Forschungspraktikum - naosoundextractor

Generated by Doxygen 1.8.8

Tue Mar 3 2015 01:36:16

Contents

Chapter 1

NaoSoundExtractor

Author

Bernhard Gäde

1.1 Abstract

NaoSoundExtractor is a C++ framework to transfer audio data from the NAO robot via the network to a remote computer for further processing.

1.2 Overview

1.2.1 Task

Some funny text

1.2.2 Code structure

even more funny text

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

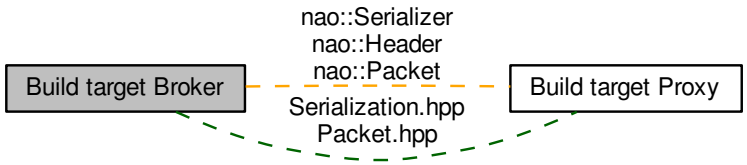
Chapter 6

Module Documentation

6.1 Build target Broker

The build target broker produces a "module" which can be seen as an extension to NAO's operating system.

Collaboration diagram for Build target Broker:



Files

- file main_broker.cpp

  Main file for target Build target Broker.
- file Serialization.hpp

  Contains the definition of class nao::Serializer.
- file AudioDataBroker.hpp

  Contains the declaration of class AudioDataBroker.
- file Packet.hpp

  Contains the definition of class nao::Packet and struct nao::Header.

Classes

- struct nao::ALSoundProcessingParams

  Helper-struct to bundle parameters.
- class nao::Serializer

  Simple serializer used to insert header information into packets.
- class nao::AudioDataBroker
- struct nao::Header

Header struct for class Packet.

- class nao::Packet

  Packet type which is used for data transfer between proxy and client

6.1.1 Detailed Description

Depending on the preprocessor switch IS_REMOTE, a module named NaoSoundProcessor is produced that either runs locally on the robot or, as an stand-alone executable, on a remote computer. Once started, the module will remain loaded into NAO's operating system NAOqi and overwrite the default sound processing behaviour. The sound data captured by NAO's microphones will be handed over to the AudioDataBroker, which packs the raw data together with some information on the recording parameters into packets and delivers those via a simple TcpServer to the AudioDataProxy.

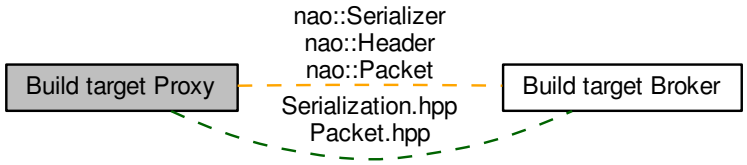
Note

In order to create a local module, the compiler must be instructed to create a shared library and to export the _createModule symbol. Please refer to the Aldebaran C++ SDK documentation.

6.2 Build target Proxy

TODO.

Collaboration diagram for Build target Proxy:



Files

- file ThreadsafeQueue.hpp

  Contains the definition of class nao::ThreadsafeQueue.
- file ExceptionTcpClient.hpp

  Contains the definition of class nao::ExceptionTcpClient.
- file alsoundprocessing.hpp

  Contains the declaration of class nao::AlSoundProcessing and struct nao::AlSoundProcessingParams.
- file AudioFormatBase.hpp

  Contains the definition of class nao::AudioFormatBase.
- file has_member_in.hpp

  Contains the definition of makro GEN_HAS_MEMBER_IN(MEMBER, NAMESPACE)
- file Serialization.hpp

  Contains the definition of class nao::Serializer.
- file AudioDataProxy.hpp

  Contains the definition of class nao::ExceptionTcpClient.
- file AudioFormatAldebaranBase.hpp

  Contains the definition of class nao::AudioFormatAldebaranBase.
- file AudioData.hpp

  Contains the definition of class nao::AudioData.
- file TcpClient.hpp

  Contains the declaration of class TcpClient.
- file AudioFormatAldebaran.hpp

  Contains the definitions of classes AudioFormatAldebaranAll, AudioFormatAldebaranFront, AudioFormatAldebaran↩ Left AudioFormatAldebaranRight and AudioFormatAldebaranRear.
- file exception_enabled_unique_ptr.hpp

  Contains the definition of class nao::exception_enabled_unique_ptr.
- file ExceptionTimeout.hpp

  Contains the definition of class nao::ExceptionTimeout.
- file ExceptionBadFileFormat.hpp

  Contains the definition of class nao::ExceptionBadFileFormat.
- file ExceptionFileNotFound.hpp

  Contains the definition of class nao::ExceptionFileNotFound.
- file AudioFormat.hpp

Include file for grouping all defined audio formats.

- file ClientState.hpp

  Contains the definition of enum ClientState.
- file Packet.hpp

  Contains the definition of class nao::Packet and struct nao::Header.
- file ThreadsafeQueue_fwd.hpp

  Contains a forward declaration of class ThreadsafeQueue.
- file AudioFormatSingleChannel.hpp

  Contains the definition of class nao::AudioFormatSingleChannel.

Classes

- class nao::ThreadsafeQueue< T >

  A basic thread-safe std::queue wrapper object.
- class nao::ExceptionTcpClient

  Exception class for reporting errors within the TcpClient code.
- class nao::AudioFormatBase

  Base class for all audio formats.
- class nao::Serializer

  Simple serializer used to insert header information into packets.
- class nao::AudioDataProxy

  Proxy class for receiving audio data.
- class nao::AudioFormatAldebaran

  Base class for audio formats which can be processed by the AudioDataProxy.
- class nao::AudioData< FMT >

  A container for audiodata.
- class nao::TcpClient

  A small TCP Client for the connection to the AudioDatabroker.
- class nao::AudioFormatAldebaranAll

  An audio format for all existing NAO-Channels.
- class nao::AudioFormatAldebaranFront

  An audio format for the front microphone channel only.
- class nao::AudioFormatAldebaranLeft

  An audio format for the left microphone channel only.
- class nao::AudioFormatAldebaranRight

  An audio format for the right microphone channel only.
- class nao::AudioFormatAldebaranRear

  An audio format for the rear microphone channel only.
- class nao::exception_enabled_unique_ptr< T, D >

  std::unique_ptr with the ability to store exceptions.
- class nao::ExceptionTimeout

  Exception class for reporting timeouts.
- class nao::ExceptionBadFileFormat

  Exception class for reporting incorrect file formats.
- class nao::ExceptionFileNotFound

  Exception class for reporting missing files.
- struct nao::Header

  Header struct for class Packet.
- class nao::Packet

  Packet type which is used for data transfer between proxy and client
- class nao::AudioFormatSingleChannel

  An exemplary audio format for a single channel.

Enumerations

- enum **Channels** { **FRONT** = 0, **LEFT**, **RIGHT**, **REAR** }
- enum **Channels** { **FRONT** = 0 }
- enum **Channels** { **LEFT** = 0 }
- enum **Channels** { **RIGHT** = 0 }
- enum **Channels** { **REAR** = 0 }
- enum nao::ClientState { **disconnected**, **disconnecting**, **connecting**, **idle**, **running** }

  Enum values representing the state of the TcpClient.
- enum **Channels** { **Channel** = 0 }

6.2.1 Detailed Description

MORE TODO

Chapter 7

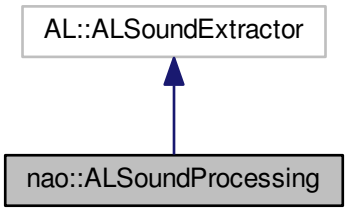
Class Documentation

7.1 nao::ALSoundProcessing Class Reference

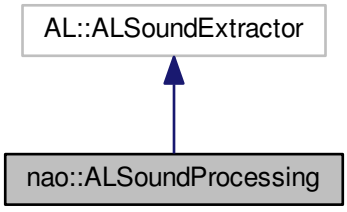
Module class to extend NAO's sound processing capabilities.

`#include <alsoundprocessing.hpp>`

Inheritance diagram for nao::ALSoundProcessing:



Collaboration diagram for nao::ALSoundProcessing:



Public Member Functions

- **ALSoundProcessing** (boost::shared_ptr< AL::ALBroker > pBroker, const ALSoundProcessingParams &params)
- void process (const int &nbOfChannels, const int &nbOfSamplesByChannel, const AL_SOUND_FORMAT *buffer, const AL::ALValue &timeStamp) override

  Method inherited from almodule that is (apparently) called within a seperate thread and accepts sound data to be sent to the client.

Protected Member Functions

- void init ()

  Method inheried form almodule and called within the factory function shortly after construction. It sets up the audio detection parameters and starts detection.

7.1.1 Detailed Description

The class ALSoundProcessing implements the sound extraction module and is therefore derived from ALSound↩ Extractor. It can be seen as a plugin to the naoqi-Operating system with the only purpose of forwarding audio data to the AudioDataBrocker (server).

7.1.2 Member Function Documentation

7.1.2.1 void nao::ALSoundProcessing::process (const int & *nbOfChannels*, const int & *nbOfSamplesByChannel*, const AL_SOUND_FORMAT * *buffer*, const AL::ALValue & *timeStamp*) [override]

Parameters

nbOfChannels	Number of channels recorded by NAO
nbOfSamples↩ ByChannel	Number of samples per channel in current block
buffer	Pointer to a memory block witch contains the audio data
timeStamp	time stamp of the current block (unused)

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

- alsoundprocessing.hpp
- alsoundprocessing.cpp

7.2 nao::ALSoundProcessingParams Struct Reference

Helper-struct to bundle parameters.

`#include <alsoundprocessing.hpp>`

Public Member Functions

- ALSoundProcessingParams (const std::string &parentName, const int &timeout, const int &port)

  Constructor.

Public Attributes

- std::string **parentName_**
- unsigned int **timeout_**
- unsigned int **port_**

7.2.1 Detailed Description

The ALSoundProcessingParams struct bundles multiple parameters for the ALSoundProcessing class. Bundling is necessary as the factory function createModule of the Aldebaran framework accepts only a single parameter.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 nao::ALSoundProcessingParams::ALSoundProcessingParams (const std::string & *parentName*, const int & *timeout*, const int & *port*) [inline]

Parameters

parentName	name of the parent module
timeout	timeout for server functionalities in ms
port	port on which the server should listen

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

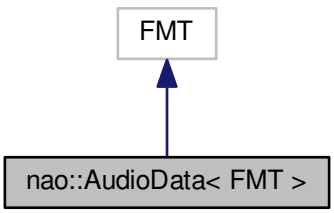
- alsoundprocessing.hpp

7.3 nao::AudioData< FMT > Class Template Reference

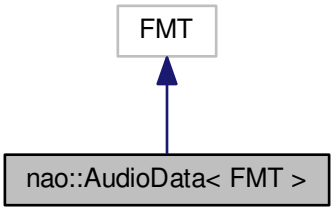
A container for audiodata.

`#include <AudioData.hpp>`

Inheritance diagram for nao::AudioData< FMT >:



Collaboration diagram for nao::AudioData< FMT >:



Public Types

- using **SampleVector** = std::vector< typename FMT::value_type >
- using **ChannelArray** = std::array< SampleVector, FMT::numberOfChannels_ >
- using **ConstChannelIterator** = typename ChannelArray::const_iterator
- using **ChannelIterator** = typename ChannelArray::iterator
- using **ConstSampleIterator** = typename SampleVector::const_iterator
- using **SampleIterator** = typename SampleVector::iterator
- using **value_type** = typename FMT::value_type

Public Member Functions

- AudioData (size_t sampleRate, size_t samplesPerChannel)

  Constructor.
- AudioData (const FMT &format)

  Constructor.
- const FMT & GetAudioFormat () const

  Returns the associated audio format.
- size_t GetNumberOfChannels () const

  Returns the number of Channels.
- SampleVector & GetChannel (size_t index)

  Returns the channel at the given index.
- SampleVector & operator[] (size_t index)

  Index operator, acts like GetNumberOfChannels(), except of checking for a possible violation of the index bounds.
- SampleVector & operator[] (typename FMT::Channels ch)

  Index operator, which uses the channel alias names given by the underlying audio format.
- ChannelIterator begin ()

  Returns an iterator to the first channel in the channel array.
- ChannelIterator end ()

  Returns an iterator to the channel past the last one in the channel array.
- ConstChannelIterator cbegin () const

  Returns a const iterator to the first channel in the channel array.
- ConstChannelIterator cend () const

  Returns an iterator to the channel past the last one in the channel array.

7.3.1 Detailed Description

template<typename FMT>class nao::AudioData< FMT >

This container is built around a std::array<std::vector<FMT::value_type>>. Therefore, the channels can be easily extracted for separate processing. As a disadvantage, the simultaneous access to the same index of different channels will suffer performance losses due to cache effects.

The policy class FMT is used to adjust the container to a specific audio format by setting parameters like the number of channels, the underlying data type and so on. For details refer to the documentation of the class AudioFormatBase.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 template<typename FMT > nao::AudioData< FMT >::AudioData (size_t *sampleRate*, size_t *samplesPerChannel*) [inline]

Sets the length of all channels to samplesPerChannel.

Parameters

sampleRate	sampling Rate
samplesPerChannel	number of samples per channel

7.3.2.2 template<typename FMT> nao::AudioData< FMT >::AudioData (const FMT & *format*) [inline]

Constructs AudioData from the given audio format and sets the length of all channels to the thereby specified length.

Parameters

format	Reference to an audio format object

7.3.3 Member Function Documentation

7.3.3.1 template<typename FMT > AudioData< FMT >::ChannelIterator nao::AudioData< FMT >::begin ( ) [inline]

Returns

iterator to the first channel

7.3.3.2 template<typename FMT > AudioData< FMT >::ConstChannelIterator nao::AudioData< FMT >::cbegin ( ) const [inline]

Returns

const iterator to the first channel

7.3.3.3 template<typename FMT > AudioData< FMT >::ConstChannelIterator nao::AudioData< FMT >::cend ( ) const [inline]

Returns

const iterator to the channel past the last one

7.3.3.4 template<typename FMT > AudioData< FMT >::ChannelIterator nao::AudioData< FMT >::end ( ) [inline]

Returns

iterator to the channel past the last one

7.3.3.5 template<typename FMT> const FMT& nao::AudioData< FMT >::GetAudioFormat ( ) const

Returns

reference to the associated audio format

7.3.3.6 template<typename FMT > AudioData< FMT >::SampleVector & nao::AudioData< FMT >::GetChannel (size_t *index*) [inline]

Parameters

index	channel index

Returns

vector of samples

7.3.3.7 template<typename FMT > size_t nao::AudioData< FMT >::GetNumberOfChannels ( ) const [inline]

Returns

number of channels

7.3.3.8 template<typename FMT > AudioData< FMT >::SampleVector & nao::AudioData< FMT >::operator[] (size_t *index*) [inline]

Parameters

index	channel index

Returns

vector of samples

7.3.3.9 template<typename FMT> AudioData< FMT >::SampleVector & nao::AudioData< FMT >::operator[] (typename FMT::Channels *ch*) [inline]

Parameters

ch	channel alias name (class enum type inherited from FMT)

Returns

vector of samples

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

- AudioData.hpp

7.4 nao::AudioDataBroker Class Reference

`#include <AudioDataBroker.hpp>`

Public Types

- using **PacketPtr** = boost::shared_ptr< nao::Packet >
- using **PacketQueue** = std::queue< PacketPtr >

Public Member Functions

- **AudioDataBroker** (const int port, const unsigned int waitTimeout_ms)
- void Transmit (signed short *data, const int &nbOfChannels, const int &nbOfSamplesByChannel)

7.4.1 Detailed Description

TCP-server for transmitting audio data over the network

The class AudioDataBroker implements a server based on boost::asio. Incoming raw audio data is put into packets (see Packet.hpp) including some additional information on sample rate, samples per channels and so on. Afterwards these packets are queued for transmission via network.

7.4.2 Member Function Documentation

7.4.2.1 void nao::AudioDataBroker::Transmit (signed short * *data*, const int & *nbOfChannels*, const int & *nbOfSamplesByChannel*)

Creates a new packet from the given parameters and inserts it into the output queue.

Parameters

data	memory block containing audio data
nbOfChannels	number of channels represented by the memory block
nbOfSamples↩ ByChannel	number of samples per channel

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

- AudioDataBroker.hpp
- AudioDataBroker.cpp

7.5 nao::AudioDataProxy Class Reference

Proxy class for receiving audio data.

`#include <AudioDataProxy.hpp>`

Public Member Functions

- AudioDataProxy (const std::string &ip, const int port)

  Constructor which connects to the broker.
- AudioDataProxy ()

  Default constructor, does not connect to the broker. For doing so, Connect() has to be called.

- virtual ~AudioDataProxy ()

  Destructor.
- void Connect (const std::string &ip, const int port)

  Establishes a connection to the broker.
- void Disconnect ()

  Closes the connection to the broker.
- bool Connected ()

  Indicates whether the the proxy is connected to the broker or not.
- bool Paused ()

  Indicates whether the the proxy is paused or not.
- bool Running ()

  Indicates whether the the proxy is running (connected and accepting data) to the broker or not.
- void Pause ()

  Pauses the proxy so that it is still connected, but will not push any new packet into the input buffer.
- void Resume ()

  Allows the proxy to push packets into the queue.
- void Start ()

  Allows the proxy to push packets into the queue.
- TcpClient::PacketQueue::size_type PacketsInQueue () const

  Returns the length of the input buffer queue.
- void ClearQueue ()

  Clears the input buffer queue.
- void SetOverflowHandler (const TcpClient::OverflowHandler &ovh=DefaultOverflowHandler)

  Function to set a new overflow handler. See class TcpClient.
- TcpClient::OverflowHandler GetOverflowHandler () const

  Returns the currently set overflow handler object.
- template<class FMT >
  std::unique_ptr< AudioData< FMT > > WaitForData ()

  Blocks the active thread while waiting for a new Packet, which will be returned as an owning smart pointer. If no packet is available within the timeout period, a default constructed smart pointer (~ nullptr) will be returned.
- template<class FMT >
  std::unique_ptr< AudioData< FMT > > WaitForData (size_t blocksize)

  Same as WaitForData() except for the parameter blocksize, which is used to specify the number of samples per channel within the returned AudioData object.

7.5.1 Detailed Description

This class is a proxy for the TcpClient. It simplifies the interface and adds functions to convert the packets used by TcpClient to an AudioData<FMT> container. Different audio formats can be retrieved by efficient compiler-generated specialized versions of WaitForData(). The number of samples per container can be dynamically set at compile time when using the parametrized Overload of WaitForData(). Otherwise, WaitForData() produces AudioData<FMT> containers with the same length as that of the received Packet.

Examples:

main_proxy.cpp.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 nao::AudioDataProxy::AudioDataProxy (const std::string & *ip*, const int *port*)

Parameters

ip	IP address of the broker
port	port on which the broker is listening

7.5.2.2 nao::AudioDataProxy::AudioDataProxy ( )

7.5.2.3 nao::AudioDataProxy::~AudioDataProxy ( ) [virtual]

7.5.3 Member Function Documentation

7.5.3.1 void nao::AudioDataProxy::ClearQueue ( ) [inline]

7.5.3.2 void nao::AudioDataProxy::Connect (const std::string & *ip*, const int *port*)

Parameters

ip	IP address of the broker
port	port on which the broker is listening

7.5.3.3 bool nao::AudioDataProxy::Connected ( ) [inline]

Returns

true if connected, false otherwise

7.5.3.4 void nao::AudioDataProxy::Disconnect ( ) [inline]

Examples:

main_proxy.cpp.

7.5.3.5 TcpClient::OverflowHandler nao::AudioDataProxy::GetOverflowHandler ( ) const [inline]

Returns

overflow handler object (documented in file TcpClient.hpp)

7.5.3.6 TcpClient::PacketQueue::size_type nao::AudioDataProxy::PacketsInQueue ( ) const [inline]

Returns

length of the input buffer queue

7.5.3.7 void nao::AudioDataProxy::Pause ( ) [inline]

Examples:

main_proxy.cpp.

7.5.3.8 bool nao::AudioDataProxy::Paused ( ) [inline]

Returns

true if paused, false otherwise

7.5.3.9 void nao::AudioDataProxy::Resume ( ) [inline]

7.5.3.10 bool nao::AudioDataProxy::Running ( ) [inline]

Returns

true if running, false otherwise

7.5.3.11 void nao::AudioDataProxy::SetOverflowHandler (const TcpClient::OverflowHandler & *ovh* = DefaultOverflowHandler) [inline]

Parameters

ovh	overflow handler function

7.5.3.12 void nao::AudioDataProxy::Start ( ) [inline]

Examples:

main_proxy.cpp.

7.5.3.13 template<class FMT > std::unique_ptr< AudioData< FMT > > nao::AudioDataProxy::WaitForData ( )

The template parameter FMT specifies the audio format, for which the AudioData container shall be created. The class FMT has to be derived form AudioFormatAldebaranBase.

As opposed to it's parameterized overload, this function always returns an AudioData object with just as many samples per channel as a packet of Aldebaran's own format contains. This improves speed as well as latency.

Note: When called after the parametrized overload, some samples might be discarded in order to synchronize to the block margins.

Returns

smart pointer to an AudioData object

Examples:

main_proxy.cpp.

7.5.3.14 template<class FMT > std::unique_ptr< AudioData< FMT > > nao::AudioDataProxy::WaitForData (size_t *blocksize*)

Parameters

blocksize	desired number of samples per channel

Returns

smart pointer to an AudioData object

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

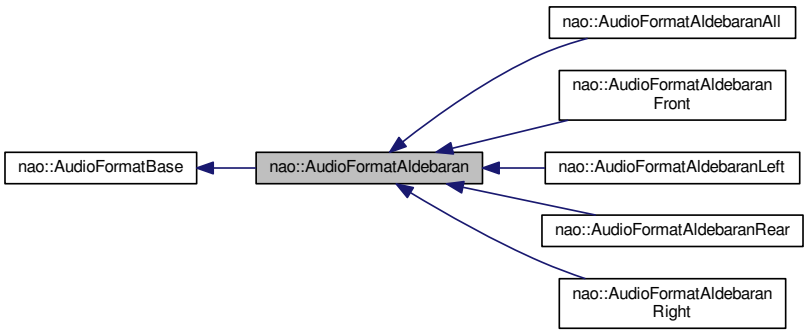
- AudioDataProxy.hpp
- AudioDataProxy.cpp

7.6 nao::AudioFormatAldebaran Class Reference

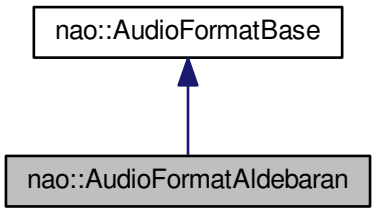
Base class for audio formats which can be processed by the AudioDataProxy.

`#include <AudioFormatAldebaranBase.hpp>`

Inheritance diagram for nao::AudioFormatAldebaran:



Collaboration diagram for nao::AudioFormatAldebaran:



Public Types

- using **value_type** = uint16_t

Protected Member Functions

- AudioFormatAldebaran (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Additional Inherited Members

7.6.1 Detailed Description

The class AudioFormatAldebaran serves as a base class for audio formats which can be treated by AudioDataProxy. Those classes must define an enum class field named "Channels" and a function "size_t GetAssociatedChannel(↩ Channels) const". The enum constants FRONT, LEFT, RIGHT, REAR refer to the corresponding microphones on the NAO robot and serve as index of the channel array of an AudioData<FMT> object. They are also evaluated at compile time in order to generate optimized copy functions (see AudioDataProxy.hpp). Therefore, to implement an AudioFormat which contains the left NAO microphone channel and an arbitrary user defined one, one has to derive from AudioFormatAldebaran and add the field "enum class Channels {LEFT, THE_OTHER_CHANNEL}". AudioDataProxy<FMT> will then detect the channel "LEFT" and fill it with data, leaving "THE_OTHER_CHANNEL" untouched.

As NAO produces unsigned 16-bit samples, value_type is defined accordingly.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 nao::AudioFormatAldebaran::AudioFormatAldebaran (size_t *sampleRate,* size_t *samplesPerChannel*) [inline], [protected]

Parameters

sampleRate	sampling rate
samplesPer↩ Channel	number of samples per channel

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

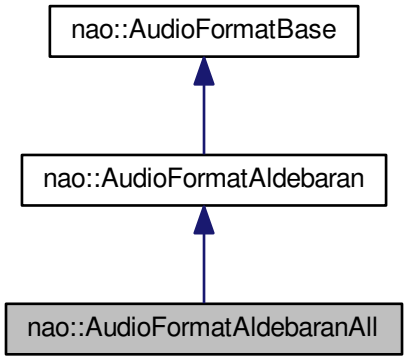
- AudioFormatAldebaranBase.hpp

7.7 nao::AudioFormatAldebaranAll Class Reference

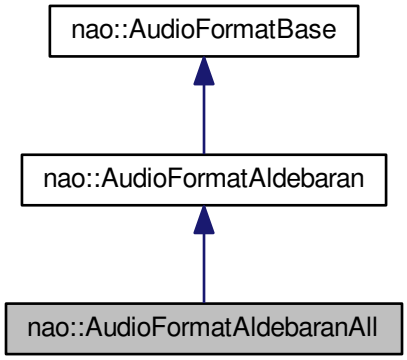
An audio format for all existing NAO-Channels.

`#include <AudioFormatAldebaran.hpp>`

Inheritance diagram for nao::AudioFormatAldebaranAll:



Collaboration diagram for nao::AudioFormatAldebaranAll:



Public Types

- enum **Channels** { **FRONT** = 0, **LEFT**, **RIGHT**, **REAR** }

Public Member Functions

- AudioFormatAldebaranAll (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Static Public Attributes

- static constexpr size_t numberOfChannels_ = 4

Protected Member Functions

- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

7.7.1 Detailed Description

Examples:

main_proxy.cpp.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 nao::AudioFormatAldebaranAll::AudioFormatAldebaranAll (size_t *sampleRate,* size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPer↩ Channel	number of samples per Channel

7.7.3 Member Function Documentation

7.7.3.1 size_t nao::AudioFormatAldebaranAll::GetAssociatedChannel (Channels *ch*) const [inline], [protected]

Parameters

ch	channel alias name

Returns

index in AudioData

7.7.4 Member Data Documentation

7.7.4.1 constexpr size_t nao::AudioFormatAldebaranAll::numberOfChannels_ = 4 [static]

defines the length of the ChannelArray in the class AudioData

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

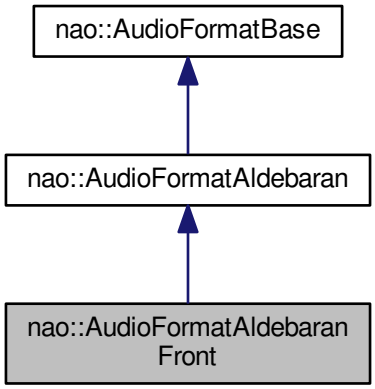
- AudioFormatAldebaran.hpp

7.8 nao::AudioFormatAldebaranFront Class Reference

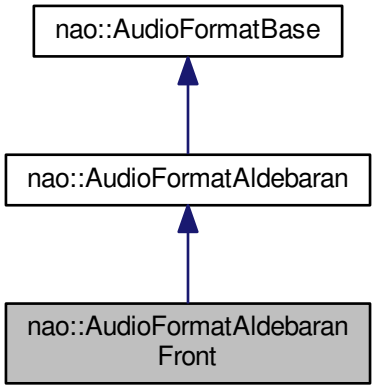
An audio format for the front microphone channel only.

`#include <AudioFormatAldebaran.hpp>`

Inheritance diagram for nao::AudioFormatAldebaranFront:



Collaboration diagram for nao::AudioFormatAldebaranFront:



Public Types

- enum **Channels** { **FRONT** = 0 }

Public Member Functions

- AudioFormatAldebaranFront (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Static Public Attributes

- static const size_t numberOfChannels_ = 1

Protected Member Functions

- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

7.8.1 Constructor & Destructor Documentation

7.8.1.1 nao::AudioFormatAldebaranFront::AudioFormatAldebaranFront (size_t *sampleRate,* size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPerChannel	number of samples per Channel

7.8.2 Member Function Documentation

7.8.2.1 size_t nao::AudioFormatAldebaranFront::GetAssociatedChannel (Channels *ch*) const [inline], [protected]

Parameters

ch	channel alias name

Returns

index in AudioData

7.8.3 Member Data Documentation

7.8.3.1 const size_t nao::AudioFormatAldebaranFront::numberOfChannels_ = 1 [static]

defines the length of the ChannelArray in the class AudioData

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

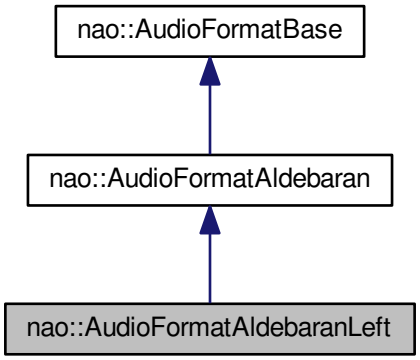
- AudioFormatAldebaran.hpp

7.9 nao::AudioFormatAldebaranLeft Class Reference

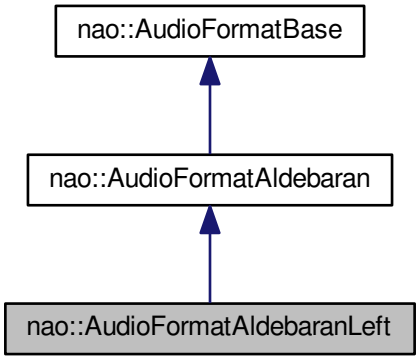
An audio format for the left microphone channel only.

`#include <AudioFormatAldebaran.hpp>`

Inheritance diagram for nao::AudioFormatAldebaranLeft:



Collaboration diagram for nao::AudioFormatAldebaranLeft:



Public Types

- enum **Channels** { **LEFT** = 0 }

Public Member Functions

- AudioFormatAldebaranLeft (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Static Public Attributes

- static const size_t numberOfChannels_ = 1

Protected Member Functions

- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

7.9.1 Constructor & Destructor Documentation

7.9.1.1 nao::AudioFormatAldebaranLeft::AudioFormatAldebaranLeft (size_t *sampleRate*, size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPerChannel	number of samples per Channel

7.9.2 Member Function Documentation

7.9.2.1 size_t nao::AudioFormatAldebaranLeft::GetAssociatedChannel (Channels *ch*) const [inline], [protected]

Parameters

ch	channel alias name

Returns

index in AudioData

7.9.3 Member Data Documentation

7.9.3.1 const size_t nao::AudioFormatAldebaranLeft::numberOfChannels_ = 1 [static]

defines the length of the ChannelArray in the class AudioData

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

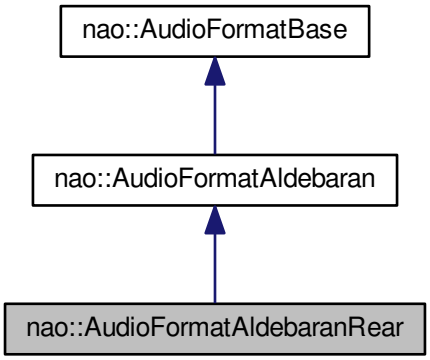
- AudioFormatAldebaran.hpp

7.10 nao::AudioFormatAldebaranRear Class Reference

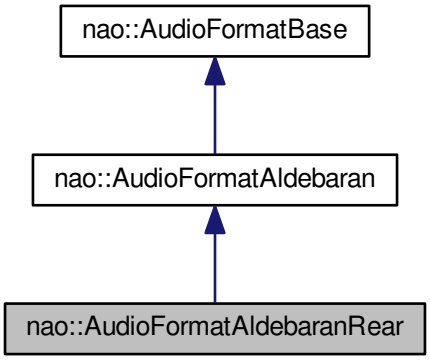
An audio format for the rear microphone channel only.

`#include <AudioFormatAldebaran.hpp>`

Inheritance diagram for nao::AudioFormatAldebaranRear:



Collaboration diagram for nao::AudioFormatAldebaranRear:



Public Types

- enum **Channels** { **REAR** = 0 }

Public Member Functions

- AudioFormatAldebaranRear (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Static Public Attributes

- static const size_t numberOfChannels_ = 1

Protected Member Functions

- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

7.10.1 Constructor & Destructor Documentation

7.10.1.1 nao::AudioFormatAldebaranRear::AudioFormatAldebaranRear (size_t *sampleRate*, size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPerChannel	number of samples per Channel

7.10.2 Member Function Documentation

7.10.2.1 size_t nao::AudioFormatAldebaranRear::GetAssociatedChannel (Channels *ch*) const [inline], [protected]

Parameters

ch	channel alias name

Returns

index in AudioData

7.10.3 Member Data Documentation

7.10.3.1 const size_t nao::AudioFormatAldebaranRear::numberOfChannels_ = 1 [static]

defines the length of the ChannelArray in the class AudioData

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

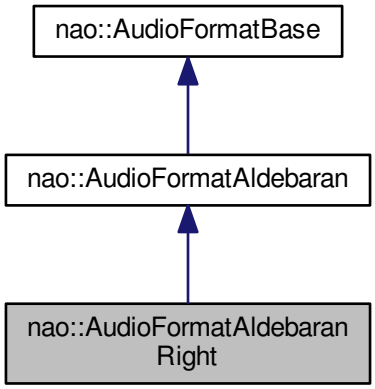
- AudioFormatAldebaran.hpp

7.11 nao::AudioFormatAldebaranRight Class Reference

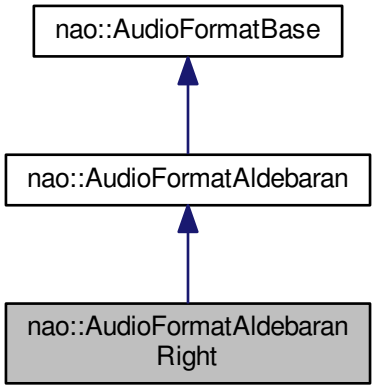
An audio format for the right microphone channel only.

`#include <AudioFormatAldebaran.hpp>`

Inheritance diagram for nao::AudioFormatAldebaranRight:



Collaboration diagram for nao::AudioFormatAldebaranRight:



Public Types

- enum **Channels** { **RIGHT** = 0 }

Public Member Functions

- AudioFormatAldebaranRight (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

Static Public Attributes

- static const size_t numberOfChannels_ = 1

Protected Member Functions

- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

7.11.1 Constructor & Destructor Documentation

7.11.1.1 nao::AudioFormatAldebaranRight::AudioFormatAldebaranRight (size_t *sampleRate*, size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPerChannel	number of samples per Channel

7.11.2 Member Function Documentation

7.11.2.1 size_t nao::AudioFormatAldebaranRight::GetAssociatedChannel (Channels *ch*) const [inline], [protected]

Parameters

ch	channel alias name

Returns

index in AudioData

7.11.3 Member Data Documentation

7.11.3.1 const size_t nao::AudioFormatAldebaranRight::numberOfChannels_ = 1 [static]

defines the length of the ChannelArray in the class AudioData

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

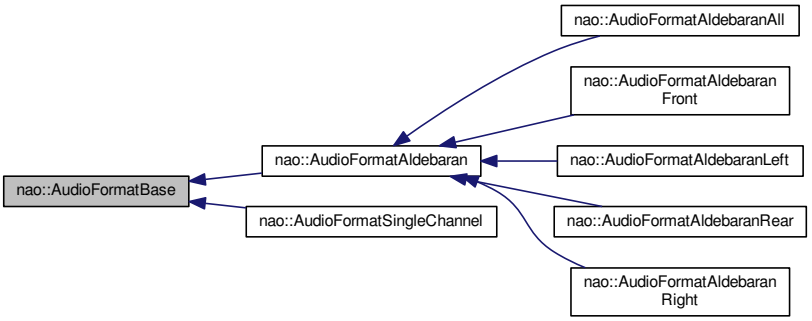
- AudioFormatAldebaran.hpp

7.12 nao::AudioFormatBase Class Reference

Base class for all audio formats.

`#include <AudioFormatBase.hpp>`

Inheritance diagram for nao::AudioFormatBase:



Public Member Functions

- virtual ~AudioFormatBase ()

  Destructor.
- size_t GetSampleRate () const

  Returns the sampling rate represented by this format.
- size_t GetSamplesPerChannel () const

  Returns the number of samples per channel represented by this format.

Protected Member Functions

- AudioFormatBase (size_t sampleRate, size_t samplesPerChannel)

  Constructor.

7.12.1 Detailed Description

This is the base class for all audio formats. They specify the sampling rate, the number of samples per channel and, via the alias declaration value_type, the type for one sample.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 virtual nao::AudioFormatBase::~AudioFormatBase ( ) [inline], [virtual]

7.12.2.2 nao::AudioFormatBase::AudioFormatBase (size_t *sampleRate*, size_t *samplesPerChannel*) [inline], [protected]

Parameters

sampleRate	sampling rate
samplesPer↩ Channel	number of samples per Channel

7.12.3 Member Function Documentation

7.12.3.1 size_t nao::AudioFormatBase::GetSampleRate ( ) const [inline]

Returns

sampling rate

7.12.3.2 size_t nao::AudioFormatBase::GetSamplesPerChannel ( ) const [inline]

Returns

number of samples per channel

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

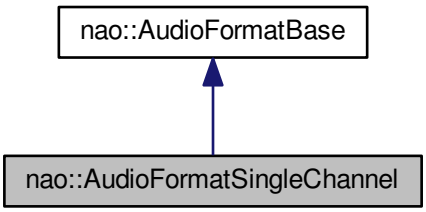
- AudioFormatBase.hpp

7.13 nao::AudioFormatSingleChannel Class Reference

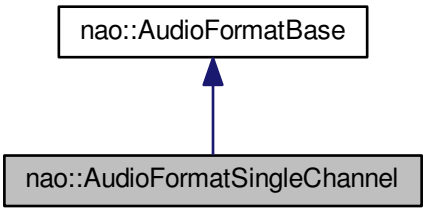
An exemplary audio format for a single channel.

`#include <AudioFormatSingleChannel.hpp>`

Inheritance diagram for nao::AudioFormatSingleChannel:



Collaboration diagram for nao::AudioFormatSingleChannel:



Public Types

- enum **Channels** { **Channel** = 0 }

Public Member Functions

- AudioFormatSingleChannel (size_t sampleRate, size_t samplesPerChannel)

  Constructor.
- size_t GetAssociatedChannel (Channels ch) const

  Translates a channels alias name to an index to address a particular channel in AudioData<FMT>.

Additional Inherited Members

7.13.1 Constructor & Destructor Documentation

7.13.1.1 nao::AudioFormatSingleChannel::AudioFormatSingleChannel (size_t *sampleRate*, size_t *samplesPerChannel*) [inline]

Parameters

sampleRate	sampling rate
samplesPer↩ Channel	number of samples per Channel

7.13.2 Member Function Documentation

7.13.2.1 size_t nao::AudioFormatSingleChannel::GetAssociatedChannel (Channels *ch*) const [inline]

Parameters

ch	channel alias name

Returns

index in AudioData

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

- AudioFormatSingleChannel.hpp

7.14 nao::exception_enabled_unique_ptr< T, D > Class Template Reference

std::unique_ptr with the ability to store exceptions.

`#include <exception_enabled_unique_ptr.hpp>`

Public Types

- using **pointer** = typename std::unique_ptr< T >::pointer
- using **element_type** = T
- using **deleter_type** = D

Public Member Functions

- constexpr exception_enabled_unique_ptr () noexcept=default

  Default Constructor.
- constexpr exception_enabled_unique_ptr (std::nullptr_t np) noexcept

Nullpointer constructor.

- exception_enabled_unique_ptr (std::exception_ptr ep) noexcept

  Exception pointer constructor, sets the internal exception_ptr, which will be rethrown when the smart pointer is dereferenced.

- exception_enabled_unique_ptr (pointer p) noexcept

  Constructor for arbitrary pointers.

- exception_enabled_unique_ptr (pointer p, typename std::remove_reference< D >::type &&del) noexcept

  Constructor with an arbitrary pointer and an rvalue deleter object as arguments.

- exception_enabled_unique_ptr (pointer p, typename std::conditional< std::is_reference< deleter_type >::value, deleter_type, const deleter_type & >::type del) noexcept

  Constructor with an arbitrary pointer and an lvalue deleter object as arguments.

- template<typename T2 , typename D2 >
  exception_enabled_unique_ptr (exception_enabled_unique_ptr< T2, D2 > &&rhs) noexcept

  Pointer-conversion move constructor. Moves the ownership from one exception_enabled_unique_ptr to the other if the managed pointer types are convertible.

- exception_enabled_unique_ptr (const exception_enabled_unique_ptr &)=delete

  Deleted copy constructor.

- exception_enabled_unique_ptr (exception_enabled_unique_ptr &&)=default

  Default move constructor.

- exception_enabled_unique_ptr & operator= (const exception_enabled_unique_ptr &)=delete

  Deleted copy assignment operator.

- exception_enabled_unique_ptr & operator= (std::nullptr_t np) noexcept

  Nullpointer assignment operator.

- exception_enabled_unique_ptr & operator= (std::exception_ptr ep) noexcept

  Exception assignment operator.

- exception_enabled_unique_ptr & operator= (exception_enabled_unique_ptr &&)=default

  Default move assignment operator.

- template<typename T2 , typename D2 >
  exception_enabled_unique_ptr & operator= (exception_enabled_unique_ptr< T2, D2 > &&rhs) noexcept

  Pointer conversion move assignment operator.

- void touch () const

  Checks if the smart pointer stores an exception and, if so, throws it.

- pointer get () const

  Returns the managed raw pointer or throws a stored exception.

- pointer get_noexcept () const noexcept

  Like get(), but never throws.

- std::exception_ptr get_exception () noexcept

  Returns the internal exception pointer (may be uninitialized).

- bool stores_exception () const noexcept

  Indicated whether the exception_enabled_unique_ptr stores an exception or not.

- pointer release ()

  Releases the managed raw pointer. Throws if the exception_enabled_unique_ptr stores an exception.

- deleter_type & get_deleter () noexcept

  Returns a reference to the deleter object.

- const deleter_type & get_deleter () const noexcept

  Returns a reference to the deleter object (const overload).

- void reset (pointer p=pointer()) noexcept

  Resets the smart pointer: the deleter is called for the current managed raw pointer, which is subsequently replaced by p.

- void reset (std::exception_ptr ep) noexcept

  Resets the smart pointer: the deleter is called for the current managed raw pointer, which is subsequently replaced a nullptr. The internal exception_ptr is set to ep.

- std::add_lvalue_reference
  < element_type >::type operator* () const

  Indirection operator. Throws if the exception_enabled_unique_ptr stores an exception.
- pointer operator-> () const

  Structure dereferencing operator. Throws if the exception_enabled_unique_ptr stores an exception.
- operator bool () const noexcept

  Type conversion operator to bool.
- operator std::unique_ptr< T, D > ()

  Type conversion operator to std::unique_ptr. Throws if the exception_enabled_unique_ptr stores an exception.

7.14.1 Detailed Description

template<typename T, typename D = std::default_delete<T>>class nao::exception_enabled_unique_ptr< T, D >

The class exception_enabled_unique_ptr is a wrapper around std::unique_ptr, extended to store exception pointers. It can be used to transfer results as well as exceptions between two threads. For non-exception pointers the behaviour of exception_enabled_unique_ptr remains unchanged compared to std::unique_ptr, except of some exceptions specifications (i.e. the dereferencing operators and the member functions get() and release() may throw). Exceptions pointers can be assigned to the exception_enabled_unique_ptr just like any other pointer, but as soon as one tries to dereference the pointer, the corresponding exceptions are rethrown.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 template<typename T, typename D = std::default_delete<T>> constexpr nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr ( ) [default], [noexcept]

7.14.2.2 template<typename T, typename D = std::default_delete<T>> constexpr nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (std::nullptr_t *np*) [inline], [noexcept]

Parameters

np	nullpointer

7.14.2.3 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (std::exception_ptr *ep*) [inline], [noexcept]

Parameters

ep	pointer to an active exception

7.14.2.4 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (pointer *p*) [inline], [explicit], [noexcept]

Parameters

p	pointer whose ownership shall be taken

7.14.2.5 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (pointer *p*, typename std::remove_reference< D >::type && *del*) [inline], [noexcept]

Parameters

p	pointer whose ownership shall be taken
del	callable deleter object (rvalue)

7.14.2.6 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (pointer *p,* typename std::conditional< std::is_reference< deleter_type >::value, deleter_type, const deleter_type & >::type *del*) [inline], [noexcept]

Parameters

p	pointer whose ownership shall be taken
del	callable deleter object (lvalue)

7.14.2.7 template<typename T, typename D = std::default_delete<T>> template<typename T2 , typename D2 > nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (exception_enabled_unique_ptr< T2, D2 > && *rhs*) [inline], [noexcept]

Parameters

rhs	universal reference to the moved-from smart pointer

7.14.2.8 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (const exception_enabled_unique_ptr< T, D > &) [delete]

7.14.2.9 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::exception_enabled_unique_ptr (exception_enabled_unique_ptr< T, D > &&) [default]

7.14.3 Member Function Documentation

7.14.3.1 template<typename T, typename D = std::default_delete<T>> pointer nao::exception_enabled_unique_ptr< T, D >::get ( ) const [inline]

Returns

managed raw pointer

7.14.3.2 template<typename T, typename D = std::default_delete<T>> deleter_type& nao::exception_enabled_unique_ptr< T, D >::get_deleter ( ) [inline], [noexcept]

Returns

reference to deleter object

7.14.3.3 template<typename T, typename D = std::default_delete<T>> const deleter_type& nao::exception_enabled_unique_ptr< T, D >::get_deleter ( ) const [inline], [noexcept]

Returns

const reference to deleter object

7.14.3.4 template<typename T, typename D = std::default_delete<T>> std::exception_ptr nao::exception_enabled_unique_ptr< T, D >::get_exception ( ) [inline], [noexcept]

Returns

exception pointer

7.14.3.5 template<typename T, typename D = std::default_delete<T>> pointer nao::exception_enabled_unique_ptr< T, D >::get_noexcept ( ) const [inline], [noexcept]

Returns

managed raw pointer

7.14.3.6 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::operator bool ( ) const [inline], [noexcept]

Returns

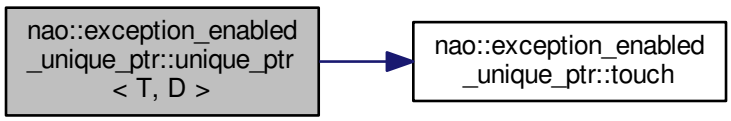
true if the smart pointer is initialized, false otherwise

7.14.3.7 template<typename T, typename D = std::default_delete<T>> nao::exception_enabled_unique_ptr< T, D >::operator std::unique_ptr< T, D > ( ) [inline]

NOTE: Since the ownership is moved form ∗this to the std::unique_ptr, ∗this is altered! This means that the the following line alters the content of an exception_enabled_unique_ptr:

```
static_cast<std::unique_ptr<T>>(exEnPtr);
```

Here is the call graph for this function:



7.14.3.8 template<typename T, typename D = std::default_delete<T>> std::add_lvalue_reference<element_type>::type nao::exception_enabled_unique_ptr< T, D >::operator∗ ( ) const [inline]

Returns

managed raw pointer

7.14.3.9 template<typename T, typename D = std::default_delete<T>> pointer nao::exception_enabled_unique_ptr< T, D >::operator-> ( ) const [inline]

Returns

managed raw pointer

7.14.3.10 template<typename T, typename D = std::default_delete<T>> exception_enabled_unique_ptr& nao::exception_enabled_unique_ptr< T, D >::operator= (const exception_enabled_unique_ptr< T, D > &) [delete]

7.14.3.11 template<typename T, typename D = std::default_delete<T>> exception_enabled_unique_ptr& nao::exception_enabled_unique_ptr< T, D >::operator= (std::nullptr_t *np*) [inline], [noexcept]

Parameters

np	nullpointer

7.14.3.12 template<typename T, typename D = std::default_delete<T>> exception_enabled_unique_ptr& nao::exception_enabled_unique_ptr< T, D >::operator= (std::exception_ptr *ep*) [inline], [noexcept]

Parameters

ep	pointer to an active exception

Returns

reference to ∗this

7.14.3.13 template<typename T, typename D = std::default_delete<T>> exception_enabled_unique_ptr& nao::exception_enabled_unique_ptr< T, D >::operator= (exception_enabled_unique_ptr< T, D > &&) [default]

7.14.3.14 template<typename T, typename D = std::default_delete<T>> template<typename T2 , typename D2 > exception_enabled_unique_ptr& nao::exception_enabled_unique_ptr< T, D >::operator= (exception_enabled_unique_ptr< T2, D2 > && *rhs*) [inline], [noexcept]

Parameters

rhs	moved from smart pointer

Returns

reference to ∗this

7.14.3.15 template<typename T, typename D = std::default_delete<T>> pointer nao::exception_enabled_unique_ptr< T, D >::release ( ) [inline]

Returns

former managed, now owning raw pointer

7.14.3.16 template<typename T, typename D = std::default_delete<T>> void nao::exception_enabled_unique_ptr< T, D >::reset (pointer *p* = `pointer()`) `[inline],[noexcept]`

Parameters

p	new raw pointer to be managed.

7.14.3.17 template<typename T, typename D = std::default_delete<T>> void nao::exception_enabled_unique_ptr< T, D >::reset (std::exception_ptr *ep*) [inline], [noexcept]

Parameters

ep	pointer to an active exception

7.14.3.18 template<typename T, typename D = std::default_delete<T>> bool nao::exception_enabled_unique_ptr< T, D >::stores_exception ( ) const [inline], [noexcept]

Returns

true, if the smart pointer stores an exception, false otherwise

7.14.3.19 template<typename T, typename D = std::default_delete<T>> void nao::exception_enabled_unique_ptr< T, D >::touch ( ) const [inline]

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

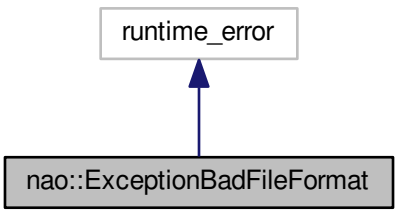
- exception_enabled_unique_ptr.hpp

7.15 nao::ExceptionBadFileFormat Class Reference

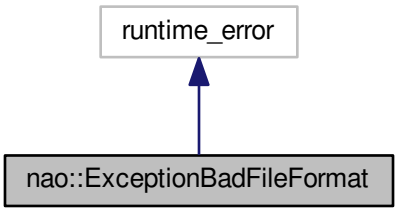
Exception class for reporting incorrect file formats.

`#include <ExceptionBadFileFormat.hpp>`

Inheritance diagram for nao::ExceptionBadFileFormat:



Collaboration diagram for nao::ExceptionBadFileFormat:



Public Member Functions

- ExceptionBadFileFormat (const std::string &whatArg)

  Constructor taking a reference to std::string (representing an error message) as parameter.

- ExceptionBadFileFormat (const char ∗whatArg)

  Constructor taking a pointer to const char (representing an error message) as parameter.

7.15.1 Constructor & Destructor Documentation

7.15.1.1 nao::ExceptionBadFileFormat::ExceptionBadFileFormat (const std::string & *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

7.15.1.2 nao::ExceptionBadFileFormat::ExceptionBadFileFormat (const char ∗ *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

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

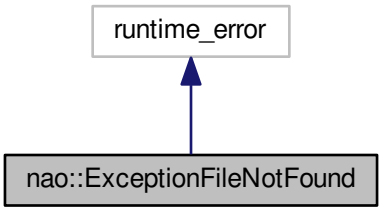
- ExceptionBadFileFormat.hpp

7.16 nao::ExceptionFileNotFound Class Reference

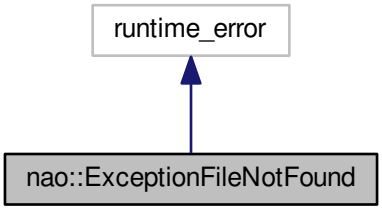
Exception class for reporting missing files.

`#include <ExceptionFileNotFound.hpp>`

Inheritance diagram for nao::ExceptionFileNotFound:



Collaboration diagram for nao::ExceptionFileNotFound:



Public Member Functions

- ExceptionFileNotFound (const std::string &whatArg)

  Constructor taking a reference to std::string (representing an error message) as parameter.
- ExceptionFileNotFound (const char ∗whatArg)

  Constructor taking a pointer to const char (representing an error message) as parameter.

7.16.1 Constructor & Destructor Documentation

7.16.1.1 nao::ExceptionFileNotFound::ExceptionFileNotFound (const std::string & *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

7.16.1.2 nao::ExceptionFileNotFound::ExceptionFileNotFound (const char ∗ *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

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

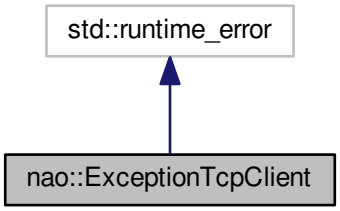
- ExceptionFileNotFound.hpp

7.17 nao::ExceptionTcpClient Class Reference

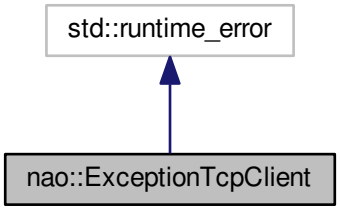
Exception class for reporting errors within the TcpClient code.

`#include <ExceptionTcpClient.hpp>`

Inheritance diagram for nao::ExceptionTcpClient:



Collaboration diagram for nao::ExceptionTcpClient:



Public Member Functions

- ExceptionTcpClient (const std::string &whatArg, ClientState conState)

  Constructor taking a reference to std::string (representing an error message) and the state of the client at which the error occurred as parameters.
- ExceptionTcpClient (const char ∗whatArg, ClientState conState)

  Constructor taking a pointer to const char (representing an error message) and the state of the client at which the error occurred as parameters.

7.17.1 Constructor & Destructor Documentation

7.17.1.1 nao::ExceptionTcpClient::ExceptionTcpClient (const std::string & *whatArg*, ClientState *conState*) [inline]

Parameters

whatArg	error message which shall be returned by what()
conState	state the client was in as the error occurred

7.17.1.2 nao::ExceptionTcpClient::ExceptionTcpClient (const char * *whatArg*, ClientState *conState*) [inline]

Parameters

whatArg	error message which shall be returned by what()
conState	state the client was in as the error occurred

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

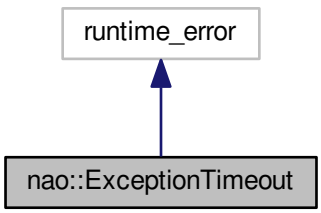
- ExceptionTcpClient.hpp

7.18 nao::ExceptionTimeout Class Reference

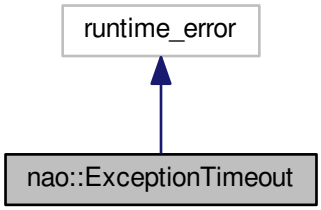
Exception class for reporting timeouts.

`#include <ExceptionTimeout.hpp>`

Inheritance diagram for nao::ExceptionTimeout:



Collaboration diagram for nao::ExceptionTimeout:



Public Member Functions

- ExceptionTimeout (const std::string &whatArg)
  Constructor taking a reference to std::string (representing an error message) as parameter.
- ExceptionTimeout (const char *whatArg)
  Constructor taking a pointer to const char (representing an error message) as parameter.

7.18.1 Constructor & Destructor Documentation

7.18.1.1 nao::ExceptionTimeout::ExceptionTimeout (const std::string & *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

7.18.1.2 nao::ExceptionTimeout::ExceptionTimeout (const char * *whatArg*) [inline], [explicit]

Parameters

whatArg	error message which shall be returned by what()

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

- ExceptionTimeout.hpp

7.19 nao::Header Struct Reference

Header struct for class Packet.

`#include <Packet.hpp>`

Public Member Functions

- void Serialize (Serializer &s)
  Serializes all member variables (except of HeaderSize_) using the given Serializer.
- void Deserialize (Serializer &s)
  Deserializes all member variables (except of HeaderSize_) using the given Serializer.

Public Attributes

- uint8_t protocolVersion_
- uint8_t isError_
- uint8_t numberOfChannels_
- uint32_t samplesPerChannel_
- uint32_t samplingRate_
- uint32_t payloadLength_
- uint8_t reserved_

Static Public Attributes

- static const int HeaderSize_ = 16

7.19.1 Detailed Description

Simple struct for storing the header information variables with two methods to serialize/deserialze them.

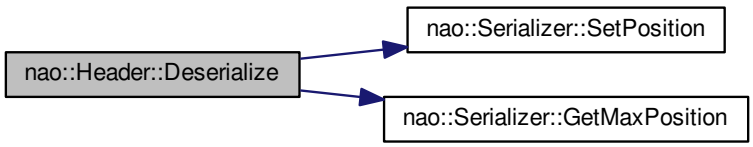
7.19.2 Member Function Documentation

7.19.2.1 void nao::Header::Deserialize (Serializer & *s*) [inline]

Parameters

s	Serializer object

Here is the call graph for this function:

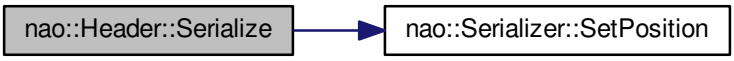


7.19.2.2 void nao::Header::Serialize (Serializer & *s*) [inline]

Parameters

s	Serializer object

Here is the call graph for this function:



7.19.3 Member Data Documentation

7.19.3.1 const int nao::Header::HeaderSize_ = 16 [static]

Header length in bytes. Has to be set correctly!

7.19.3.2 uint8_t nao::Header::isError_

Error flag: Packet contains an error message as payload

7.19.3.3 uint8_t nao::Header::numberOfChannels_

Number of audio channels recorded

7.19.3.4 uint32_t nao::Header::payloadLength_

Length of payload only

7.19.3.5 uint8_t nao::Header::protocolVersion_

Version of the tranfer protocol

7.19.3.6 uint8_t nao::Header::reserved_

unused field

7.19.3.7 uint32_t nao::Header::samplesPerChannel_

Samples per channel within one packet

7.19.3.8 uint32_t nao::Header::samplingRate_

Sampling rate

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

- Packet.hpp

7.20 nao::Packet Class Reference

Packet type which is used for data transfer between proxy and client

`#include <Packet.hpp>`

Public Types

- typedef uint16_t **value_type**

Public Member Functions

- void SetPayload (void *src, size_t len)

  Allocates memory and copies the memory block starting at src with length len to the payload section of the packet.
- void SetErrorMessage (const std::string &msg)

  Sets an error message as payload and sets the header flag isError_.
- void SetPacketInformation (const uint8_t numberOfChannels, const uint32_t samplesPerChannel, const uint32_t sampleRate)

  Method to set the audio-related header fields.
- void EncodeHeader ()
- void DecodeHeader ()

  Deserialializes the header variables from the header section of the packet.
- bool IsError () const

  Returns true if the packet contains an error message, false otherwise.
- std::string GetErrorMessage () const

  Returns the payload interpreted as a string.
- int GetProtocolVersion () const

  Returns the protocol version.
- size_t GetNumberOfChannels () const

  Returns the number of recorded channels.
- size_t GetSamplesPerChannel () const

  Returns the number of samples per channel in the current packet.
- size_t GetSamplingRate () const

  Returns the sampling rate.
- size_t GetPacketLength () const

  Returns the length of the whole packet (header + payload).
- size_t GetHeaderLength () const

  Returns the header length.
- size_t GetPayloadLength () const

  Returns the payload length.
- const uint8_t * **GetHeaderPtr** () const
- uint8_t * **GetHeaderPtr** ()
- const uint8_t * **GetPayloadPtr** () const
- uint8_t * **GetPayloadPtr** ()
- const uint8_t * **GetDataPtr** () const
- uint8_t * **GetDataPtr** ()
- void Resize (size_t newSize)

  Allocates memory for the payload. MUST be used before writing to the pointer returned by GetPayloadPtr()

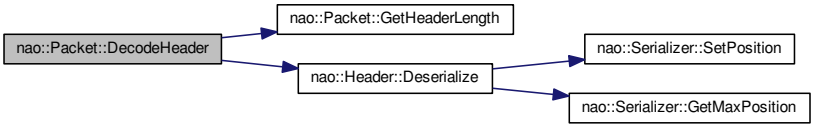
7.20.1 Detailed Description

Class Packet represents a block of memory containing audio data or an error string plus some header information. Data and serialized header information are stored in a single vector of bytes, the size of which can be adjusted dynamically. For convenience, the class has a Header field for the deserialized header information, too.

7.20.2 Member Function Documentation

7.20.2.1 void nao::Packet::DecodeHeader ( ) [inline]

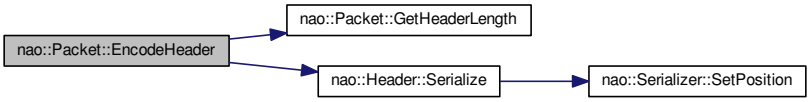
Here is the call graph for this function:



7.20.2.2 void nao::Packet::EncodeHeader ( ) [inline]

Serialializes the header variables to the header section of the packet.

Here is the call graph for this function:



7.20.2.3 std::string nao::Packet::GetErrorMessage ( ) const [inline]

Returns

error message

Here is the call graph for this function:

7.20.2.4 size_t nao::Packet::GetHeaderLength ( ) const [inline]

Returns

header length

7.20.2.5 size_t nao::Packet::GetNumberOfChannels ( ) const [inline]

Returns

number of channels

7.20.2.6 size_t nao::Packet::GetPacketLength ( ) const [inline]

Returns

packet length

7.20.2.7 size_t nao::Packet::GetPayloadLength ( ) const [inline]

Returns

payload length

7.20.2.8 int nao::Packet::GetProtocolVersion ( ) const [inline]

Returns

protocol version

7.20.2.9 size_t nao::Packet::GetSamplesPerChannel ( ) const [inline]

Returns

number of samples per channel

7.20.2.10 size_t nao::Packet::GetSamplingRate ( ) const [inline]

Returns

sampling rate

7.20.2.11 bool nao::Packet::IsError ( ) const [inline]

Returns

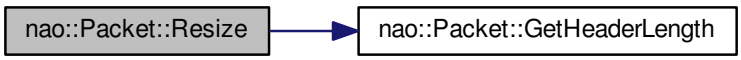
true if the packet contains an error message

7.20.2.12 void nao::Packet::Resize (size_t *newSize*) [inline]

Parameters

newSize	new payload length

Here is the call graph for this function:

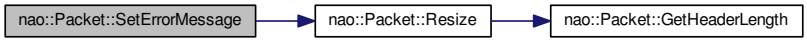


7.20.2.13 void nao::Packet::SetErrorMessage (const std::string & *msg*) [inline]

Parameters

msg	error message

Here is the call graph for this function:



7.20.2.14 void nao::Packet::SetPacketInformation (const uint8_t *numberOfChannels*, const uint32_t *samplesPerChannel*, const uint32_t *sampleRate*) [inline]

Parameters

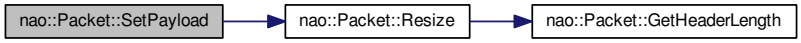
numberOf↩ Channels	number of recorded channels
samplesPer↩ Channel	samples per channel per block
sampleRate	sampling rate

7.20.2.15 void nao::Packet::SetPayload (void * *src*, size_t *len*) [inline]

Parameters

src	address of memory block
len	length of memory block

Here is the call graph for this function:



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

- Packet.hpp

7.21 nao::Serializer Class Reference

Simple serializer used to insert header information into packets.

`#include <Serialization.hpp>`

Public Member Functions

- Serializer (uint8_t *mem, size_t len)

  Constructor.
- **Serializer** (const Serializer &rhs)=delete
- Serializer & **operator=** (const Serializer &)=delete
- void SetPosition (size_t newPosition)

  Sets the read/write position (in bytes) inside the block of memory.
- size_t GetMaxPosition () const

  Returns the maximum value for the read/write position inside the memory block in bytes.
- template<typename T >
  Serializer & operator<< (const T &val)

  Serializing stream operator.
- template<typename T >
  Serializer & operator>> (T &val)

  Deserializing stream operator.

7.21.1 Detailed Description

This very simple serializer is constructed from a pointer to a block of memory and its length. Depending on the usage of the serializer, the block of memory is either interpreted as source or destination regarding serialization direction.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 nao::Serializer::Serializer (uint8_t * *mem*, size_t *len*) [inline]

Parameters

mem	block of memory on which the serializer shall operate
len	length of the above block of memory

7.21.3 Member Function Documentation

7.21.3.1 size_t nao::Serializer::GetMaxPosition ( ) const [inline]

Returns

largest value allowed for read/write position

7.21.3.2 template<typename T > Serializer& nao::Serializer::operator<< (const T & *val*) [inline]

Parameters

val	variable to be serialized

Returns

Serializer& reference to Serializer object

7.21.3.3 template<typename T > Serializer& nao::Serializer::operator>> (T & *val*) [inline]

Parameters

val	destination variable for deserialization

Returns

reference to Serializer object

7.21.3.4 void nao::Serializer::SetPosition (size_t *newPosition*) [inline]

Parameters

newPosition	between 0 and GetMaxPosition()

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

- Serialization.hpp

7.22 nao::TcpClient Class Reference

A small TCP Client for the connection to the AudioDatabroker.

`#include <TcpClient.hpp>`

Public Types

- using **PacketPtr** = exception_enabled_unique_ptr< Packet >
- using **PacketQueue** = ThreadsafeQueue< PacketPtr >
- using **OverflowHandler** = std::function< void(TcpClient &)>

Public Member Functions

- TcpClient (const std::string &ip, int port)

  Constructor, calls Connect()
- void Connect (const std::string &ip, int port)

  Connects the client with the broker.
- void StartAcceptingData ()

  Prompts the client to fill the input queue with received packets.
- void StopAcceptingData ()

  Prompts the client to stop pushing new packets into the queue.
- void ClearQueue ()

  Removes all buffered packets from the queue.
- PacketQueue::size_type PacketsInQueue () const

  Returns the number of packets in the queue.
- ClientState GetClientState () const

  Returns the current state of the TCP-client (disconnected, idle, running...).
- PacketPtr WaitForPacket (unsigned int milliseconds)

  Blocks the current thread while waiting for the TCP-client to receive a new packet. Returns as soon as a packet is in the queue or after the specified timeout.
- void SetOverflowHandler (const OverflowHandler &ovh)

  Sets the overflow handler.
- OverflowHandler **GetOverflowHandler** () const
- void Disconnect ()

  Closes the connection.

7.22.1 Detailed Description

The class TcpClient implements a simple client to receive packets from the AudioDataBroker. The received packets are stored in a queue and can be accessed via the WaitForPacket() method. To check for queue overflows, a function object with the signature "void (TcpClient&)" can be set, which is invoked before each insertion operation. This function object has to check the the number of packets in the queue and react in a proper way (clearing the queue, calling StopAcceptingData(), ...). Note, however, that if not stopped, the client will continue to insert new packets into the queue. For the case that no custom overflow handler has been set, a default handler will be used.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 nao::TcpClient::TcpClient (const std::string & *ip*, int *port*)

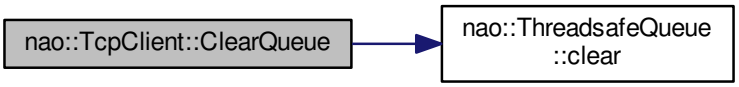
Parameters

ip	IP of the soundextractor broker
port	Port which is assigned to the broker

7.22.3 Member Function Documentation

7.22.3.1 void nao::TcpClient::ClearQueue ( ) [inline]

Here is the call graph for this function:



7.22.3.2 void nao::TcpClient::Connect (const std::string & *ip*, int *port*)

Parameters

ip	IP of the soundextractor broker
port	Port which is assigned to the broker

7.22.3.3 void nao::TcpClient::Disconnect ( )

7.22.3.4 ClientState nao::TcpClient::GetClientState ( ) const [inline]

Returns

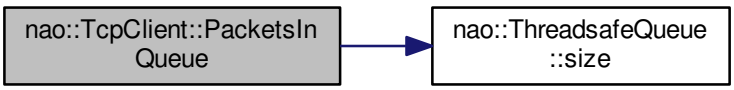
current ClientState

7.22.3.5 TcpClient::PacketQueue::size_type nao::TcpClient::PacketsInQueue ( ) const [inline]

Returns

number of packets in the queue

Here is the call graph for this function:



7.22.3.6 void nao::TcpClient::SetOverflowHandler (const OverflowHandler & *ovh*) [inline]

Parameters

ovh	reference to a callable object with the signature "void (TcpClient&)"

7.22.3.7 void nao::TcpClient::StartAcceptingData ( )

7.22.3.8 void nao::TcpClient::StopAcceptingData ( )

7.22.3.9 TcpClient::PacketPtr nao::TcpClient::WaitForPacket (unsigned int *milliseconds*) [inline]

Parameters

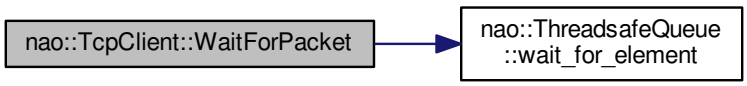
milliseconds	timeout for wait operation

Returns

PacketPtr pointer which

- points to a packet
- carries an exception origninating from the TCP-client or TCP-server
- is a nullpointer if the timeout has expired and there is still no packet in the queue

Here is the call graph for this function:



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

- TcpClient.hpp
- TcpClient.cpp

7.23 nao::detail::test_type< X > Struct Template Reference

Public Attributes

- char **a** [2]

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

- has_member_in.hpp

7.24 nao::detail::test_type< void > Struct Template Reference

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

- has_member_in.hpp

7.25 nao::ThreadsafeQueue< T > Class Template Reference

A basic thread-safe std::queue wrapper object.

`#include <ThreadsafeQueue.hpp>`

Public Types

- using **value_type** = T
- using **size_type** = typename std::queue< value_type >::size_type

Public Member Functions

- ThreadsafeQueue ()=default

  Default constructor.
- ~ThreadsafeQueue ()=default

  Default destructor.
- ThreadsafeQueue & operator= (const ThreadsafeQueue &)=delete

  Assignment operator deleted.
- bool empty () const

  Same as std::queue<T>::operator =(), but thread-safe.
- size_type size () const

  Same as std::queue<T>::size(), but thread-safe.
- template<typename U >
  void push (U &&val)

  Same as std::queue<T>::push(), but thread-safe. Notifies waiting threads that a new element has been added.
- bool try_pop (value_type &val)

  Tries to pop one element from the queue.
- bool wait_for_element (value_type &val, unsigned int milliseconds)

  Waits for a new element for at the most "milliseconds" ms. Returns immediately, if an element is available.
- void clear ()

  Waits for a new element for at the most "milliseconds" ms. Returns immediately, if an element is available. Removes all elements by swapping the internal queue with a default constructed (empty) one.
- template<class... Args>
  void emplace (Args &&...args)

  Inserts elements constructed using args.

7.25.1 Detailed Description

template<typename T>class nao::ThreadsafeQueue< T >

ThreadsafeQueue provides mutex based threadsafety for std::queue. The interface differs somewhat from the standard library in order to allow "atomic" if-empty-then-pop operations. To understand this, note that the following code is NOT thread-safe, even if both, empty() and pop() were thread-safe on their own:

```
ThreadsafeQueue<int> q;
...
//somewhere else
if(!q.empty())   //may return 1
                 //another thread kicks in here and calls pop() on q
                 //the q is now empty!
    q.pop();     //back in this thread, we will run straight into undefined
                 //behaviour
```

To work around this issue, the functions try_pop and wait_for_element are introduced. They take a reference to value_type to store the result of pop() and indicate success or failure of the operation by a boolean return value.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 template<typename T> nao::ThreadsafeQueue< T >::ThreadsafeQueue ( ) [default]

7.25.2.2 template<typename T> nao::ThreadsafeQueue< T >::~ThreadsafeQueue ( ) [default]

7.25.3 Member Function Documentation

7.25.3.1 template<typename T> template<class... Args> void nao::ThreadsafeQueue< T >::emplace (Args &&... *args*) [inline]

Parameters

args	Args&&... argument list for constructing the new elements

7.25.3.2 template<typename T> bool nao::ThreadsafeQueue< T >::empty ( ) const [inline]

Returns

true if the queue is empty, false otherwise

7.25.3.3 template<typename T> ThreadsafeQueue& nao::ThreadsafeQueue< T >::operator= (const ThreadsafeQueue< T > &) [delete]

7.25.3.4 template<typename T> template<typename U > void nao::ThreadsafeQueue< T >::push (U && *val*) [inline]

Parameters

val	value to be added to the queue

7.25.3.5 template<typename T> size_type nao::ThreadsafeQueue< T >::size ( ) const [inline]

Returns

number of elements in the queue

7.25.3.6 template<typename T> bool nao::ThreadsafeQueue< T >::try_pop (value_type & *val*) [inline]

Parameters

val	reference to the destination

Returns

true if the operation was successful, false otherwise

7.25.3.7 template<typename T> bool nao::ThreadsafeQueue< T >::wait_for_element (value_type & *val,* unsigned int *milliseconds*) [inline]

Parameters

val	refercence to an object to which the the new element shall be moved
milliseconds	timeout

Returns

true if a new element is available, false on timeout

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

- ThreadsafeQueue.hpp

7.26 ThreadsafeQueue< T > Singleton Reference

`#include <ThreadsafeQueue_fwd.hpp>`

7.26.1 Detailed Description

template<typename T>singleton ThreadsafeQueue< T >

Forward declaration of class ThreadsafeQueue

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

- ThreadsafeQueue_fwd.hpp

Chapter 8

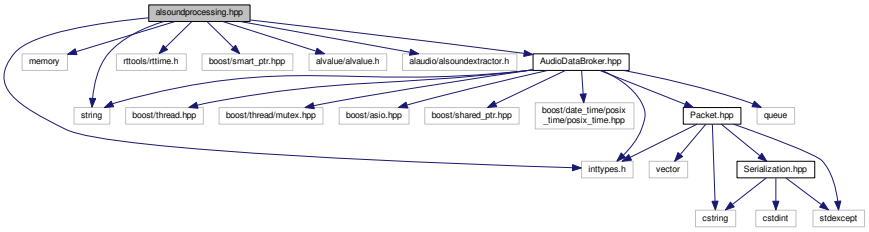
File Documentation

8.1 alsoundprocessing.hpp File Reference

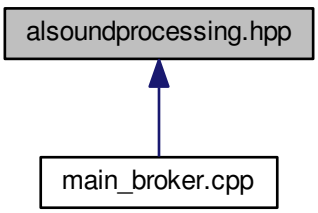
Contains the declaration of class nao::AlSoundProcessing and struct nao::AlSoundProcessingParams.

```
#include <string>
#include <memory>
#include <inttypes.h>
#include <rttools/rttime.h>
#include <boost/smart_ptr.hpp>
#include <alvalue/alvalue.h>
#include <alaudio/alsoundextractor.h>
#include "AudioDataBroker.hpp"
```

Include dependency graph for alsoundprocessing.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct nao::ALSoundProcessingParams

  Helper-struct to bundle parameters.

- class nao::ALSoundProcessing

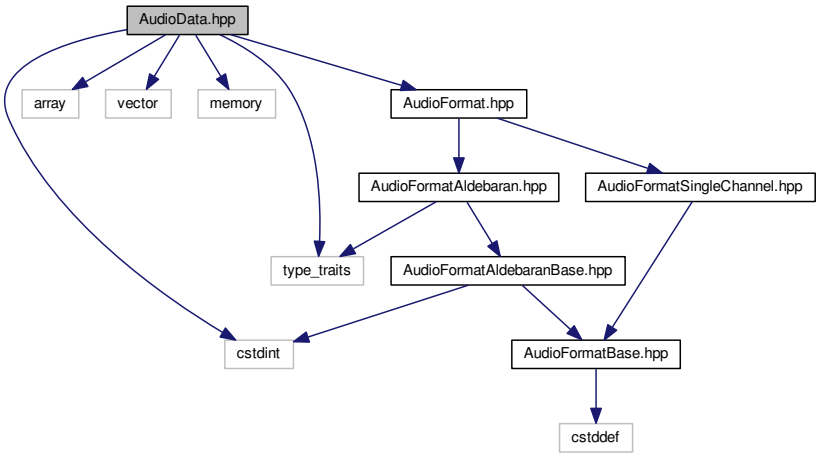
  Module class to extend NAO's sound processing capabilities.

8.2 AudioData.hpp File Reference

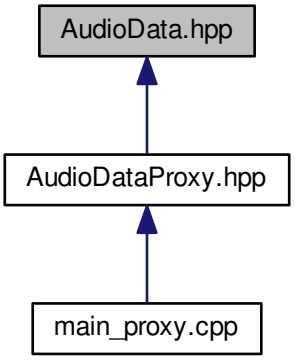
Contains the definition of class nao::AudioData.

```
#include <cstdint>
#include <array>
#include <vector>
#include <memory>
#include <type_traits>
#include "AudioFormat.hpp"
```

Include dependency graph for AudioData.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::AudioData< FMT >

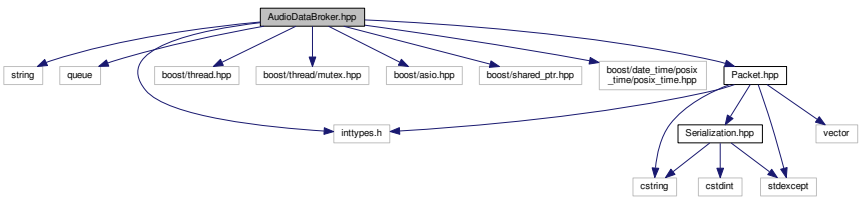
  A container for audiodata.

8.3 AudioDataBroker.hpp File Reference

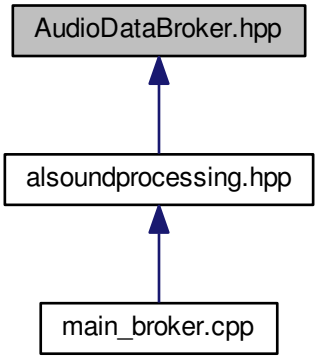
Contains the declaration of class AudioDataBroker.

```
#include <string>
#include <queue>
#include <inttypes.h>
#include <boost/thread.hpp>
#include <boost/thread/mutex.hpp>
#include <boost/asio.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include "Packet.hpp"
```

Include dependency graph for AudioDataBroker.hpp:



This graph shows which files directly or indirectly include this file:



Classes

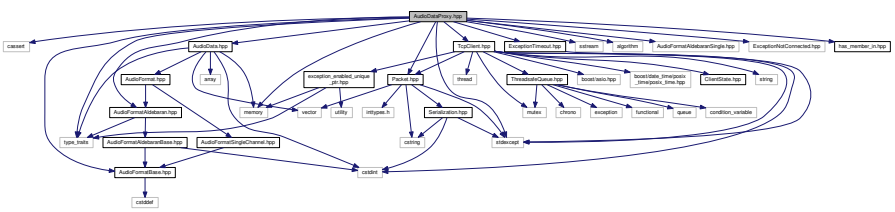
- class nao::AudioDataBroker

8.4 AudioDataProxy.hpp File Reference

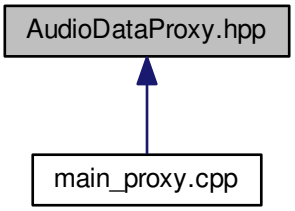
Contains the definition of class nao::ExceptionTcpClient.

```
#include <cassert>
#include <type_traits>
#include <sstream>
#include <memory>
#include <algorithm>
#include "TcpClient.hpp"
#include "Packet.hpp"
#include "AudioData.hpp"
#include "AudioFormatAldebaran.hpp"
#include "AudioFormatAldebaranSingle.hpp"
#include "ExceptionTimeout.hpp"
#include "ExceptionNotConnected.hpp"
#include "has_member_in.hpp"
```

Include dependency graph for AudioDataProxy.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::AudioDataProxy

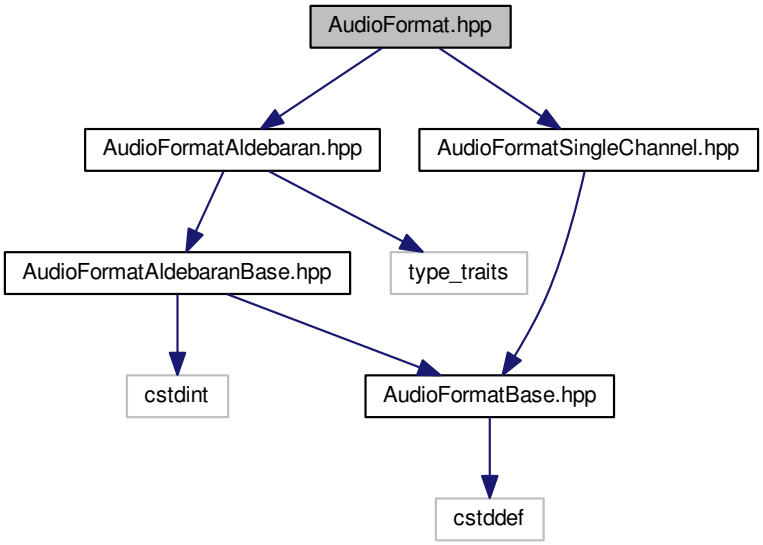
  Proxy class for receiving audio data.

8.5 AudioFormat.hpp File Reference

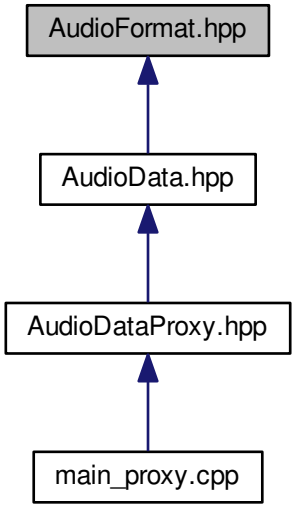
Include file for grouping all defined audio formats.

```
#include "AudioFormatAldebaran.hpp"
#include "AudioFormatSingleChannel.hpp"
```

Include dependency graph for AudioFormat.hpp:



This graph shows which files directly or indirectly include this file:

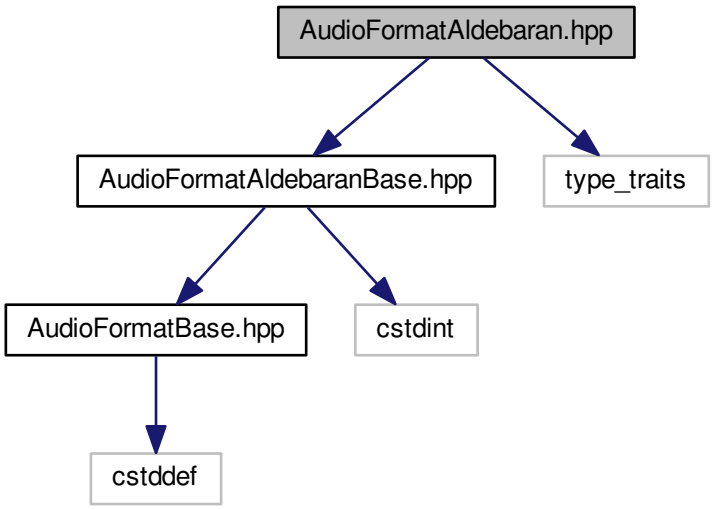


8.6 AudioFormatAldebaran.hpp File Reference

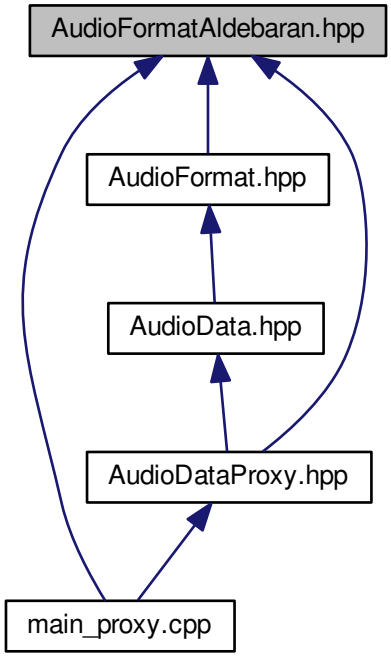
Contains the definitions of classes AudioFormatAldebaranAll, AudioFormatAldebaranFront, AudioFormat↩ AldebaranLeft AudioFormatAldebaranRight and AudioFormatAldebaranRear.

```
#include "AudioFormatAldebaranBase.hpp"
#include <type_traits>
```

Include dependency graph for AudioFormatAldebaran.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::AudioFormatAldebaranAll

  An audio format for all existing NAO-Channels.

- class nao::AudioFormatAldebaranFront

  An audio format for the front microphone channel only.

- class nao::AudioFormatAldebaranLeft

  An audio format for the left microphone channel only.

- class nao::AudioFormatAldebaranRight

  An audio format for the right microphone channel only.

- class nao::AudioFormatAldebaranRear

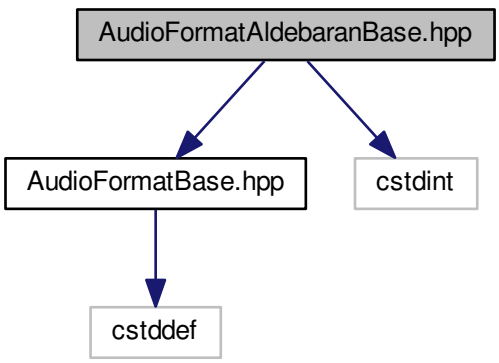
  An audio format for the rear microphone channel only.

8.7 AudioFormatAldebaranBase.hpp File Reference

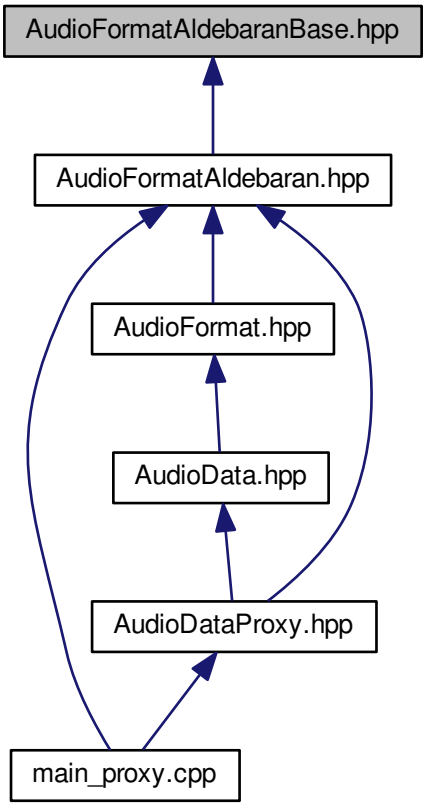
Contains the definition of class nao::AudioFormatAldebaranBase.

```
#include "AudioFormatBase.hpp"
#include <cstdint>
```

Include dependency graph for AudioFormatAldebaranBase.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::AudioFormatAldebaran

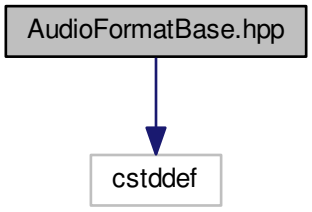
  Base class for audio formats which can be processed by the AudioDataProxy.

8.8 AudioFormatBase.hpp File Reference

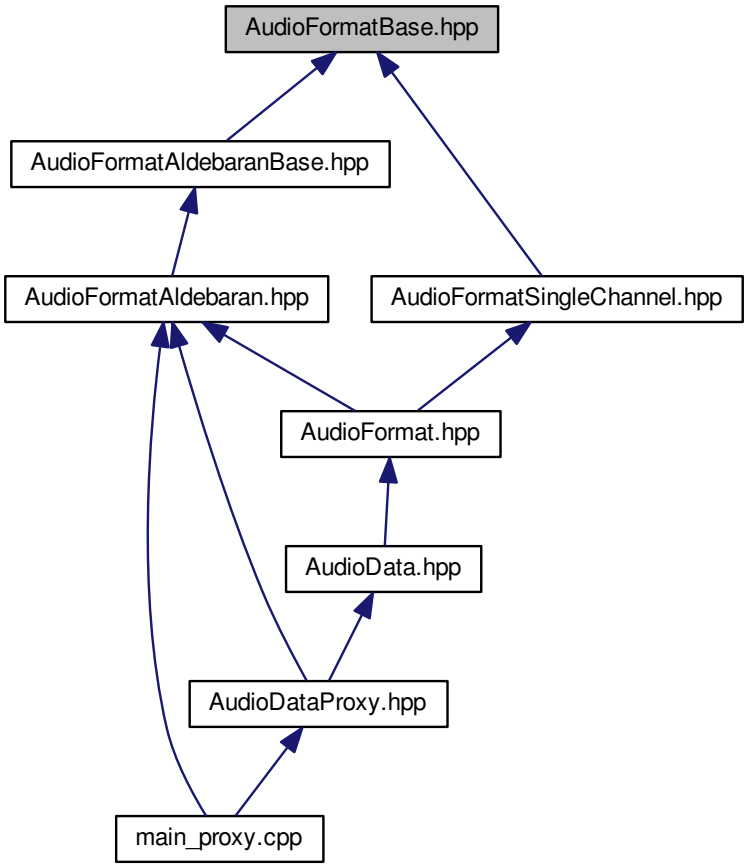
Contains the definition of class nao::AudioFormatBase.

```
#include <cstddef>
```

Include dependency graph for AudioFormatBase.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::AudioFormatBase

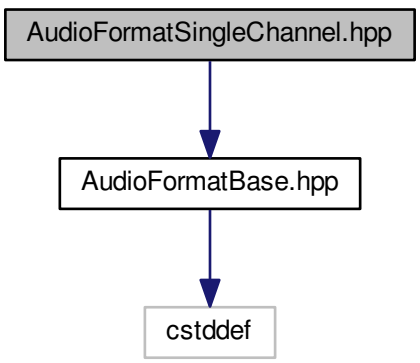
  Base class for all audio formats.

8.9 AudioFormatSingleChannel.hpp File Reference

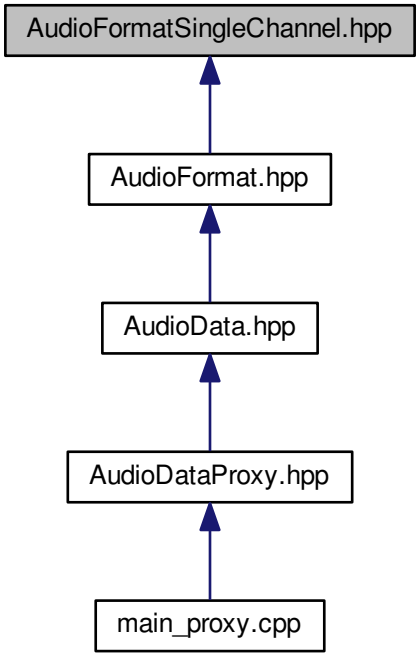
Contains the definition of class nao::AudioFormatSingleChannel.

```
#include "AudioFormatBase.hpp"
```

Include dependency graph for AudioFormatSingleChannel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

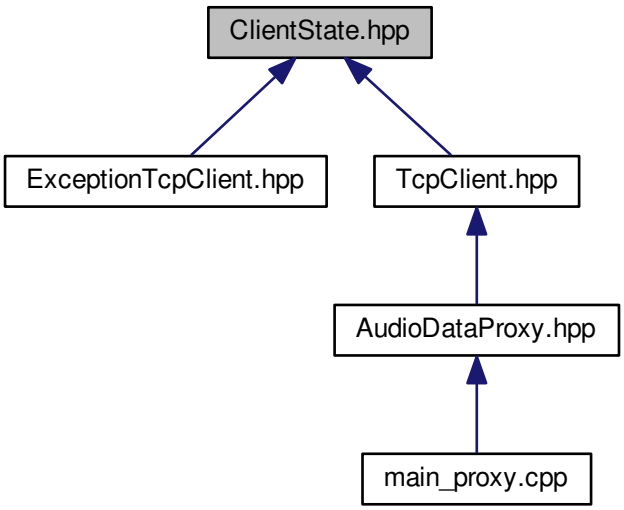
- class nao::AudioFormatSingleChannel

  An exemplary audio format for a single channel.

8.10 ClientState.hpp File Reference

Contains the definition of enum ClientState.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum nao::ClientState {
  disconnected, **disconnecting**, **connecting**, **idle**,
  running }

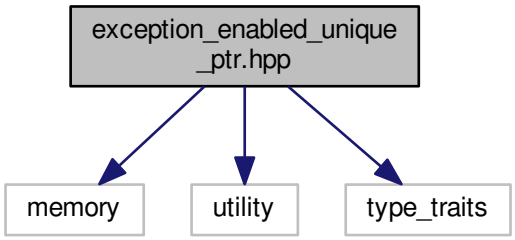
  Enum values representing the state of the TcpClient.

8.11 exception_enabled_unique_ptr.hpp File Reference

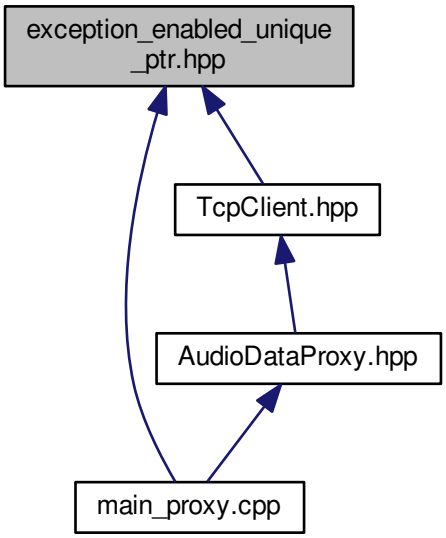
Contains the definition of class nao::exception_enabled_unique_ptr.

```
#include <memory>
#include <utility>
#include <type_traits>
```

Include dependency graph for exception_enabled_unique_ptr.hpp:



This graph shows which files directly or indirectly include this file:



Classes

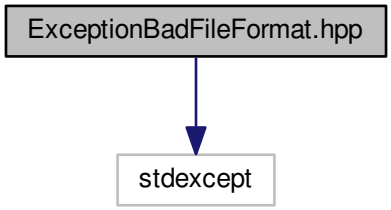
- class nao::exception_enabled_unique_ptr< T, D >

  std::unique_ptr with the ability to store exceptions.

8.12 ExceptionBadFileFormat.hpp File Reference

Contains the definition of class nao::ExceptionBadFileFormat.

`#include <stdexcept>`
Include dependency graph for ExceptionBadFileFormat.hpp:



Classes

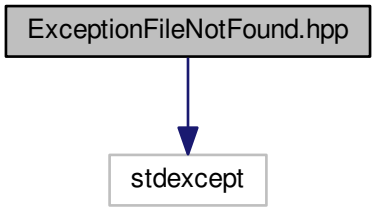
- class nao::ExceptionBadFileFormat

  Exception class for reporting incorrect file formats.

8.13 ExceptionFileNotFound.hpp File Reference

Contains the definition of class nao::ExceptionFileNotFound.

`#include <stdexcept>`
Include dependency graph for ExceptionFileNotFound.hpp:



Classes

- class nao::ExceptionFileNotFound

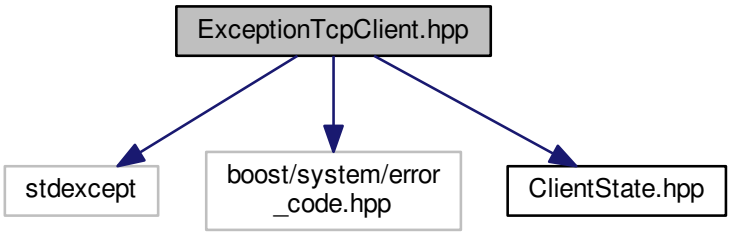
  Exception class for reporting missing files.

8.14 ExceptionTcpClient.hpp File Reference

Contains the definition of class nao::ExceptionTcpClient.

```
#include <stdexcept>
#include <boost/system/error_code.hpp>
#include "ClientState.hpp"
```

Include dependency graph for ExceptionTcpClient.hpp:



Classes

- class nao::ExceptionTcpClient

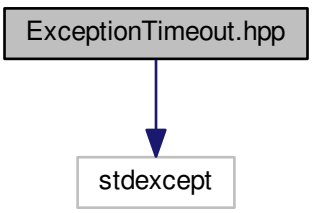
  Exception class for reporting errors within the TcpClient code.

8.15 ExceptionTimeout.hpp File Reference

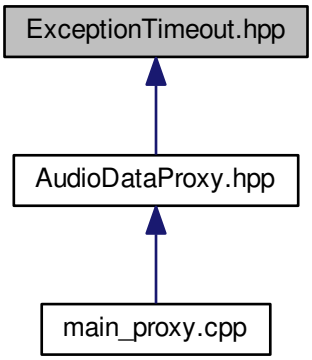
Contains the definition of class nao::ExceptionTimeout.

```
#include <stdexcept>
```

Include dependency graph for ExceptionTimeout.hpp:



This graph shows which files directly or indirectly include this file:



Classes

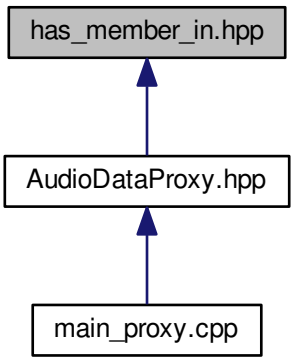
- class nao::ExceptionTimeout

  Exception class for reporting timeouts.

8.16 has_member_in.hpp File Reference

Contains the definition of makro GEN_HAS_MEMBER_IN(MEMBER, NAMESPACE)

This graph shows which files directly or indirectly include this file:



Classes

- struct nao::detail::test_type< X >

- struct nao::detail::test_type< void >

Macros

- #define GEN_HAS_MEMBER_IN(MEMBER, NAMESPACE)

8.16.1 Macro Definition Documentation

8.16.1.1 #define GEN_HAS_MEMBER_IN(*MEMBER, NAMESPACE*)

Value:

```
namespace nao
      \
{                                                                                                   \
    namespace detail                                                                                \
    {                                                                                               \
        template<typename T>                                                                        \
        class has_member_##MEMBER##_impl                                                            \
        {                                                                                           \
            public:                                                                                 \
                template<typename U>                                                                \
                static detail::test_type<decltype(U::NAMESPACE::MEMBER)> test(U*);                  \
                                                                                                    \
                template<typename>                                                                  \
                static detail::test_type<void> test(...);                                           \
                                                                                                    \
                enum { HAS_MEMBER = sizeof(test<T>(nullptr)) != sizeof(detail::test_type<void>) };  \
        };                                                                                          \
    }                                                                                               \
    template<typename T>                                                                            \
    class has_member_##MEMBER##_in_##NAMESPACE                                                      \
        : public std::integral_constant<bool, detail::has_member_##MEMBER##_impl<T>::HAS_MEMBER> {};  \
}
```

Makro to generate tests for the existence of a public variable within a namespace at compile time.

Please see the example for the usage.

This makro expands to class named has_member_MEMBER_in_NAMESPACE, with MEMBER and NAMESPACE being replaced by the makro arguments. The result of the test can be accessed via the () operator or the typedef "value" as a boolean value or via the typedef "type" as a type (either std::true_type or std::false_type).

How the compile time variable detection works: Since the test shall be evaluated during compile time, template metaprogramming techniques are used. The heart of the code is the template function "test" inside the class has↩ _member_MEMBER_impl, which is two times overloaded. The first overload takes a pointer to the class under test and returns a variable of the type detail::test_type<decltype(U::NAMESPACE::MEMBER)>. The crucial point of this expression is the decltype(U::NAMESPACE::MEMBER) part, which returns the type of MEMEBER in the namespace NAMESPACE inside the class under test U. If this variable doesn't exist (or is declared private), the whole function definition is invalid, so that the compiler is forced, due to SFINAE, to instantiate the next best overload, which is the void-returning test(...). Still, the result has to be stored in one single variable. Therefore, a template helper class, test_typeis used. The generic test_type<X> class contains an char array of size two, whereas the specialisation test_type<void> has no member at all and therefore is one byte long. With this, it is now possible to compare the size of the return values of both test overloads and assign a boolean value to a compile time constant: HAS_MEMBER = sizeof(test<T>(nullptr)) != sizeof(detail::test_type<void>).

The class has_member_MEMBER_in_NAMESPACE inherits some convenience functions from std::integral_↩ constant, such as the "value" and "type" typedefs.

Examples:

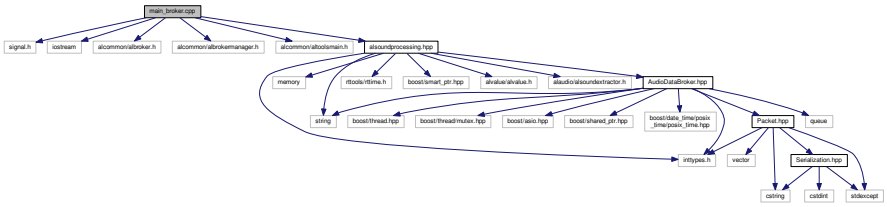
has_member_in_ex.cpp.

8.17 main_broker.cpp File Reference

Main file for target Build target Broker.

```
#include <signal.h>
#include <iostream>
#include <alcommon/albroker.h>
#include <alcommon/albrokermanager.h>
#include <alcommon/altoolsmain.h>
#include "alsoundprocessing.hpp"
```

Include dependency graph for main_broker.cpp:



Functions

- ALCALL int **_createModule** (boost::shared_ptr< AL::ALBroker > pBroker)
- ALCALL int **_closeModule** ()

8.17.1 Detailed Description

This is the main file for the creation of a NAOqi extension module as suggested by Aldebaran Robotics. To allow for switching between the generation of local and remote modules, the module creation is shifted into the function _createModule(), which acts in case of the local module as entry point of the shared library. For remote modules, the same code is executed from within ALTools::mainFunction.

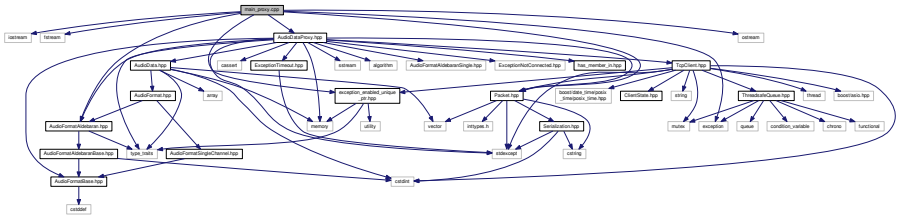
The parameters for the AlSoundProcessing module are set using a helper struct, since AL::ALModule::create↩ Module accepts only a single argument. The parameters are the desired timeout for the server operations and the port of the server, respectively.

8.18 main_proxy.cpp File Reference

This is the main file for the Build target Proxy module.

```
#include <iostream>
#include <fstream>
#include <exception>
#include <ostream>
#include "AudioDataProxy.hpp"
#include "AudioFormatAldebaran.hpp"
#include "Packet.hpp"
#include "exception_enabled_unique_ptr.hpp"
```

Include dependency graph for main_proxy.cpp:



Functions

- int **main** ()

8.18.1 Detailed Description

This main file features a minimal example of the usage of the library:

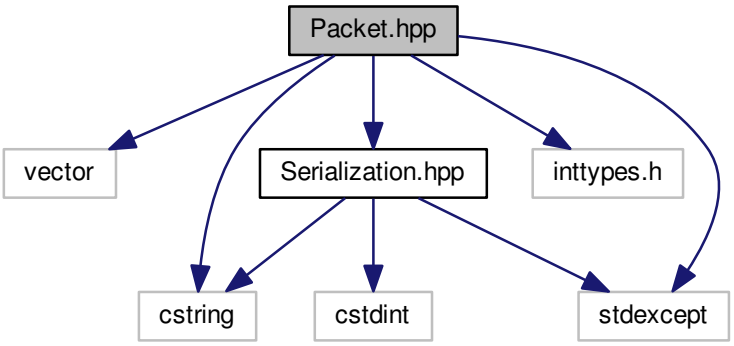
```
#include <iostream>
#include <fstream>

#include <exception>
#include <ostream>

#include "AudioDataProxy.hpp"
#include "AudioFormatAldebaran.hpp"

#include "Packet.hpp"

#include "exception_enabled_unique_ptr.hpp"

int main()
{
    try
    {
        // connect to the AudioDataBroker running on this computer and
        // listens on port 5392
        nao::AudioDataProxy proxy("localhost", 5392);

        // accept packets
        proxy.Start();

        // wait for 100 samples and bring them to the format
        // AudioFormatAldebaran
        // (returns an AudioData<nao::AudioFormatAldebaranAll> container)
        auto p = proxy.WaitForData<nao::AudioFormatAldebaranAll>(100);

        // do accept no more packets
        proxy.Pause();

        // do something with p...

        // close the connection gracefully
        proxy.Disconnect();
    }

    catch(const std::exception &e)
    {
        std::cout << e.what() << std::endl;
    }

    return 0;
}
```

8.19 Packet.hpp File Reference

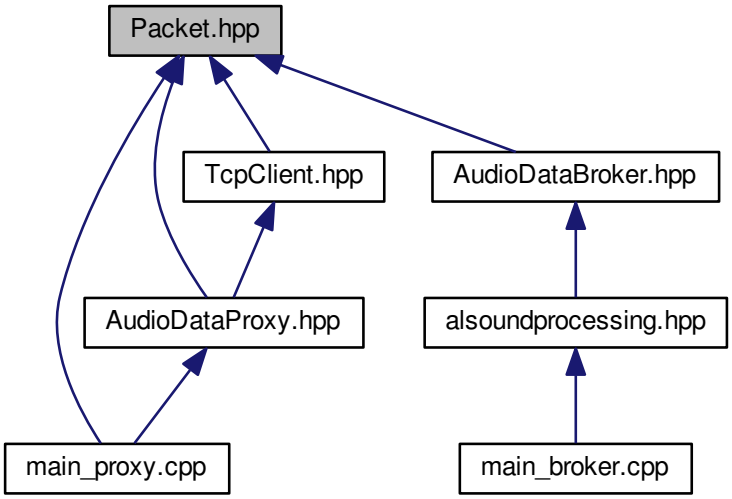
Contains the definition of class nao::Packet and struct nao::Header.

```
#include <vector>
#include <cstring>
#include <stdexcept>
#include <inttypes.h>
#include "Serialization.hpp"
```

Include dependency graph for Packet.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct nao::Header

  Header struct for class Packet.
- class nao::Packet

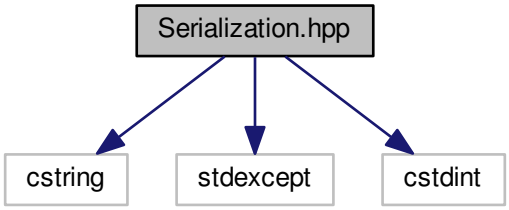
Packet type which is used for data transfer between proxy and client

8.20 Serialization.hpp File Reference

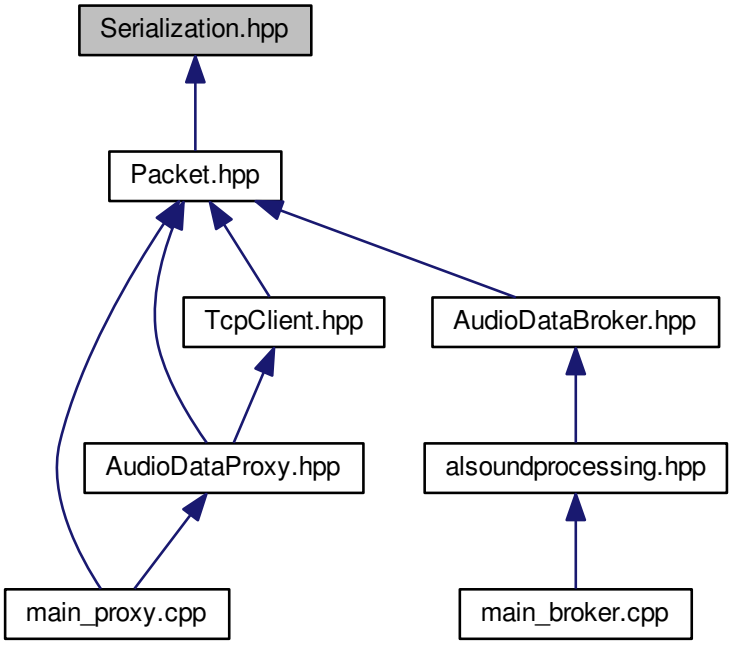
Contains the definition of class nao::Serializer.

```
#include <cstring>
#include <stdexcept>
#include <cstdint>
```

Include dependency graph for Serialization.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::Serializer

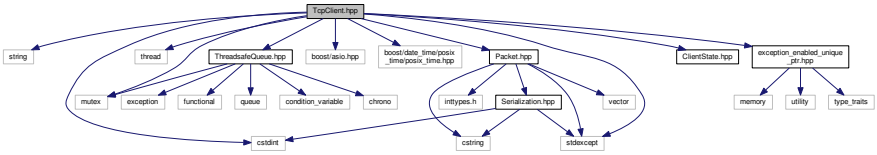
  Simple serializer used to insert header information into packets.

8.21 TcpClient.hpp File Reference

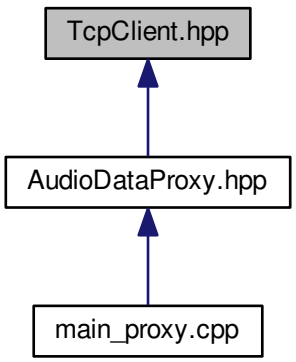
Contains the declaration of class TcpClient.

```
#include <string>
#include <cstdint>
#include <thread>
#include <mutex>
#include <stdexcept>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include "ThreadsafeQueue.hpp"
#include "Packet.hpp"
#include "ClientState.hpp"
#include "exception_enabled_unique_ptr.hpp"
```

Include dependency graph for TcpClient.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::TcpClient

  A small TCP Client for the connection to the AudioDatabroker.

Variables

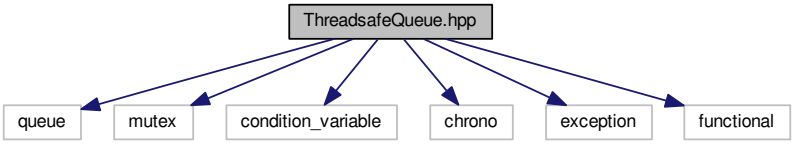
- constexpr auto **nao::DefaultOverflowHandler**

8.22 ThreadsafeQueue.hpp File Reference

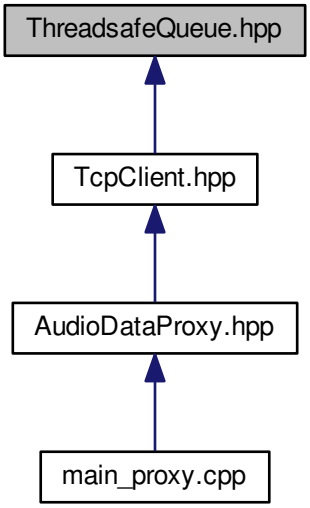
Contains the definition of class nao::ThreadsafeQueue.

```
#include <queue>
#include <mutex>
#include <condition_variable>
#include <chrono>
#include <exception>
#include <functional>
```

Include dependency graph for ThreadsafeQueue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class nao::ThreadsafeQueue< T >

A basic thread-safe std::queue wrapper object.

8.23 ThreadsafeQueue_fwd.hpp File Reference

Contains a forward declaration of class ThreadsafeQueue.

Classes

- singleton ThreadsafeQueue< T >

Chapter 9

Example Documentation

9.1 has_member_in_ex.cpp

```
#include <type_traits>
#include "../include/has_member_in.hpp"

#include "../include/AudioFormatAldebaran.hpp"

struct object_under_test
{
    enum class enums {A=0, B};

    struct structs
    {
        int A;
        int B;
    };
};

GEN_HAS_MEMBER_IN(A, enums)
GEN_HAS_MEMBER_IN(B, structs)

GEN_HAS_MEMBER_IN(C, enums)
GEN_HAS_MEMBER_IN(D, structs)

//Test cases where the variables DO exist
static_assert(nao::has_member_A_in_enums<object_under_test>(), "Ouch, you should never see this message!
      ");
static_assert(nao::has_member_B_in_structs<object_under_test>(), "Ouch, you should never see this
      message!");

//Test cases where the variables DO NOT exist
static_assert(!nao::has_member_C_in_enums<object_under_test>(), "Ouch, you should never see this message
      !");
static_assert(!nao::has_member_D_in_structs<object_under_test>(), "Ouch, you should never see this
      message!");
```

9.2 main_proxy.cpp

```
#include <iostream>
#include <fstream>

#include <exception>
#include <ostream>

#include "AudioDataProxy.hpp"
#include "AudioFormatAldebaran.hpp"

#include "Packet.hpp"

#include "exception_enabled_unique_ptr.hpp"

int main()
{
    try
    {
```

```
        // connect to the AudioDataBroker running on this computer and
        // listens on port 5392
        nao::AudioDataProxy proxy("localhost", 5392);

        // accept packets
        proxy.Start();

        // wait for 100 samples and bring them to the format
        // AudioFormatAldebaran
        // (returns an AudioData<nao::AudioFormatAldebaranAll> container)
        auto p = proxy.WaitForData<nao::AudioFormatAldebaranAll>(100
      );

        // do accept no more packets
        proxy.Pause();

        // do something with p...

        // close the connection gracefully
        proxy.Disconnect();
    }

    catch(const std::exception &e)
    {
        std::cout << e.what() << std::endl;
    }

    return 0;
}
```

Index