac::ChannelTypes7_1, 18
- Load
 - Ac::AudioSystem, 13
- LoadSound
 - Ac::AudioSystem, 13
- OpenAudioStream
 - Ac::AudioSystem, 14
- Play
 - Ac::AudioSystem, 14
- QueueBuffer
 - Ac::Sound, 25
- ReadSample
 - Ac::WaveBuffer, 31

ReadWaveBuffer
 Ac::AudioSystem, [15](#)
ReceivedInput
 Ac::Microphone, [20](#)

sampleRate
 Ac::WaveBufferFormat, [33](#)
Seek
 Ac::AudioStream, [9](#)
SetChannels
 Ac::WaveBuffer, [31](#)
SetFormat
 Ac::WaveBuffer, [32](#)
SetLooping
 Ac::Sound, [25](#)
SetStreamSource
 Ac::Sound, [26](#)
Start
 Ac::Microphone, [20](#), [21](#)
StreamWaveBuffer
 Ac::AudioStream, [10](#)
Streaming
 Ac::AudioSystem, [15](#), [16](#)
SwapEndianness
 Ac::WaveBuffer, [32](#)

WriteAudioBuffer
 Ac::AudioSystem, [16](#)