RESTORE 1.2.0

by zplane.development

January 8, 2021

Contents

1 RESTORE Documentation

1.1 Introduction

The RESTORE SDK can be used to clean recordings from background noise and clicks. It enhances the sound quality of digitized analog recordings (e.g. from vinyl or shellac discs or analog audio tapes). The SDK consists of two different parts: a denoising API, that is able to reduce stationary as well as non-stationary noise in audio and speech signals without need of a pre-calculated noise fingerprint, and a declicking API that can be used to reduce click sounds. The adaptive algorithms are able to work without user interaction and are capable of real-time processing.

The project contains the required libraries for the operating system it was licensed for with the appropriate header files. An example application illustrates the functionality of this SDK.

1.2 API Documentation

The SDK provides two separate interfaces: DenoisingIf.h contains the API for the noise reduction part and DeclickingIf.h for the click removal part.

1.2.1 Denoising

In order to reduce background noise, the denoising algorithm works with so-called *noise fingerprints*, i.e. prototypical spectra that characterize the noise in the input audio. The algorithm can work with two different types of fingerprints: an adaptive fingerprint that adaptively estimates the noise floor and that should be used in case of non-stationary background noise, and a static noise fingerprint that can be estimated from a noise-only passage. The method CDenoisingIf::CaptureFingerprint() can be used to provide noise input samples for the static fingerprint. CDenoisingIf::CalculateFingerprint() has to be called once to signal the end of the noise input and to compute the fingerprint. The adaptive fingerprint estimates a local noise fingerprint. Since the estimation method is less accurate in the lower frequency range, the amount of low frequency energy in the fingerprint can be manually adjusted.

The method CDenoisingIf::Process() runs the denoising algorithm on some audio input and returns the denoised output with a small and fixed latency. The latency can be obtained by calling CDenoisingIf::GetLatencyInSamples(). Various parameters can be set to control the quality of the noise reduction as explained in the API documentation below.

The following figure shows an example spectrogram of a noisy audio signal and its denoised version.

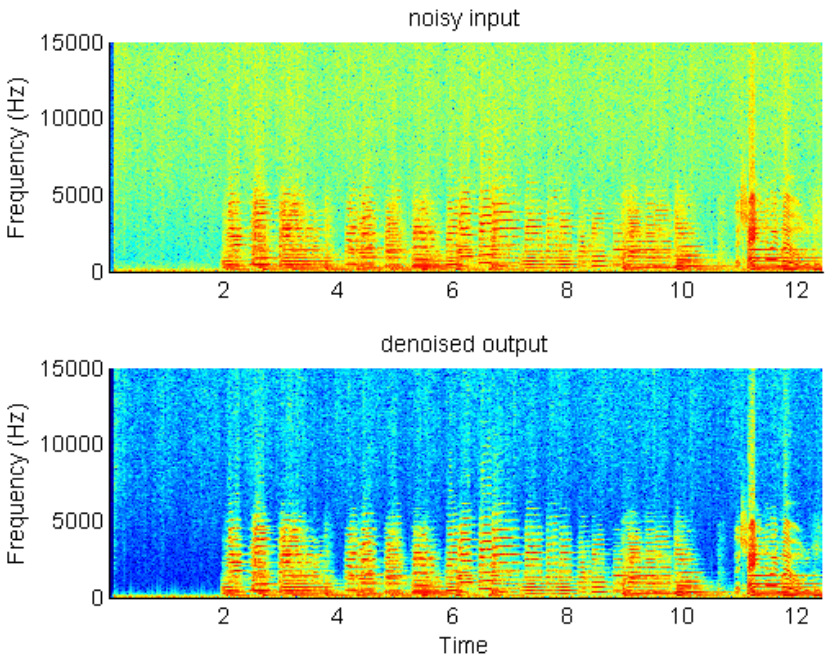

Figure 1: example denoising

1.2.2 Declicking

The method CDeclickingIf::Process() runs the algorithm on an arbitrary number of input samples from which clicks should be removed. It returns the same number of samples with a fixed latency. The latency can be obtained by calling CDeclickingIf::GetLatencyInSamples(). The algorithm parameters can be set by the method CDeclickingIf::SetParamValue(). A detailed explanation for each parameter can be found in the API documentation below.

The figures below show an example of a signal with a significant number of clicks as well as its processed version that has the clicks removed.

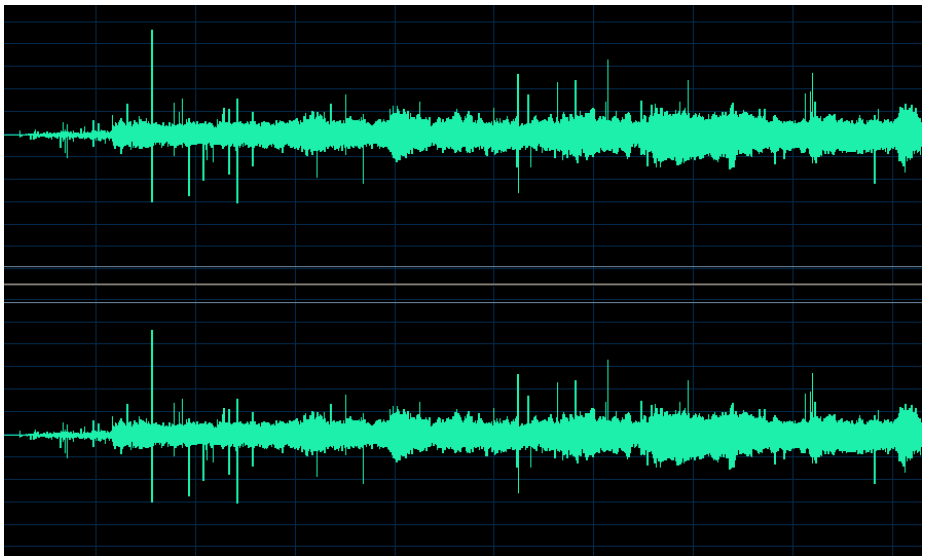

Figure 2: example: before Declicking

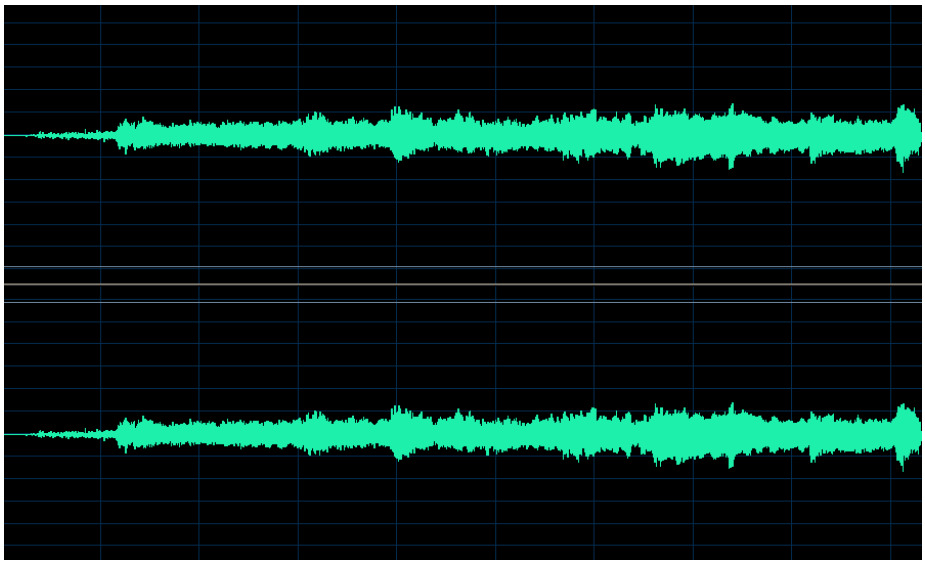

Figure 3: example: after Declicking

1.2.3 Naming Conventions

A **frames** denotes the number of audio samples per channel, i.e. 512 stereo frames correspond to 1024 float values (samples).

1.3 C++-API description

1.3.1 Memory Allocation

The RESTORE SDK does not allocate buffers handled by the calling application. The input buffers have to be allocated by the calling application. Audio buffers are allocated as arrays of size [numberOfChannels][numberOfSamplesPerChannel].

1.3.2 Denoising

Instance Handling Functions

- **DenoisingError_t CDenoisingIf::CreateInstance (CDenoisingIf ∗&pCInstance↩ Pointer, float fSampleRate, int iNumberOfChannels)**
  Creates a new instance of the denoiser. The handle to the new instance is returned in parameter pCInstancePointer. The sample rate and the number of channels of the audio data to be analyzed is given in parameter fSampleRate resp. iNumber↩ OfChannels. The return value indicates whether an error occurred or not.

- **DenoisingError_t CDenoisingIf::DestroyInstance (CDenoisingIf ∗& pCInstance↩ Pointer)**
  Destroys an instance of the denoiser. The handle to instance is in parameter p↩ CInstancePointer, which is set to 0 by this function. The return value indicates whether an error occurred or not.

- **DenoisingError_t CDenoisingIf::ResetInstance ()**
  Resets all internal variables and buffers to the default state. The return value indicates whether an error occurred or not.

Process Functions

- **DenoisingError_t CDenoisingIf::Process (float const ∗const ∗const ppfInput↩ Buffer, float ∗const ∗const ppfOutputBuffer, int iNumberOfFrames)**
  Does the actual denoising of the input buffer. The result is written into the output buffer. If the usage of a noise fingerprint is enabled and a fingerprint was captured, the noise reduction is done using the precalculated fingerprint. Otherwise an adaptive noise fingerprint is estimated for each frame. The return value indicates whether an error occurred or not. The processing can be controlled by a various number of parameters which are combined together in the enum CDenoisingIf::DenoisingParam_t and will be explained in detail later on.

- **DenoisingError_t CDenoisingIf::CaptureFingerprint (float ∗∗ppfInputBuffer, int iNumberOfFrames)**
  Captures the magnitude spectrum using the samples in ppfInputBuffer and adds them to an internal buffer. To reset the fingerprint CDenoisingIf::ResetFingerprint() has to be executed. The return value indicates whether an error occurred or not.

- **DenoisingError_t CDenoisingIf::CalculateFingerprint ()**
  Calculates the noise fingerprint using the captured magnitude spectra. To reset the fingerprint CDenoisingIf::ResetFingerprint() has to be executed. The return value indicates whether an error occurred or not.

- **DenoisingError_t CDenoisingIf::ResetFingerprint ()**
  Resets the current Fingerprint to default value. The return value indicates whether an error occurred or not.

Parameter Retrieving and Setting Functions

- **DenoisingError_t CDenoisingIf::GetParamValue (DenoisingParam_t iParam↩ Idx, float &fValue)**
  Writes parameter with index iParamIdx to fValue. The parameter indices are listed in enum CDenoisingIf::DenoisingParam_t{ }. The return value indicates whether an error occurred or not.

- **DenoisingError_t CDenoisingIf::SetParamValue (DenoisingParam_t iParam↩ Idx, float fValue)**
  Sets parameter with index iParamIdx with fValue. The parameter indices are listed in enum CDenoisingIf::DenoisingParam_t. The return value indicates whether an error occurred or not.

- **int CDenoisingIf::FingerprintExists ()**
  Returns 1 if a fingerprint exists and 0 if not.

Utilitary Functions

- **DenoisingError_t CDenoisingIf::GetSpectrum (Spectra_t iSpecIdx, float *pf↩ SpectrumBuffer, int iBlockSize, int iChannel)**
  Writes the current requested spectrum with index iSpecIdx into the buffer pf↩ SpectrumBuffer with size iBlockSize. The current processing block size is returned by function GetBlockSize(). The spectrum indices are listed in enum CDenoisingIf::Spectra_t.

- **int CDenoisingIf::GetBlockSize ()**
  Returns the current processing block size. The block size can be between [256...4096] depending on the current sample rate.

Processing parameters The parameters are defined in enum CDenoisingIf::DenoisingParam_t. Below is a short description of all parameters, their ranges and their usage:

- **kReductionFactor**
  The reduction factor scales the noise power spectrum from [0..1], the closer to 1 the higher noise the reduction. This parameter can be adjusted during processing.

- **kVibrance**
  Controls the influence that previous denoising estimates have over the current estimate, in the interval `0, 1` Lower values produce a more dry, potentially muffled output, whereas higher values sound more vibrant or even metallic.

- **kNumArParam**
  Scales the number of model parameters in percentage of kBlockSize/2 [0 = 2, 1 = kBlockSize/2] which are used to estimate the current noise power spectrum. The higher the number of parameters the more accurate is the noise spectrum estimation and the higher is computing time. This parameter should be fixed during processing.

- **kSpectralThresholding**
  Using spectral thresholding high frequency components will be preserved. The spectral thresholding factor scales the amount of spectral thresholding during adaptive noise power spectrum estimation in percentage between [0..1] with 0 being lowest and 1 being the highest amount of spectral thresholding. This parameter can be adjusted during processing.

- **kIsMono**
  Switch whether the signal should be handled as mono or not. If the signal is a stereo or multi channel signal and kIsMono is enabled the input buffer is downmixed to one mono buffer and the processing is done only for this mono buffer. Otherwise the processing is done for each channel. This parameter should be fixed during processing.

- **kUseFingerprint**
  Switch whether a precalculated noise power spectrum fingerprint should be used or not. To get a new fingerprint do ResetInstance() followed by Capture↩ Fingerprint(.) and then CalculateFingerprint(.). If no fingerprint exists the processing is automatically done using the adaptive noise power spectrum estimation. This parameter can be adjusted during processing.

- **kFingerprintCrossmixFactor**
  It is also possible to use both a precalculated and adaptive fingerprint. Using this factor the crossmix between both fingerprints is done. This parameter can be adjusted during processing.

- **kOutputNoiseOnly**
  Switch whether subtracted noise will only be outputed. This is useful to adjust the amount of noise reduction. This parameter can be adjusted during processing.

- **kLowFreqNoiseFactor**
  The low frequency noise factor adjustes the amount extra low frequency noise reduction in percentage between [0 to 1]. This parameter can be adjusted during processing.

- **kLowFreqNoiseCutOffFreq**
  The cut-off frequency of extra low frequency noise reduction can be controled using this parameter in percentage between [0 to 1] = [0 to 1 kHz]. This parameter can be adjusted during processing.

Error Codes The errors are defined in enum CDenoisingIf::DenoisingError_t. Below is a short description of possible error codes:

- **kNoError**
  Indicates that no error occurred.

- **kMemError**
  Indicates that the memory allocation failed.
- **kInvalidFunctionParamError**
  Indicates that one or more function parameters are not valid.
- **kNotInitializedError**
  Indicates that the instance has not been initialized yet.
- **kNoFingerprint**
  Indicates that the processing is done using a fingerprint but no fingerprint was calculated yet.
- **kFingerprintNotValid**
  Indicates that the calculated fingerprint is not valid (selection was equal to zero).
- **kFingerprintAlreadyExists**
  Indicates that a fingerprint already exists and it was tried to recalculate one. To get a new fingerprint do ResetInstance() followed by CaptureFingerprint(.) and then CalculateFingerprint(.).
- **kUnknownError**
  Indicates that an unknown error occurred.

1.3.3 Declicking

Instance Handling Functions

- **DeclickingError_t CDeclickingIf::CreateInstance (CDeclickingIf ∗&pCInstance↩ Pointer, float fSampleRate, int iNumberOfChannels)**
  Creates a new instance of the [Declicking] declicker. The handle to the new instance is returned in parameter pCInstancePointer. The sample rate and the number of channels of the audio data to be analyzed is given in parameter f↩ SampleRate resp. iNumberOfChannels. The return value indicates whether an error occurred or not.
- **DeclickingError_t CDeclickingIf::DestroyInstance (CDeclickingIf ∗& pC↩ InstancePointer)**
  Destroys an instance of the [Declicking] declicker. The handle to instance is in parameter pCInstancePointer, which is set to 0 by this function. The return value indicates whether an error occurred or not.
- **DeclickingError_t CDeclickingIf::ResetInstance ()**
  Resets all internal variables and buffers to the default state. The return value indicates whether an error occurred or not.

Process Functions

- **DeclickingError_t CDeclickingIf::Process (float ∗∗ppfInputBuffer, float ∗∗ppf↩ OutputBuffer, int iNumberOfFrames)**
  Does the actual Declicking of the input buffer. The result is written into the output buffer. The return value indicates whether an error occurred or not. The processing can be controled by a various number of parameters which are combined together in the enum CDeclickingIf::DeclickingParam_t and will be explained in detail later on.

Parameter Retrieving and Setting Functions

- **DeclickingError_t CDeclickingIf::GetParamValue (DeclickingParam_t iParamIdx, float &fValue)**
  Writes parameter with index iParamIdx to fValue. The parameter indices are listed in enum CDeclickingIf::DeclickingParam_t{ }. The return value indicates whether an error occurred or not.

- **DeclickingError_t CDeclickingIf::SetParamValue (DeclickingParam_t iParamIdx, float fValue)**
  Sets parameter with index iParamIdx with fValue. The parameter indices are listed in enum CDeclickingIf::DeclickingParam_t. The return value indicates whether an error occurred or not.

Processing parameters The parameters are defined in enum CDeclickingIf::DeclickingParam_t. Below is a short description of all parameters, their ranges and their usage:

- **kSensitivity**
  The sensitivity of the click detection algorithm is adjusted via this parameter in percentage from [0..1]. A value close to 0 forces the algorithm to detect only large clicks while smaller clicks will be ignored. Values close to 1 will force the algorithm to detect even small crackles, but can cause an increased false alarm rate. This parameter can be adjusted during processing.

- **kMinThresholddB**
  The minimum click amplitude threshold can be adjusted using the parameter from [-70..-20] dB. Detected clicks with a lower amplitude will be ignored. This parameter can be adjusted during processing.

- **kLmaxSec**
  This parameter adjustes maximum click length in seconds between [0.001..0.01]. According to this parameter the interpolation window length is scaled. This parameter can be adjusted during processing.

- **kLintSec**
  This parameter adjustes interpolation filter model length in seconds between [0.001..0.01]. The higher the length the more accur is the interpolation but the higher is the computation time. After a change of this parameter CDeclickingIf::ResetInstance() has to be executed to reset the writing position of all buffers. This parameter should be fixed during processing.

- **kIsMono**
  Switch whether the signal should be handled as mono or not. If the signal is a stereo or multi channel signal and kIsMono is enabled the input buffer is downmixed to one mono buffer and the processing is done only for this mono buffer. Otherwise the processing is done for each channel. This parameter should be fixed during processing.

Error Codes The errors are defined in enum CDeclickingIf::DeclickingError_t. Below is a short description of possible error codes:

- **kNoError**
  Indicates that no error occurred.

- **kMemError**
  Indicates that the memory allocation failed.
- **kInvalidFunctionParamError**
  Indicates that one or more function parameters are not valid.
- **kNotInitializedError**
  Indicates that the instance has not been initialized yet.
- **kUnknownError**
  Indicates that an unknown error occurred.

1.4 Support

Support for the source code is - within the limits of the agreement - available from:

zplane.development
grunewaldstr. 83
d-10965 berlin
germany

fon: +49.30.854 09 15.0
fax: +49.30.854 09 15.5

@: info@zplane.de

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Class Documentation

4.1 CDeclickingIf Class Reference

Collaboration diagram for CDeclickingIf:

4.2 CDenoisingIf Class Reference

Collaboration diagram for CDenoisingIf:

Public Types

- enum DenoisingParam_t {
  kReductionFactor, kVibrance, kNumArParam, kSpectralThresholding,
  kIsMono, kUseFingerprint, kFingerprintCrossmixFactor, kOutputNoiseOnly,
  kLowFreqNoiseFactor, kLowFreqNoiseCutOffFreq, kNumParams }
- enum Spectra_t {
  kInputSpectrum, kOutputspectrum, kNoiseSpectrum, kFingerprintSpectrum,
  kNoiseReductionFunction, kNumSpectra }
- enum DenoisingError_t {
  kNoError, kMemError, kInvalidFunctionParamError, kNotInitializedError,
  kNoFingerprint, kFingerprintNotValid, kFingerprintAlreadyExists, kUnknownError,
  kNumErrors }
- enum Version_t {
  kMajor, kMinor, kPatch, kBuild,
  kNumVersionInts }

Public Member Functions

- virtual DenoisingError_t ResetInstance ()=0
- virtual DenoisingError_t GetParamValue (DenoisingParam_t iParamIdx, float &f↩ Value)=0
- virtual DenoisingError_t SetParamValue (DenoisingParam_t iParamIdx, float f↩ Value)=0
- virtual DenoisingError_t Process (float const ∗const ∗const ppfInputBuffer, float ∗const ∗const ppfOutputBuffer, int iNumberOfFrames)=0
- virtual int GetLatencyInSamples ()=0
- virtual DenoisingError_t CaptureFingerprint (float const ∗const ∗const ppfInput↩ Buffer, int iNumberOfFrames)=0
- virtual DenoisingError_t CalculateFingerprint ()=0
- virtual DenoisingError_t ResetFingerprint ()=0
- virtual int FingerprintExists ()=0
- virtual DenoisingError_t GetSpectrum (Spectra_t iSpecIdx, float ∗pfSpectrum↩ Buffer, int iChannel)=0
- virtual DenoisingError_t SetFingerprintSpectrum (float ∗pfSpectrumBuffer, int iChannel)=0
- virtual int GetBlockSize ()=0

Static Public Member Functions

- static const int GetVersion (const Version_t eVersionIdx)
- static const char * GetBuildDate ()
- static DenoisingError_t CreateInstance (CDenoisingIf *&pCInstancePointer, float fSampleRate, int iNumberOfChannels)
- static DenoisingError_t DestroyInstance (CDenoisingIf *&pCInstancePointer)

4.2.1 Member Enumeration Documentation

DenoisingError_t `enum CDenoisingIf::DenoisingError_t`

Enumerator

kNoError	no error occurred
kMemError	memory allocation failed
kInvalidFunctionParamError	one or more function parameters are not valid
kNotInitializedError	instance has not been initialized yet
kNoFingerprint	no fingerprint was calculated
kFingerprintNotValid	calculated fingerprint is not valid (selection was equal to zero)
kFingerprintAlreadyExists	fingerprint already exists -> recapture fingerprint
kUnknownError	unknown error occurred
kNumErrors	

DenoisingParam_t `enum CDenoisingIf::DenoisingParam_t`

Enumerator

kReductionFactor	reduction factor scales the noise power spectrum from [0 to 1], the closer to 1 the higher is the reduction
kVibrance	influence of previous denoising estimates over the current estimate, in the interval `0, 1`
kNumArParam	number of autoregressive parameters in percentage of kBlockSize/2 [0 = 2, 1 = kBlockSize/2]
kSpectralThresholding	factor for spectral thresholding
kIsMono	switch whether the signal should be handled as mono or not
kUseFingerprint	switch whether noise spectrum fingerprint is used or not
kFingerprintCrossmixFactor	scale the ratio between precalculated and adaptive noise fingerprint

Enumerator

kOutputNoiseOnly	switch whether subtracted noise will be outputed
kLowFreqNoiseFactor	adjust the amount of extra low frequency noise reduction [0 to 1]
kLowFreqNoiseCutOffFreq	adjust the cut-off frequency of extra low frequency noise reduction [0 to 1] = [0 to 1 kHz]
kNumParams	

Spectra_t `enum CDenoisingIf::Spectra_t`

Enumerator

kInputSpectrum	input magnitude spectrum
kOutputspectrum	output magnitude spectrum
kNoiseSpectrum	noise magnitude spectrum
kFingerprintSpectrum	fingerprint magnitude spectrum
kNoiseReductionFunction	noise reduction function between [0..1]
kNumSpectra	

Version_t `enum CDenoisingIf::Version_t`

Enumerator

kMajor	
kMinor	
kPatch	
kBuild	
kNumVersionInts	

4.2.2 Member Function Documentation

CalculateFingerprint() `virtual DenoisingError_t CDenoisingIf::CalculateFingerprint ( ) [pure virtual]`

Calculates the noise fingerprint using the captured magnitude spectra. To reset the fingerprint CDenoisingIf::ResetFingerprint() has to be executed. The return value indicates whether an error occurred or not.

Returns

DenoisingError_t :

CaptureFingerprint() `virtual DenoisingError_t CDenoisingIf::CaptureFingerprint (`

```
float const *const *const ppfInputBuffer,
int iNumberOfFrames ) [pure virtual]
```

Captures the magnitude spectrum using the samples in ppfInputBuffer and adds them to an internal buffer. To reset the fingerprint CDenoisingIf::ResetFingerprint() has to be executed. The return value indicates whether an error occurred or not.

Parameters

ppfInputBuffer	: input buffer of length iNumberOfFrames
iNumberOfFrames	: length of in- and output buffers

Returns

DenoisingError_t :

CreateInstance() `static DenoisingError_t CDenoisingIf::CreateInstance (`

```
CDenoisingIf *& pCInstancePointer,
float fSampleRate,
int iNumberOfChannels ) [static]
```

Creates a new instance of the denoiser. The handle to the new instance is returned in parameter pCInstancePointer. The sample rate and the number of channels of the audio data to be analyzed is given in parameter fSampleRate resp. iNumberOfChannels. The return value indicates whether an error occurred or not.

Parameters

pCInstancePointer	: instance of denoising class
fSampleRate	: sample rate
iNumberOfChannels	: number of audio channels

Returns

DenoisingError_t :

DestroyInstance() `static DenoisingError_t CDenoisingIf::DestroyInstance (`

```
CDenoisingIf *& pCInstancePointer ) [static]
```

Destroys an instance of the denoiser. The handle to instance is in parameter pC↩InstancePointer, which is set to 0 by this function. The return value indicates whether an error occurred or not.

Parameters

pCInstancePointer	: instance of class denoising

Returns

DenoisingError_t :

FingerprintExists() `virtual int CDenoisingIf::FingerprintExists ( ) [pure virtual]`

Returns

int : 1 if fingerprint exists else 0

GetBlockSize() `virtual int CDenoisingIf::GetBlockSize ( ) [pure virtual]`

Returns the current processing block size. The block size can be between [256...4096] depending on the current sample rate.

Returns

int : block size

GetBuildDate() `static const char* CDenoisingIf::GetBuildDate ( ) [static]`

GetLatencyInSamples() `virtual int CDenoisingIf::GetLatencyInSamples ( ) [pure virtual]`

Returns the fixed latency in samples.

Returns

int : latency in samples

GetParamValue() `virtual DenoisingError_t CDenoisingIf::GetParamValue ( DenoisingParam_t iParamIdx, float & fValue ) [pure virtual]`

Writes parameter with index iParamIdx to fValue. The parameter indices are listed in enum CDenoisingIf::DenoisingParam_t. The return value indicates whether an error occurred or not.

Parameters

iParamIdx	: index of enum DenoisingParam_t
&	fValue : output pointer

Returns

DenoisingError_t :

GetSpectrum() `virtual DenoisingError_t CDenoisingIf::GetSpectrum (`
`Spectra_t iSpecIdx,`
`float * pfSpectrumBuffer,`
`int iChannel ) [pure virtual]`

Writes the current requested spectrum with index iSpecIdx into the buffer pf↩ SpectrumBuffer with size iBlockSize. The current processing block size is returned by function GetBlockSize(). The spectrum indices are listed in enum CDenoisingIf::Spectra_t.

Parameters

index	: index of requested spectrum (enum Spectra_t)
pfSpectrumBuffer	: buffer for requested spectrum (needs to have a size of GetBlockSize()/2)
iChannel	: current audio channel

Returns

DenoisingError_t :

GetVersion() `static const int CDenoisingIf::GetVersion (`
`const Version_t eVersionIdx ) [static]`

Process() `virtual DenoisingError_t CDenoisingIf::Process (`
`float const *const *const ppfInputBuffer,`
`float *const *const ppfOutputBuffer,`
`int iNumberOfFrames ) [pure virtual]`

Does the actual denoising of the input buffer. The result is written into the output buffer. If the usage of a noise fingerprint is enabled and a fingerprint was captured, the noise reduction is done using the precalculated fingerprint. Otherwise an adaptive noise fingerprint is estimated for each frame. The return value indicates whether an error occurred or not. The processing can be controlled by a various number of parameters which are combined together in the enum CDenoisingIf::DenoisingParam_t.

Parameters

ppfInputBuffer	: input buffer of length iNumberOfFrames
ppfOutputBuffer	: output buffer of length iNumberOfFrames (input and output can be the same)
iNumberOfFrames	: length of in- and output buffers

Returns

DenoisingError_t :

ResetFingerprint() `virtual DenoisingError_t CDenoisingIf::ResetFingerprint ( ) [pure virtual]`

Resets the current Fingerprint to default value. The return value indicates whether an error occurred or not.

Returns

DenoisingError_t :

ResetInstance() `virtual DenoisingError_t CDenoisingIf::ResetInstance ( ) [pure virtual]`

Resets all internal variables and buffers to the default state.

Returns

DenoisingError_t :

SetFingerprintSpectrum() `virtual DenoisingError_t CDenoisingIf::SetFingerprint↩Spectrum (`

```
float * pfSpectrumBuffer,
int iChannel ) [pure virtual]
```

replaces the internal fingerprint spectrum per channel optimally a modified spectrum retrieved by GetSpectrum() is used.

Parameters

pfSpectrumBuffer	: buffer for the fingerprint spectrum (needs to have a size of GetBlockSize()/2), values may not be smaller than FLOAT_MIN
iChannel	: current audio channel

Returns

DenoisingError_t :

SetParamValue() `virtual DenoisingError_t CDenoisingIf::SetParamValue ( DenoisingParam_t iParamIdx, float fValue ) [pure virtual]`

Sets parameter with index iParamIdx with fValue. The parameter indices are listed in enum CDenoisingIf::DenoisingParam_t. The return value indicates whether an error occurred or not.

Parameters

iParamIdx	: index of enum DenoisingParam_t
fValue	: parameter value

Returns

DenoisingError_t :

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

- DenoisingIf.h

5 File Documentation

5.1 DeclickingIf.h File Reference

interface of the CDeclickingIf class.

Classes

- class CDeclickingIf

5.1.1 Detailed Description

:

5.2 DenoisingIf.h File Reference

interface of the CDenoisingIf class.

Classes

- class CDenoisingIf

5.2.1 Detailed Description

:

5.3 docugen.txt File Reference

5.3.1 Detailed Description

source documentation main file