NAME

ost::Audio − Generic audio class to hold master data types and various useful class encapsulated friend functions as per GNU Common C++ 2 coding standard.

SYNOPSIS

```
#include <audio2.h>
```

Inherited by **ost::AudioBase**, **ost::AudioCodec**, **ost::AudioResample**, **ost::AudioTone**, and **ost::DTMFDetect**.

Public Types

enum **Rate** { **rateUnknown**, **rate6khz** = 6000, **rate8khz** = 8000, **rate16khz** = 16000, **rate32khz** = 32000, **rate44khz** = 44100 }
Audio encoding rate, samples per second.

enum **Mode** { **modeRead**, **modeReadAny**, **modeReadOne**, **modeWrite**, **modeCache**, **modeInfo**, **modeFeed**, **modeAppend**, **modeCreate** }
File processing mode, whether to skip missing files, etc.

enum **Encoding** { **unknownEncoding** = 0, **g721ADPCM**, **g722Audio**, **g722_7bit**, **g722_6bit**, **g723_2bit**, **g723_3bit**, **g723_5bit**, **gsmVoice**, **msgsmVoice**, **mulawAudio**, **alawAudio**, **mp1Audio**, **mp2Audio**, **mp3Audio**, **okiADPCM**, **voxADPCM**, **sx73Voice**, **sx96Voice**, **cdaStereo**, **cdaMono**, **pcm8Stereo**, **pcm8Mono**, **pcm16Stereo**, **pcm16Mono**, **pcm32Stereo**, **pcm32Mono**, **speexVoice**, **speexAudio**, **g729Audio**, **ilbcAudio**, **speexUltra**, **speexNarrow** = speexVoice, **speexWide** = speexAudio, **g723_4bit** = g721ADPCM }
Audio *encoding formats.*

enum **Format** { **raw**, **snd**, **riff**, **mpeg**, **wave** }
Audio *container file format.*

enum **DeviceMode** { **PLAY**, **RECORD**, **PLAYREC** }
Audio *device access mode.*

enum **Error** { **errSuccess** = 0, **errReadLast**, **errNotOpened**, **errEndOfFile**, **errStartOfFile**, **errRateInvalid**, **errEncodingInvalid**, **errReadInterrupt**, **errWriteInterrupt**, **errReadFailure**, **errWriteFailure**, **errReadIncomplete**, **errWriteIncomplete**, **errRequestInvalid**, **errTOCFailed**, **errStatFailed**, **errInvalidTrack**, **errPlaybackFailed**, **errNotPlaying**, **errNoCodec** }
Audio *error conditions.*

typedef int16_t **snd16_t**
typedef int32_t **snd32_t**
typedef int16_t **Level**
typedef int16_t **Sample**
typedef int16_t * **Linear**
typedef unsigned long **timeout_t**
typedef unsigned char * **Encoded**
typedef enum **Rate Rate**
typedef enum **Mode Mode**
typedef enum **Encoding Encoding**
typedef enum **Format Format**
typedef enum **DeviceMode DeviceMode**
typedef enum **Error Error**

Static Public Member Functions

static **Level tolevel** (float dbm)
Convert dbm power level to integer value (0-32768).

static float **todbm** (**Level** power)
Convert integer power levels to dbm.

static bool **hasDevice** (unsigned device=0)
Test for the presense of a specified (indexed) audio device.

static **AudioDevice** * **getDevice** (unsigned device=0, **DeviceMode** mode=PLAY)
Get a audio device object that can be used to play or record audio.

static const char * **getCodecPath** (void)
Get pathname to where loadable codec modules are stored.

static const char * **getMIME** (**Info** &info)

Get the mime descriptive type for a given **Audio** *encoding description, usually retrieved from a newly opened audio file.*

static const char * **getName** (**Encoding** encoding)
Get the short ascii description used for the given audio encoding type.

static const char * **getExtension** (**Encoding** encoding)
Get the preferred file extension name to use for a given audio encoding type.

static **Encoding getEncoding** (const char *name)
Get the audio encoding format that is specified by a short ascii name.

static **Encoding getStereo** (**Encoding** encoding)
Get the stereo encoding format associated with the given format.

static **Encoding getMono** (**Encoding** encoding)
Get the mono encoding format associated with the given format.

static bool **isLinear** (**Encoding** encoding)
Test if the audio encoding format is a linear one.

static bool **isBuffered** (**Encoding** encoding)
Test if the audio encoding format must be packetized (that is, has irregular sized frames) and must be processed only through buffered codecs.

static bool **isMono** (**Encoding** encoding)
Test if the audio encoding format is a mono format.

static bool **isStereo** (**Encoding** encoding)
Test if the audio encoding format is a stereo format.

static **Rate getRate** (**Encoding** encoding)
Return default sample rate associated with the specified audio encoding format.

static **Rate getRate** (**Encoding** e, **Rate** request)
Return optional rate setting effect.

static **timeout_t getFraming** (**Encoding** encoding, **timeout_t** timeout=0)
Return frame timing for an audio encoding format.

static **timeout_t getFraming** (**Info** &info, **timeout_t** timeout=0)
Return frame time for an audio source description.

static bool **isEndian** (**Encoding** encoding)
Test if the endian byte order of the encoding format is different from the machine's native byte order.

static bool **isEndian** (**Info** &info)
Test if the endian byte order of the audio source description is different from the machine's native byte order.

static bool **swapEndian** (**Encoding** encoding, void *buffer, unsigned number)
Optionally swap endian of audio data if the encoding format endian byte order is different from the machine's native endian.

static void **swapEncoded** (**Info** &info, **Encoded** data, size_t bytes)
Optionally swap endian of encoded audio data based on the audio encoding type, and relationship to native byte order.

static bool **swapEndian** (**Info** &info, void *buffer, unsigned number)
Optionally swap endian of audio data if the audio source description byte order is different from the machine's native endian byte order.

static **Level getImpulse** (**Encoding** encoding, void *buffer, unsigned number)
Get the energey impulse level of a frame of audio data.

static **Level getImpulse** (**Info** &info, void *buffer, unsigned number=0)
Get the energey impulse level of a frame of audio data.

static **Level getPeak** (**Encoding** encoding, void *buffer, unsigned number)
Get the peak (highest energy) level found in a frame of audio data.

static **Level getPeak** (**Info** &info, void *buffer, unsigned number=0)
Get the peak (highest energy) level found in a frame of audio data.

static void **toTimestamp** (**timeout_t** duration, char *address, size_t size)
Provide ascii timestamp representation of a timeout value.

static **timeout_t toTimeout** (const char *timestamp)
Convert ascii timestamp representation to a timeout number.

static int **getFrame** (**Encoding** encoding, int samples=0)
Returns the number of bytes in a sample frame for the given encoding type, rounded up to the nearest integer.

static int **getCount** (**Encoding** encoding)
Returns the number of samples in all channels for a frame in the given encoding.
static unsigned long **toSamples** (**Encoding** encoding, size_t bytes)
Compute byte counts of audio data into number of samples based on the audio encoding format used.
static unsigned long **toSamples** (**Info** &info, size_t bytes)
Compute byte counts of audio data into number of samples based on the audio source description used.
static size_t **toBytes** (**Info** &info, unsigned long number)
Compute the number of bytes a given number of samples in a given audio encoding will occupy.
static size_t **toBytes** (**Encoding** encoding, unsigned long number)
Compute the number of bytes a given number of samples in a given audio encoding will occupy.
static void **fill** (unsigned char *address, int number, **Encoding** encoding)
Fill an audio buffer with 'empty' (silent) audio data, based on the audio encoding format.
static bool **loadPlugin** (const char *path)
Load a dso plugin (codec plugin), used internally.
static size_t **maxFramesize** (**Info** &info)
Maximum framesize for a given coding that may be needed to store a result.

Static Public Attributes

static const unsigned **ndata**

Classes

struct **dtmf_detect_state_t**
struct **goertzel_state_t**
class **Info**
Audio *source description.*
struct **mpeg_audio**
struct **mpeg_tagv1**
struct **tone_detection_descriptor_t**

Detailed Description

Generic audio class to hold master data types and various useful class encapsulated friend functions as per GNU Common C++ 2 coding standard.

Author:

David Sugar <dyfet AT ostel DOT com> Master audio class.

Member Typedef Documentation

typedef int16_t ost::Audio::snd16_t
typedef int32_t ost::Audio::snd32_t
typedef int16_t ost::Audio::Level
typedef int16_t ost::Audio::Sample
typedef int16_t* ost::Audio::Linear
typedef unsigned long ost::Audio::timeout_t
typedef unsigned char* ost::Audio::Encoded
typedef enum Rate ost::Audio::Rate
typedef enum Mode ost::Audio::Mode
typedef enum Encoding ost::Audio::Encoding
typedef enum Format ost::Audio::Format
typedef enum DeviceMode ost::Audio::DeviceMode
typedef enum Error ost::Audio::Error

Member Enumeration Documentation

enum ost::Audio::Rate

Audio encoding rate, samples per second.

Enumerator:

rateUnknown

rate6khz

rate8khz

rate16khz

rate32khz

rate44khz

enum ost::Audio::Mode

File processing mode, whether to skip missing files, etc.

Enumerator:

modeRead

modeReadAny

modeReadOne

modeWrite

modeCache

modeInfo

modeFeed

modeAppend

modeCreate

enum ost::Audio::Encoding

Audio encoding formats.

Enumerator:

unknownEncoding

g721ADPCM

g722Audio

g722_7bit

g722_6bit

g723_2bit

g723_3bit

g723_5bit

gsmVoice

msgsmVoice

mulawAudio

alawAudio

mp1Audio

mp2Audio

mp3Audio

okiADPCM

voxADPCM

sx73Voice

sx96Voice

cdaStereo

cdaMono

pcm8Stereo

pcm8Mono

pcm16Stereo

pcm16Mono

pcm32Stereo

pcm32Mono

speexVoice

speexAudio

g729Audio

ilbcAudio

speexUltra

speexNarrow

speexWide

g723_4bit

enum ost::Audio::Format

Audio container file format.

Enumerator:

raw

snd

riff

mpeg

wave

enum ost::Audio::DeviceMode

Audio device access mode.

Enumerator:

PLAY

RECORD

PLAYREC

enum ost::Audio::Error

Audio error conditions.

Enumerator:

errSuccess

errReadLast

errNotOpened

errEndOfFile

errStartOfFile

errRateInvalid

errEncodingInvalid

errReadInterrupt

errWriteInterrupt

errReadFailure

errWriteFailure

errReadIncomplete

errWriteIncomplete

errRequestInvalid

errTOCFailed

errStatFailed

errInvalidTrack

errPlaybackFailed

errNotPlaying

errNoCodec

Member Function Documentation

static Level ost::Audio::tolevel (float dbm) `[static]`

Convert dbm power level to integer value (0-32768).

Parameters:

dbm power level

Returns:

integer value.

static float ost::Audio::todbm (Level power) `[static]`

Convert integer power levels to dbm.

Parameters:

power level.

Returns:

dbm power level.

static bool ost::Audio::hasDevice (unsigned device = `0`**)** `[static]`

Test for the presense of a specified (indexed) audio device.

This is normally used to test for local soundcard access.

Parameters:

device index or 0 for default audio device.

Returns:

true if device exists.

static AudioDevice* ost::Audio::getDevice (unsigned device = `0`**, DeviceMode mode** = `PLAY`**)** `[static]`

Get a audio device object that can be used to play or record audio.

This is normally a local soundcard, though an abstract base class is returned, so the underlying device may be different.

Parameters:

device index or 0 for default audio device.
mode of device; play, record, or full duplex.

Returns:

pointer to abstract audio device object interface class.

static const char* ost::Audio::getCodecPath (void) `[static]`

Get pathname to where loadable codec modules are stored.

Returns:

file path to loadable codecs.

static const char* ost::Audio::getMIME (Info & info) `[static]`

Get the mime descriptive type for a given **Audio** encoding description, usually retrieved from a newly opened audio file.

Parameters:

info source description object

Returns:

text of mime type to use for this audio source.

static const char* ost::Audio::getName (Encoding encoding) `[static]`

Get the short ascii description used for the given audio encoding type.

Parameters:

encoding format.

Returns:

ascii name of encoding format.

static const char* ost::Audio::getExtension (Encoding encoding) `[static]`

Get the preferred file extension name to use for a given audio encoding type.

Parameters:

encoding format.

Returns:

ascii file extension to use.

static Encoding ost::Audio::getEncoding (const char * name) `[static]`

Get the audio encoding format that is specified by a short ascii name.

This will either accept names like those returned from **getName()**, or .xxx file extensions, and return the audio encoding type associated with the name or extension.

Parameters:

name of encoding or file extension.

Returns:

audio encoding format.

See also:

getName

static Encoding ost::Audio::getStereo (Encoding encoding) `[static]`

Get the stereo encoding format associated with the given format.

Parameters:

encoding format being tested for stereo.

Returns:

associated stereo audio encoding format.

static Encoding ost::Audio::getMono (Encoding encoding) `[static]`

Get the mono encoding format associated with the given format.

Parameters:

encoding format.

Returns:

associated mono audio encoding format.

static bool ost::Audio::isLinear (Encoding encoding) `[static]`

Test if the audio encoding format is a linear one.

Returns:

true if encoding format is linear audio data.

Parameters:

encoding format.

static bool ost::Audio::isBuffered (Encoding encoding) `[static]`
Test if the audio encoding format must be packetized (that is, has irregular sized frames) and must be processed only through buffered codecs.

Returns:
true if packetized audio.

Parameters:
encoding format.

static bool ost::Audio::isMono (Encoding encoding) `[static]`
Test if the audio encoding format is a mono format.

Returns:
true if encoding format is mono audio data.

Parameters:
encoding format.

static bool ost::Audio::isStereo (Encoding encoding) `[static]`
Test if the audio encoding format is a stereo format.

Returns:
true if encoding format is stereo audio data.

Parameters:
encoding format.

static Rate ost::Audio::getRate (Encoding encoding) `[static]`
Return default sample rate associated with the specified audio encoding format.

Returns:
sample rate for audio data.

Parameters:
encoding format.

static Rate ost::Audio::getRate (Encoding e, Rate request) `[static]`
Return optional rate setting effect.

Many codecs are fixed rate.

Returns:
result rate for audio date.

Parameters:
encoding format.
requested rate.

static timeout_t ost::Audio::getFraming (Encoding encoding, timeout_t timeout = 0) `[static]`
Return frame timing for an audio encoding format.

Returns:
frame time to use in milliseconds.

Parameters:
encoding of frame to get timing segment for.
timeout of frame time segment to request.

static timeout_t ost::Audio::getFraming (Info & info, timeout_t timeout = 0) `[static]`
Return frame time for an audio source description.

Returns:
frame time to use in milliseconds.

Parameters:
info descriptor of frame encoding to get timing segment for.
timeout of frame time segment to request.

static bool ost::Audio::isEndian (Encoding encoding) `[static]`
Test if the endian byte order of the encoding format is different from the machine's native byte order.

Returns:
true if endian format is different.

Parameters:
encoding format.

static bool ost::Audio::isEndian (Info & info) `[static]`
Test if the endian byte order of the audio source description is different from the machine's native byte order.

Returns:
true if endian format is different.

Parameters:
info source description object.

static bool ost::Audio::swapEndian (Encoding encoding, void * buffer, unsigned number) `[static]`
Optionally swap endian of audio data if the encoding format endian byte order is different from the machine's native endian.

Returns:
true if endian format was different.

Parameters:
encoding format of data.
buffer of audio data.
number of audio samples.

static void ost::Audio::swapEncoded (Info & info, Encoded data, size_t bytes) `[static]`
Optionally swap endian of encoded audio data based on the audio encoding type, and relationship to native byte order.

Parameters:
info source description of object.
buffer of audio data.
number of bytes of audio data.

static bool ost::Audio::swapEndian (Info & info, void * buffer, unsigned number) `[static]`
Optionally swap endian of audio data if the audio source description byte order is different from the machine's native endian byte order.

Returns:
true if endian format was different.

Parameters:
info source description object of data.
buffer of audio data.
number of audio samples.

static Level ost::Audio::getImpulse (Encoding encoding, void * buffer, unsigned number) `[static]`
Get the energey impulse level of a frame of audio data.

Returns:
impulse energy level of audio data.

Parameters:
encoding format of data to examine.
buffer of audio data to examine.
number of audio samples to examine.

static Level ost::Audio::getImpulse (Info & info, void * buffer, unsigned number = `0`) `[static]`
Get the energey impulse level of a frame of audio data.

Returns:
impulse energy level of audio data.

Parameters:
info encoding source description object.
buffer of audio data to examine.

number of audio samples to examine.

static Level ost::Audio::getPeak (Encoding encoding, void * buffer, unsigned number) `[static]`

Get the peak (highest energy) level found in a frame of audio data.

Returns:

peak energy level found in data.

Parameters:

encoding format of data.
buffer of audio data.
number of samples to examine.

static Level ost::Audio::getPeak (Info & info, void * buffer, unsigned number = `0`**)** `[static]`

Get the peak (highest energy) level found in a frame of audio data.

Returns:

peak energy level found in data.

Parameters:

info description object of audio data.
buffer of audio data.
number of samples to examine.

static void ost::Audio::toTimestamp (timeout_t duration, char * address, size_t size) `[static]`

Provide ascii timestamp representation of a timeout value.

Parameters:

duration timeout value
address for ascii data.
size of ascii data.

static timeout_t ost::Audio::toTimeout (const char * timestamp) `[static]`

Convert ascii timestamp representation to a timeout number.

Parameters:

timestamp ascii data.

Returns:

timeout_t duration from data.

static int ost::Audio::getFrame (Encoding encoding, int samples = `0`**)** `[static]`

Returns the number of bytes in a sample frame for the given encoding type, rounded up to the nearest integer.

A frame is defined as the minimum number of bytes necessary to create a point or points in the output waveform for all output channels. For example, 16-bit mono PCM has a frame size of two (because those two bytes constitute a point in the output waveform). GSM has it’s own definition of a frame which involves decompressing a sequence of bytes to determine the final points on the output waveform. The minimum number of bytes you can feed to the decompression engine is 32.5 (260 bits), so this function will return 33 (because we round up) given an encoding type of GSM. Other compressed encodings will return similar results. Be prepared to deal with nonintuitive return values for rare encodings.

Parameters:

encoding The encoding type to get the frame size for.
samples Reserved. Use zero.

Returns:

The number of bytes in a frame for the given encoding.

static int ost::Audio::getCount (Encoding encoding) `[static]`

Returns the number of samples in all channels for a frame in the given encoding.

For example, pcm32Stereo has a frame size of 8 bytes: Note that different codecs have different definitions of a frame - for example, compressed encodings have a rather large frame size relative to the sample size due to the way bytes are fed to the decompression engine.

Parameters:

encoding The encoding to calculate the frame sample count for.

Returns:
samples The number of samples in a frame of the given encoding.

static unsigned long ost::Audio::toSamples (Encoding encoding, size_t bytes) `[static]`
Compute byte counts of audio data into number of samples based on the audio encoding format used.

Returns:
number of audio samples in specified data.

Parameters:
encoding format.
bytes of data.

static unsigned long ost::Audio::toSamples (Info & info, size_t bytes) `[static]`
Compute byte counts of audio data into number of samples based on the audio source description used.

Returns:
number of audio samples in specified data.

Parameters:
info encoding source description.
bytes of data.

static size_t ost::Audio::toBytes (Info & info, unsigned long number) `[static]`
Compute the number of bytes a given number of samples in a given audio encoding will occupy.

Returns:
number of bytes samples will occupy.

Parameters:
info encoding source description.
number of samples.

static size_t ost::Audio::toBytes (Encoding encoding, unsigned long number) `[static]`
Compute the number of bytes a given number of samples in a given audio encoding will occupy.

Returns:
number of bytes samples will occupy.

Parameters:
encoding format.
number of samples.

static void ost::Audio::fill (unsigned char * address, int number, Encoding encoding) `[static]`
Fill an audio buffer with 'empty' (silent) audio data, based on the audio encoding format.

Parameters:
address of data to fill.
number of samples to fill.
encoding format of data.

static bool ost::Audio::loadPlugin (const char * path) `[static]`
Load a dso plugin (codec plugin), used internally.

Returns:
true if loaded.

Parameters:
path to codec.

static size_t ost::Audio::maxFramesize (Info & info) `[static]`
Maximum framesize for a given coding that may be needed to store a result.

Parameters:
info source description object.

Returns:
maximum possible frame size to allocate for encoded data.

Member Data Documentation

const unsigned ost::Audio::ndata `[static]`

Author

Generated automatically by Doxygen for ccAudio from the source code.