Metering SDK 2.1.8

by zplane.development

February 15, 2018

Contents

1 Metering Documentation

1.1 Introduction

The Metering SDK offers implements several approaches of sample accurate level measurements of an input audio source to emulate typical audio mixing meter behaviours.

1.2 SDK Content

1.2.1 Folder Structure

Documentation This documentation and all other documentation can be found in the directory **./doc**.

Project Files The MS VisualC++-Solution (.sln) and all single Projectfiles (.vcproj) can be found in the directory **./build** and its subfolders, where the subfolders names correspond to the project names.

Source Files All source files are in the directory **./src** and its subfolders, where the subfolder names equally correspond to the project names.

Include Files If include files are project-intern, they are in the source directory **./src** of the project itself. If include files are to be included by other projects, they can be found in **./src/incl**. The main interface header of the SDK can be found in **./inc**.

Resource Files The resource files, if present, can be found in the subdirectory **./res** of the corresponding build-directory.

Library Files The directory **./lib** is for used and built libraries.

Binary Files The final executable (as well as the distributable Dynamic Link Libraries if contained in the project) can be found in the directory **./bin/release**. In debug-builds, the binary files are in the subfolder **./bin/Debug**.

Temporary Files The directory **./tmp** is for all temporary files while building the projects, structured into project and configuration names.

1.2.2 Project Structure

The project structure is as following:

- **libMetering**: The actual DynamicRangeCompression-library. Consists of the following files:
  - MeteringIf.h: SDK-interface for PPM, VU, RMS and TruePeak metering
  - LoudnessIf.h: SDK-interface for EBU R128 loudness metering

  The project output is a Static Library (Lib).
- **MeteringTestCL**: Application using the SDK. Consists of the following files:
  - MeteringTestCLMain.cpp: example code showing how to integrate the SDK.
  - LoudnessTestCLMain.cpp: example code for integrating loudness metering.

  The project output is an executable binary (EXE).

1.2.3 Project Configurations

For all projects included in the workspace, the default configurations Win32 Release and Win32 Debug are available.

1.3 API Documentation

The interface of the SDK is based on the push principle: succeeding blocks of input audio frames are pushed into the process function. Internal memory cannot be accessed from outside, while external memory (e.g. the audio buffer) will not be altered during API function calls, except the result buffer.
The SDK is capable of running multiple instances at the same time, but the API is not threadsafe.

1.3.1 Naming Conventions

When talking about **frames**, the number of audio samples per channel is meant. I.e. 512 stereo frames correspond to 1024 float values (samples). If the sample size is 32bit float, one sample has a memory usage of 4 byte.

1.4 CMeteringIf - SDK-interface for PPM, VU, RMS and True Peak metering

1.4.1 Instance Control Functions

The following functions have to be called when using the Metering library:

- **CMeteringIf::CreateInstance (CMeteringIf*& pCInstancePointer, int iSampleRate, int iNumberOfChannels, CMeteringIf::MeterTypes_t eType)**
  Creates a new instance of metering. The handle to the new instance is written to the variable pCInstancePointer, the audio sample rate in Hz is in parameter iSampleRate, the nunmber of audio channels in function parameter iNumberOfChannels, and the metering type as declared in CMeteringIf::MeterTypes_t in parameter eType.
  The function returns 0 in case of no error.

- **CMeteringIf::DestroyInstance (CMeteringIf*& pCInstancePointer)**
  Destroys an instance of metering. The handle to the instance to be destroyed is variable pCInstancePointer and is set to NULL upon success.
  The function returns 0 in case of no error.

1.4.2 Parameter Functions

- **CMeteringIf::SetParam (CMeteringIf::Parameters_t eParamIndex, float fParamValue)**
  Sets the time constants for the metering, as defined in CMeteringIf::Parameters_t. The first function value is the parameter index, the second the corresponding value. ParameterIndex kParamTime1InMs is - for all modes except kPpm - the integration time, kParamTime2InMs is not being used in these modes. For the meter type kPpm, kParamTime1InMs is the attack time in ms and kParamTime2InMs is the release time in ms.
  The function returns 0 in case of no error.

- **CMeteringIf::GetParam (CMeteringIf::Parameters_t eParamIndex)**
  Returns the value of the paramete with index eParamIndex.

1.4.3 Process Function

- **::CMeteringIf::Process (float *pfInputBufferInterleaved, float *pfOutputBufferInterleaved, int iNumberOfFrames)**

Processes a block of interleaved audio data of length iNumberOfFrames and writes the metering result into buffer pfOutputBufferInterleaved.
The function returns 0 in case of no error.

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 CLoudnessIf Class Reference

Implements loudness metering according to the EBU R128 recommendation.

```
#include <LoudnessIf.h>
```

Public Types

- enum ChannelType_t {
  kFront, kSurround, kLFE, kOther,
  kNumChannelTypes }

  The possible channel types in a surround setup for SetChannelConfig()
- enum Version_t {
  kMajor, kMinor, kPatch, kBuild,
  kNumVersionInts }

Public Member Functions

- virtual void SetChannelConfig (CLoudnessIf::ChannelType_t *aeChannelTypes)=0

  Configure the channel order for multi-channel audio signals.
- virtual void Process (float **ppfSampleData, int iNumFrames)=0

  Process the next incoming audio block.
- virtual float ComputeLoudness ()=0

  Calculate the integrated loudness of the based on the audio signal passed to the Process function.
- virtual float GetMomentaryLoudness () const =0

  Calculate the current momentary loudness in LUFS.
- virtual float GetShortTermLoudness () const =0

  Calculate the current short-term loudness in LUFS.
- virtual float GetMaximumMomentaryLoudness () const =0

  Calculate the maximum value for momentary loudness (while integrated metering is running)
- virtual float GetMaximumShortTermLoudness () const =0

  Calculate the maximum value for short-term loudness (while integrated metering is running)
- virtual float GetProgrammeLoudness () const =0

  devlivers the integrated loudness that has calculated in the ComputeLoudness function
- virtual float GetLoudnessRange () const =0

  Calculate the loudness range of the complete signal.
- virtual float GetLoudnessRange (float &fLow, float &fHigh) const =0

  Calculate the loudness range of the complete signal.
- virtual bool IsRunning () const =0

  Check if the integrated loudness metering is currently running.
- virtual void Reset ()=0

  Reset the results of integrated loudness metering.
- virtual void Pause ()=0

  Suspend integrated loudness metering.
- virtual void Start ()=0

  Continue integrated loudness metering after it was suspended.
- virtual void ResetInstance ()=0

  Completely reset this instance to its initial state.

Static Public Member Functions

- static int CreateInstance (CLoudnessIf *&pCInstancePointer, int iSampleRate, int iNumberOfChannels, unsigned long long ulMaxProgrammeLengthInFrames, int iMaxBlockSize)

  Create a new instance and starts integrated metering.
- static int DestroyInstance (CLoudnessIf *&pCInstancePointer)

  Destroy an instance of CLoudnessIf and free all resources.
- static const int GetVersion (const Version_t eVersionIdx)
- static const char * GetBuildDate ()

Static Public Attributes

- static const float kfMinLUFS

  returned by GetProgrammeLoudness() if all signal was below the absolute gating threshold of -70 LUFS

- static const float kfUndefinedLUFS

  returned by GetProgrammeLoudness() and GetLoudnessRange() if integrated metering results are not (yet) available

4.1.1 Detailed Description

Implements loudness metering according to the EBU R128 recommendation.
Definition at line 43 of file LoudnessIf.h.

4.1.2 Member Enumeration Documentation

ChannelType_t `enum CLoudnessIf::ChannelType_t`

The possible channel types in a surround setup for SetChannelConfig()

Enumerator

kFront	one of the front channels (L,R,C) that is used unweighted
kSurround	a surround channel that is weighted with +1.5dB
kLFE	the LFE channel which is ignored by the loudness meter
kOther	all channels added to 5.1 are unweighted see BS 1770-4
kNumChannelTypes	

Definition at line 48 of file LoudnessIf.h.

```
49      {
51          kFront,
53          kSurround,
55          kLFE,
57          kOther,
58
59          kNumChannelTypes
60      };
```

Version_t `enum CLoudnessIf::Version_t`

Enumerator

kMajor	
kMinor	
kPatch	
kBuild	
kNumVersionInts	

Definition at line 244 of file LoudnessIf.h.

```
245     {
246         kMajor,
247         kMinor,
248         kPatch,
249         kBuild,
250 
251         kNumVersionInts
252     };
```

4.1.3 Member Function Documentation

ComputeLoudness() `virtual float CLoudnessIf::ComputeLoudness ( ) [pure virtual]`

Calculate the integrated loudness of the based on the audio signal passed to the Process function.

You can call this method while measuring, but you should avoid to do so in the audio thread if processing in real time. If integrated metering wasn't running long enough (since creation of this instance or the last call of Reset), this method will return CLoudnessIf::kfUndefinedLUFS to signal that there is no valid result available. It will return CLoudnessIf::kfMinLUFS if all input was below the absolute gating threshold.

Returns

the programme loudness in LUFS (or CLoudnessIf::kfUndefinedLUFS / CLoudnessIf::kfMinLUFS)

Referenced by main().

CreateInstance()

```
static int CLoudnessIf::CreateInstance (
            CLoudnessIf *& pCInstancePointer,
            int iSampleRate,
            int iNumberOfChannels,
            unsigned long long ulMaxProgrammeLengthInFrames,
            int iMaxBlockSize ) [static]
```

Create a new instance and starts integrated metering.

After creating, the instance returned in the pCInstancePointer reference is immediately ready for metering. If you don't want the integrated loudness metering to start with the first frame, you should call Pause() before calling Process().

None of the parameters can be changed afterwards, so you have to destroy and create a new new instance for each signal to be measured.

Parameters

pCInstancePointer	: handle to the new instance
iSampleRate	: sample rate of the audio signal to be analyzed (in Hz)
iNumberOfChannels	: number of channels in the audio signal to be analyzed
ulMaxProgrammeLengthInFrames	: the length of the audio signal
iMaxBlockSize	: the largest block size to expect

Returns

int : 0 when no error

Referenced by main().

DestroyInstance() `static int CLoudnessIf::DestroyInstance ( CLoudnessIf *& pCInstancePointer ) [static]`

Destroy an instance of CLoudnessIf and free all resources.

The handle to the instance to be destroyed is passed as the pCInstancePointer reference which is set to NULL upon success.

Parameters

pCInstancePointer	: handle to the instance to be destroyed

Returns

int : 0 when no error

Referenced by main().

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

Referenced by CLShowProgInfo().

GetLoudnessRange() `[1/2] virtual float CLoudnessIf::GetLoudnessRange ( ) const [pure virtual]`

Calculate the loudness range of the complete signal.

You can call this method while measuring, but you should avoid to do so in the audio thread if processing in real time. If integrated metering wasn't running long enough (since creation of this instance or the last call of Reset), this method will return CLoudnessIf::kfUndefinedLUFS to signal that there is no valid result available.

Returns

the loudness range in LU (or CLoudnessIf::kfUndefinedLUFS)

Referenced by main().

GetLoudnessRange() `[2/2] virtual float CLoudnessIf::GetLoudnessRange ( float & fLow, float & fHigh ) const [pure virtual]`

Calculate the loudness range of the complete signal.

Same as GetLoudnessRange() with access to the upper and lower bounds the range is calculated from.

Parameters

fLow	: reference to return the lower bounds of the range in LUFS
fHigh	: reference to return the higher bounds of the range in LUFS

Returns

the loudness range in LU (or CLoudnessIf::kfUndefinedLUFS)

GetMaximumMomentaryLoudness() `virtual float CLoudnessIf::GetMaximum↩MomentaryLoudness ( ) const [pure virtual]`

Calculate the maximum value for momentary loudness (while integrated metering is running)

Returns

the maximum momentary loudness in LUFS (or CLoudnessIf::kfUndefinedLUFS if integrated metering wasn't started)

GetMaximumShortTermLoudness() `virtual float CLoudnessIf::GetMaximum↩ShortTermLoudness ( ) const [pure virtual]`

Calculate the maximum value for short-term loudness (while integrated metering is running)

Returns

the maximum short-term loudness in LUFS (or CLoudnessIf::kfUndefinedLUFS if integrated metering wasn't started)

GetMomentaryLoudness() `virtual float CLoudnessIf::GetMomentaryLoudness ( ) const [pure virtual]`

Calculate the current momentary loudness in LUFS.

You can call this immediately after Process() to get a continuously updating loudness value for the most recent 400 millisecond time window. The momentary loudness is always available even if integrated metering is paused or reset.

Returns

the momentary loudness in LUFS

GetProgrammeLoudness() `virtual float CLoudnessIf::GetProgrammeLoudness ( ) const [pure virtual]`

devlivers the integrated loudness that has calculated in the ComputeLoudness function

If integrated metering wasn't running long enough (since creation of this instance or the last call of Reset), this method will return CLoudnessIf::kfUndefinedLUFS to signal that there is no valid result available. It will return CLoudnessIf::kfMinLUFS if all input was below the absolute gating threshold.

Returns

the programme loudness in LUFS (or CLoudnessIf::kfUndefinedLUFS / CLoudnessIf::kfMinLUFS)

GetShortTermLoudness() `virtual float CLoudnessIf::GetShortTermLoudness ( ) const [pure virtual]`

Calculate the current short-term loudness in LUFS.

You can call this immediately after Process() to get a continuously updating loudness value for the most recent 3 second time window. The short-term loudness is always available even if integrated metering is paused or reset.

Returns

the short-term loudness in LUFS

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

Referenced by CLShowProgInfo().

IsRunning() `virtual bool CLoudnessIf::IsRunning ( ) const [pure virtual]`

Check if the integrated loudness metering is currently running.

If true, the next incoming audio block passed to Process will be considered for the integrated metering results as returned by GetProgrammeLoudness() and GetLoudnessRange(). By default, integrated metering is started during instantiation. It can be suspended at any time with Pause() and continued with Start()

Returns

true if integrated metering is running

Pause() `virtual void CLoudnessIf::Pause ( ) [pure virtual]`

Suspend integrated loudness metering.

After calling this method, incoming audio blocks will not be considered for the integrated metering results as returned by GetProgrammeLoudness() and GetLoudnessRange(). This does not affect the results of GetMomentaryLoudness() and GetShortTermLoudness().

Process() `virtual void CLoudnessIf::Process (`
`float ** ppfSampleData,`
`int iNumFrames ) [pure virtual]`

Process the next incoming audio block.

The audio data is passed as an array of float pointers for each channel. The number of frames in the block must never exceed the maximum block size that was specified when creating this instance.

Parameters

ppfSampleData	: audio buffer of dimension [channels][frames]
iNumFrames	: number of frames in this block

Referenced by main().

Reset() `virtual void CLoudnessIf::Reset ( ) [pure virtual]`

Reset the results of integrated loudness metering.

This method can be called at any time to discard the current results returned by GetProgrammeLoudness(), GetLoudnessRange(), GetMaximumMomentaryLoudness and GetMomentaryLoudness() and start a new measurement beginning with the next incoming audio block. This method does not affect the results of GetMomentaryLoudness() and GetShortTermLoudness().

ResetInstance() `virtual void CLoudnessIf::ResetInstance ( ) [pure virtual]`

Completely reset this instance to its initial state.

SetChannelConfig() `virtual void CLoudnessIf::SetChannelConfig (`
`CLoudnessIf::ChannelType_t * aeChannelTypes ) [pure virtual]`

Configure the channel order for multi-channel audio signals.

You don't have to call this method when measuring loudness of a mono, stereo or 5.1 signal with default L,R,C,LFE,Ls,Rs channel order. For other channel orders, you need to create an array of ChannelType_t enums that specifies the kind of weighting to use for each channel. You can delete that array immediately after calling this method.

Parameters

aeChannelTypes	: an array with a ChannelType_t enum for each channel of the audio signal

Start() `virtual void CLoudnessIf::Start ( ) [pure virtual]`

Continue integrated loudness metering after it was suspended.

After calling this method, incoming audio blocks will be considered for the integrated metering results as returned by GetProgrammeLoudness() and GetLoudnessRange().

Call Reset() before starting if you also want to clear the intermediate results for programme loudness and range.

4.1.4 Member Data Documentation

kfMinLUFS `const float CLoudnessIf::kfMinLUFS [static]`
returned by GetProgrammeLoudness() if all signal was below the absolute gating threshold of -70 LUFS
Definition at line 63 of file LoudnessIf.h.

kfUndefinedLUFS `const float CLoudnessIf::kfUndefinedLUFS [static]`
returned by GetProgrammeLoudness() and GetLoudnessRange() if integrated metering results are not (yet) available
Definition at line 66 of file LoudnessIf.h.
The documentation for this class was generated from the following file:

- LoudnessIf.h

4.2 CMeteringIf Class Reference

`#include <MeteringIf.h>`

Public Types

- enum MeterTypes_t {
  kPpm, kPpmType1, kPpmType2, kRms,
  kVu, kLeqa, kBs1770, kPpmTp,
  kPpmTpHR, kTruePeak = kPpmTpHR, kNumMeterTypes }
- enum Parameters_t { kParamTime1InMs, kParamTime2InMs, kNumOfParameters }
- enum Maxima_t { kLastBlock, kOverall, kNumMaxima }
- enum Version_t {
  kMajor, kMinor, kPatch, kBuild,
  kNumVersionInts }

Public Member Functions

- virtual int Process (float **ppfInputBuffer, float **ppfOutputBuffer, int iNumberOfFrames)=0
- virtual int ApplyFilterFunction (float **ppfInputBuffer, float **pfOutputBuffer, int iNumberOfFrames)=0
- virtual int SetParam (Parameters_t eParamIndex, float fParamValue)=0
- virtual float GetParam (Parameters_t eParamIndex)=0
- virtual int SetAddDenormalNoise (bool bAddNoise=true)=0
- virtual bool GetAddDenormalNoise ()=0
- virtual int SetOutputInDB (bool bOutputInDB=true)=0
- virtual bool GetOutputInDB ()=0

- virtual int GetMax (Maxima_t eMaxType, float *pfMaxInChannel)=0
- virtual int Reset (bool bOnlyMax=true)=0

Static Public Member Functions

- static int CreateInstance (CMeteringIf *&pCInstancePointer, int iSampleRate, int iNumberOfChannels, MeterTypes_t eType)
- static int DestroyInstance (CMeteringIf *&pCInstancePointer)
- static const int GetVersion (const Version_t eVersionIdx)
- static const char * GetBuildDate ()

Protected Member Functions

- virtual ~CMeteringIf ()

4.2.1 Detailed Description

Definition at line 40 of file MeteringIf.h.

4.2.2 Member Enumeration Documentation

Maxima_t `enum CMeteringIf::Maxima_t`
defines the available maxima in CMEteringIf::GetMax

Enumerator

kLastBlock	the maximum as measured during the last process call
kOverall	the overall maximum since the last reset call
kNumMaxima	

Definition at line 71 of file MeteringIf.h.

```
72     {
73         kLastBlock,
74         kOverall,
75 
76         kNumMaxima
77     };
```

MeterTypes_t `enum CMeteringIf::MeterTypes_t`
defines the available metering types

Enumerator

kPpm	general PPM style meter
kPpmType1	IEC268/DIN45406 Type I meter.
kPpmType2	IEC268/DIN45406 Type II meter.

Enumerator

kRms	RMS measurement.
kVu	VU style meter.
kLeqa	RMS based A-weighted loudness measurement.
kBs1770	ITU-R BS.1770 based loudness measurement.
kPpmTp	ITU-R BS.1770 Annex B "true-peak" meter with fast and less correct resampling.
kPpmTpHR	ITU-R BS.1770 Annex B "true-peak" meter with slower but correct resampling.
kTruePeak	name in previous versions (obsolete!)
kNumMeterTypes	

Definition at line 44 of file MeteringIf.h.

```
45     {
46         kPpm,
47         kPpmType1,
48         kPpmType2,
49         kRms,
50         kVu,
51         kLeqa,
52         kBs1770,
53         kPpmTp,
54         kPpmTpHR,
55         // obsolete: for compatibility with older versions
56         kTruePeak = kPpmTpHR,
57 
58         kNumMeterTypes
59     };
```

Parameters_t `enum CMeteringIf::Parameters_t`
defines the available parameters

Enumerator

kParamTime1InMs	PPM : attack time in ms, RMS,VU,LEQA: integration time in ms.
kParamTime2InMs	PPM : release time in ms, not used for other modes.
kNumOfParameters	

Definition at line 62 of file MeteringIf.h.

```
63     {
64         kParamTime1InMs,
65         kParamTime2InMs,
66 
67         kNumOfParameters
68     };
```

Version_t `enum CMeteringIf::Version_t`

Enumerator

kMajor	
kMinor	
kPatch	
kBuild	
kNumVersionInts	

Definition at line 212 of file MeteringIf.h.

```
213     {
214         kMajor,
215         kMinor,
216         kPatch,
217         kBuild,
218 
219         kNumVersionInts
220     };
```

4.2.3 Constructor & Destructor Documentation

~CMeteringIf() `virtual CMeteringIf::~CMeteringIf ( ) [inline], [protected], [virtual]`

Definition at line 226 of file MeteringIf.h.

```
226 {};
```

4.2.4 Member Function Documentation

ApplyFilterFunction() `virtual int CMeteringIf::ApplyFilterFunction (`
`float ** ppfInputBuffer,`
`float ** pfOutputBuffer,`
`int iNumberOfFrames ) [pure virtual]`

only apply the filter function for the given metering-type on the given audio-signal (only useful for kLeqa & kBs1770)

Parameters

∗∗*ppfInputBuffer*	: pointer to channel-wise splitted audio data input in floating point format
∗∗*pfOutputBuffer*	: pointer to channel-wise splitted audio data output in floating point format may be the same as input buffer for inplace processing)
iNumberOfFrames	: number of frames per block (a frame equals one sample for mono signals and two samples for stereo signals), must not be higher than 16384

Returns

virtual int : 0 when no error

CreateInstance() `static int CMeteringIf::CreateInstance (`
`CMeteringIf *& pCInstancePointer,`
`int iSampleRate,`
`int iNumberOfChannels,`
`MeterTypes_t eType ) [static]`

creates a new instance of Metering

Parameters

pCInstancePointer	: handle to the new instance
iSampleRate	: sample rate of audio signal to be analyzed (in Hz)
iNumberOfChannels	: number of channels of audio signal to be analyzed
eType	: type of meter that is supposed to be used values: _PPM : peak program meter (works with default values, not for sure with longer attack times) _RMS : root mean square _VU : volume unit _LEQA : loudness measurement after LEQA (a-weighted equivalent loudness level)

Returns

static int : 0 when no error

Referenced by main().

DestroyInstance() `static int CMeteringIf::DestroyInstance (`
`CMeteringIf *& pCInstancePointer ) [static]`

destroys an instance of Metering

Parameters

pCInstancePointer	: handle to the instance to be destroyed

Returns

static int : 0 when no error

Referenced by main().

GetAddDenormalNoise() `virtual bool CMeteringIf::GetAddDenormalNoise ( ) [pure virtual]`

returns true if denomral noise is added internally

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

Referenced by CLShowProgInfo().

GetMax() `virtual int CMeteringIf::GetMax (`
`Maxima_t eMaxType,`
`float * pfMaxInChannel ) [pure virtual]`

get the maximum

Parameters

eMaxType	: get the block or the overall maximum
pfMaxInChannel	: maximum per channel

GetOutputInDB() `virtual bool CMeteringIf::GetOutputInDB ( ) [pure virtual]`

returns true if the output is in "amplitude"

GetParam() `virtual float CMeteringIf::GetParam (`
`Parameters_t eParamIndex ) [pure virtual]`

returns the parameters of the metering type

Parameters

eParamIndex	: index of parameter

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

Referenced by CLShowProgInfo().

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

processes the audio in blocks by using the given metering-type

Parameters

***ppfInputBuffer*	: pointer to channel-wise splitted audio data input in floating point format
***ppfOutputBuffer*	: pointer to channel-wise splitted audio data output in floating point format may be the same as input buffer for inplace processing)

Parameters

iNumberOfFrames	: number of frames per block (a frame equals one sample for mono signals and two samples for stereo signals), must not be higher than 16384

Returns

virtual int : 0 when no error

Reset() `virtual int CMeteringIf::Reset (`
`bool bOnlyMax = true ) [pure virtual]`

reset either internal buffers or the overall max

Parameters

bOnlyMax	: true if output is in dB

SetAddDenormalNoise() `virtual int CMeteringIf::SetAddDenormalNoise (`
`bool bAddNoise = true ) [pure virtual]`

allows to add small amount of noise to avoid denormals

Parameters

bAddNoise	: true if noise will be added

Referenced by main().

SetOutputInDB() `virtual int CMeteringIf::SetOutputInDB (`
`bool bOutputInDB = true ) [pure virtual]`

allows to select if output is in decibels or ”amplitude”; dB = 20∗log10(amplitude) for all scales

Parameters

bOutputInDB	: true if output is in dB

Referenced by main().

SetParam() `virtual int CMeteringIf::SetParam (`
`Parameters_t eParamIndex,`
`float fParamValue ) [pure virtual]`

sets the parameters of the metering type if SetParam is not used, Metering uses default values for each type

default values for each metering type : PPM : attack time : 10 ms release time : 1500 ms RMS : integration time : 300 ms VU : integration time : 300 ms LEQA: integration time : 300 ms

Parameters

eParamIndex	: index of parameter
fParamValue	: time in ms kTime1InMs : PPM : attack time RMS,VU,LEQA: integration time kTime2InMs : PPM : release time RMS,VU,LEQA: not in use

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

- MeteringIf.h

5 File Documentation

5.1 docugen.txt File Reference

5.1.1 Detailed Description

source documentation main file

5.2 LoudnessCLMain.cpp File Reference

```
#include <time.h>
#include <string>
#include <fstream>
#include <iostream>
#include "zplAudioFile.h"
#include "LoudnessIf.h"
#include <stdlib.h>
```

Macros

- #define kBlockSize (8192)
- #define kNumMinClArgs (2)
- #define kNumMaxChannels (8)

Functions

- static void CLShowProgInfo ()
- static void CLReadArgs (char ∗&pcInputPath, int argc, char ∗argv[ ])
- static void CLShowProcessTime (int iCurrentFrame, float fSampleRate)
- static void CLShowProcessedTime (clock_t clTime)
- int main (int argc, char ∗argv[ ])

5.2.1 Macro Definition Documentation

kBlockSize `#define kBlockSize (8192)`

Definition at line 33 of file LoudnessCLMain.cpp.
Referenced by main().

kNumMaxChannels `#define kNumMaxChannels (8)`

Definition at line 36 of file LoudnessCLMain.cpp.
Referenced by main().

kNumMinClArgs `#define kNumMinClArgs (2)`

Definition at line 35 of file LoudnessCLMain.cpp.
Referenced by main().

5.2.2 Function Documentation

CLReadArgs() `static void CLReadArgs (`
`char *& pcInputPath,`
`int argc,`
`char * argv[] ) [static]`

Definition at line 182 of file LoudnessCLMain.cpp.
Referenced by main().

```
183 {
184     pcInputPath     = argv[1];
185     return;
186 }
```

CLShowProcessedTime() `static void CLShowProcessedTime (`
`clock_t clTime ) [static]`

Definition at line 193 of file LoudnessCLMain.cpp.
Referenced by main().

```
194 {
195     fprintf(stdout, "\nTime elapsed:\t%2.2f sec\n", (float)(clTime) / CLOCKS_PER_SEC);
196
197     return;
198 }
```

CLShowProcessTime() `static void CLShowProcessTime (`
`int` *`iCurrentFrame`*`,`
`float` *`fSampleRate`* `)  [static]`

Definition at line 187 of file LoudnessCLMain.cpp.

Referenced by main().

```
188 {
189     fprintf(stderr, "\rProcessed:\t%2.2f seconds of audio data", iCurrentFrame*1.0F/fSampleRate);
190     return;
191 }
```

CLShowProgInfo() `static void CLShowProgInfo ( )  [static]`

Definition at line 167 of file LoudnessCLMain.cpp.

References CLoudnessIf::GetBuildDate(), CLoudnessIf::GetVersion(), CLoudness↩
If::kBuild, CLoudnessIf::kMajor, CLoudnessIf::kMinor, and CLoudnessIf::kPatch.

Referenced by main().

```
168 {
169     std::cout << "zplane.development Loudness Metering App" << std::endl;
170     std::cout << "(c) 2011 by zplane" << std::endl;
171 
172     std::cout << "V"
173         << CLoudnessIf::GetVersion (CLoudnessIf::kMajor) << "."
174         << CLoudnessIf::GetVersion (CLoudnessIf::kMinor) << "."
175         << CLoudnessIf::GetVersion (CLoudnessIf::kPatch) << "
      build: "
176         << CLoudnessIf::GetVersion (CLoudnessIf::kBuild) << ",
      date: "
177         << CLoudnessIf::GetBuildDate () << std::endl;
178 
179     return;
180 }
```

main() `int main (`
`int` *`argc`*`,`
`char * argv[] )`

Definition at line 46 of file LoudnessCLMain.cpp.

References CLReadArgs(), CLShowProcessedTime(), CLShowProcessTime(), C↩
LShowProgInfo(), CLoudnessIf::ComputeLoudness(), CLoudnessIf::CreateInstance(), CLoudnessIf::DestroyInstance(), CLoudnessIf::GetLoudnessRange(), kBlockSize, k↩
NumMaxChannels, kNumMinClArgs, and CLoudnessIf::Process().

```
47 {
48 
49     char                *pcInputPath    = 0;
50 
51     int                 i,
52                         iCurrentFrame   = 0;
53 
54     CzplAudioFile       *pCInputFile    = 0;
55 
56     clock_t             clTotalTime     = 0;
57     clock_t             clStartTime     = 0;
58 
59     float               *apfInputData[kNumMaxChannels];
60 
61     CLoudnessIf         *pCLoudnessHandle       = 0;        // instance handle
62 
```

```
63 #if (!defined(WITHOUT_MEMORY_CHECK) && defined(_DEBUG) && defined (WIN32))
64     // set memory checking flags
65     int iDbgFlag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
66     iDbgFlag       |= _CRTDBG_CHECK_ALWAYS_DF;
67     iDbgFlag       |= _CRTDBG_LEAK_CHECK_DF;
68     _CrtSetDbgFlag( iDbgFlag );
69 #endif
70
71 #if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined (WIN32))
72     // enable check for exceptions (don't forget to enable stop in MSVC!)
73     _controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW | _EM_UNDERFLOW | _EM_DENORMAL), _MCW_EM) ;
74 #endif // #ifndef WITHOUT_EXCEPTIONS
75
76     //check for correct number of command line arguments
77     if (argc < kNumMinClArgs)
78     {
79         std::cout << "Wrong number of command line arguments!" << std::endl
80                   << "Usage: LoudnessTestCL inputfile" << std::endl;
81         return -1;
82     }
83
84     CLShowProgInfo();
85
86     CLReadArgs( pcInputPath,
87                 argc,
88                 argv);
89
90     // open soundfile
91     pCInputFile = new CzplAudioFile();
92     pCInputFile->OpenReadFile(pcInputPath, kBlockSize);
93
94     if (!pCInputFile->IsFileOpen())
95     {
96         std::cout << "Input file could not be opened!" << std::endl;
97         delete pCInputFile;
98         return -1;
99     }
100     else if (pCInputFile->GetNumOfChannels() > kNumMaxChannels)
101     {
102         std::cout << "Invalid input channel count!" << std::endl;
103         pCInputFile->CloseFile();
104         delete pCInputFile;
105         return -1;
106     }
107
108     // allocate input sample buffers
109     for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
110     {
111         apfInputData[i] = new float [kBlockSize];
112     }
113
114     // get properties of the input file
115     int iSampleRate  = static_cast<int> (pCInputFile->GetSampleRate ());
116     int iNumChannels = pCInputFile->GetNumOfChannels();
117     int iNumFrames   = pCInputFile->GetFileSize();
118
119     // create loudness metering instance
120     CLoudnessIf::CreateInstance (pCLoudnessHandle, iSampleRate, iNumChannels,
    iNumFrames, kBlockSize);
121
122     // now we can begin to process!
123     while(iCurrentFrame < iNumFrames)
124     {
125         // read next block from input file
126         int iNumFramesRead = pCInputFile->Read(apfInputData, kBlockSize);
127
128         // process next block with the loudness meter, and sum up the processing times for all blocks
129         clStartTime = clock();
130         pCLoudnessHandle->Process(apfInputData, iNumFramesRead);
131         clTotalTime += (clock() - clStartTime);
132
133         // increase current frame pointer and show progress
134         iCurrentFrame += iNumFramesRead;
135         CLShowProcessTime(iCurrentFrame, pCInputFile->GetSampleRate ());
```

```
136     }
137     CLShowProcessedTime(clTotalTime);
138
139     // calculate the overall programme loudness and range
140     float fProgrammeLoudnessLUFS = pCLoudnessHandle->ComputeLoudness();
141     float fLoudnessRangeLU = pCLoudnessHandle->GetLoudnessRange();
142
143     // display results
144     std::cout << std::endl;
145     std::cout << "Programme loudness: " << (fProgrammeLoudnessLUFS) << " LUFS" << std::endl;
146     std::cout << "    Loudness range: " << (fLoudnessRangeLU) << " LU" << std::endl;
147     std::cout << std::endl;
148
149     // free input sample buffers
150     for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
151     {
152         delete [] apfInputData[i];
153     }
154
155     // destroy instance
156     CLoudnessIf::DestroyInstance(pCLoudnessHandle);
157
158     // close files
159     pCInputFile->CloseFile ();
160     delete pCInputFile;
161
162     return 0;
163
164 }
```

5.3 LoudnessIf.h File Reference

Contains the CLoudnessIf interface for loudness metering.

Classes

- class CLoudnessIf

  Implements loudness metering according to the EBU R128 recommendation.

5.3.1 Detailed Description

Contains the CLoudnessIf interface for loudness metering.
:

5.4 MeteringCLMain.cpp File Reference

```
#include <time.h>
#include <cstring>
#include <fstream>
#include <iostream>
#include "zplAudioFile.h"
#include "MeteringIf.h"
#include <stdlib.h>
```

Macros

- #define kBlockSize (8192)
- #define kNumMinClArgs (2)
- #define kNumMaxChannels (8)

Functions

- static void CLShowProgInfo ()
- static void CLReadArgs (char ∗&pcInputPath, char ∗&pcOutputPath, int argc, char ∗argv[ ])
- static void CLShowProcessTime (int iCurrentFrame, float fSampleRate, int i↩ BlocksizeFile)
- static void CLShowProcessedTime (clock_t clTime)
- static void ClUtilSplit2Interleaved (float ∗pfInterleavedBuffer, float ∗∗ppfSingle↩ Channels, int iNumChannels, int iNumFrames)
- int main (int argc, char ∗argv[ ])

5.4.1 Macro Definition Documentation

kBlockSize `#define kBlockSize (8192)`

Definition at line 33 of file MeteringCLMain.cpp.
Referenced by main().

kNumMaxChannels `#define kNumMaxChannels (8)`

Definition at line 36 of file MeteringCLMain.cpp.
Referenced by main().

kNumMinClArgs `#define kNumMinClArgs (2)`

Definition at line 35 of file MeteringCLMain.cpp.
Referenced by main().

5.4.2 Function Documentation

CLReadArgs() `static void CLReadArgs (`
`char *& pcInputPath,`
`char *& pcOutputPath,`
`int argc,`
`char * argv[] ) [static]`

Definition at line 213 of file MeteringCLMain.cpp.
Referenced by main().

```
214 {
215     pcInputPath     = argv[1];
216     pcOutputPath    = argv[2];
217     return;
218 }
```

CLShowProcessedTime() `static void CLShowProcessedTime (`
`clock_t clTime ) [static]`

Definition at line 225 of file MeteringCLMain.cpp.

Referenced by main().

```
226 {
227     fprintf(stdout, "\nTime elapsed:\t%2.2f sec\n", (float)(clTime) / CLOCKS_PER_SEC);
228
229     return;
230 }
```

CLShowProcessTime() `static void CLShowProcessTime (`
`int iCurrentFrame,`
`float fSampleRate,`
`int iBlocksizeFile ) [static]`

Definition at line 219 of file MeteringCLMain.cpp.

Referenced by main().

```
220 {
221     fprintf(stderr, "\rProcessed:\t%2.2f seconds of audio file", iCurrentFrame*iBlocksizeFile*1.0F/
    fSampleRate);
222     return;
223 }
```

CLShowProgInfo() `static void CLShowProgInfo ( ) [static]`

Definition at line 198 of file MeteringCLMain.cpp.

References CMeteringIf::GetBuildDate(), CMeteringIf::GetVersion(), CMeteringIf::kBuild, CMeteringIf::kMajor, CMeteringIf::kMinor, and CMeteringIf::kPatch.

Referenced by main().

```
199 {
200     cout << "zplane.development Metering App" << endl;
201     cout << "(c) 2000-2011 by zplane" << endl;
202     cout    << "V"
203         << CMeteringIf::GetVersion (CMeteringIf::kMajor) << "."
204         << CMeteringIf::GetVersion (CMeteringIf::kMinor) << "."
205         << CMeteringIf::GetVersion (CMeteringIf::kPatch) << "
      build: "
206         << CMeteringIf::GetVersion (CMeteringIf::kBuild) << ",
      date: "
207         << CMeteringIf::GetBuildDate () << endl;
208     cout << "Press Escape to cancel..." << endl << endl;
209
210     return;
211 }
```

ClUtilSplit2Interleaved() `static void ClUtilSplit2Interleaved (`
`float * pfInterleavedBuffer,`
`float ** ppfSingleChannels,`
`int iNumChannels,`
`int iNumFrames ) [static]`

Definition at line 44 of file MeteringCLMain.cpp.

```
45 {
46     int iIlIdx = 0;
47     for (int i = 0; i < iNumFrames; i++)
48         for (int c = 0; c < iNumChannels; c++,iIlIdx++)
49             pfInterleavedBuffer[iIlIdx] = ppfSingleChannels[c][i];
50     return;
51 }
```

main() `int main (`
`int` *`argc,`*
`char * argv[] )`

Definition at line 55 of file MeteringCLMain.cpp.

References CLReadArgs(), CLShowProcessedTime(), CLShowProcessTime(), CLShowProgInfo(), CMeteringIf::CreateInstance(), CMeteringIf::DestroyInstance(), kBlockSize, kNumMaxChannels, kNumMinClArgs, CMeteringIf::kPpmTp, CMeteringIf::SetAddDenormalNoise(), and CMeteringIf::SetOutputInDB().

```
56 {
57
58     char                *pcInputPath    = 0,
59                         *pcOutputPath   = 0;
60
61     int                 i,
62                         iNumFramesRead,
63                         iCurrentFrame   = 0;
64
65     bool                bReadNextFrame  = true;
66
67     CzplAudioFile       *pCInputFile    = 0;
68
69     std::ofstream       FOutputFile;
70
71     clock_t             clTotalTime = 0;
72     clock_t             clStartTime     = 0;
73
74     float               *apfSplitInputData[kNumMaxChannels],
75                         afInputData[kNumMaxChannels*kBlockSize],
76                         afOutputData[kNumMaxChannels*kBlockSize];
77
78     CMeteringIf         *pCMeteringHandle       = 0;        // instance handles
79
80
81     //check for correct number of command line arguments
82     if (argc < kNumMinClArgs)
83     {
84         fprintf (stdout,"Wrong number of command line arguments!\n");
85         return -1;
86     }
87
88     CLShowProgInfo();
89
90     CLReadArgs( pcInputPath,
91                 pcOutputPath,
92                 argc,
93                 argv);
94
95
96
97     // open soundfiles
98     pCInputFile = new CzplAudioFile();
99
100     pCInputFile->OpenReadFile(pcInputPath, kBlockSize);
101
102     if (!pCInputFile->IsFileOpen())
103     {
104         std::cout << "Input file could not be opened!" << std::endl;
105         delete pCInputFile;
```

```
106         return -1;
107     }
108     else if (pCInputFile->GetNumOfChannels() > kNumMaxChannels)
109     {
110         std::cout << "Invalid input channel count!" << std::endl;
111         pCInputFile->CloseFile();
112         delete pCInputFile;
113         return -1;
114     }
115
116     if (pcOutputPath)
117     {
118         FOutputFile.open(pcOutputPath);
119         if (!FOutputFile.is_open ())
120         {
121             std::cout << "Output file could not be opened!" << std::endl;
122             pCInputFile->CloseFile ();
123             delete pCInputFile;
124             return -1;
125         }
126     }
127
128
129     for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
130         apfSplitInputData[i]     = new float [kBlockSize];
131
132     // create instance
133     CMeteringIf::CreateInstance (   pCMeteringHandle,
134                                     static_cast<int>(pCInputFile->GetSampleRate () + .1F),
135                                     pCInputFile->GetNumOfChannels (),
136                                     CMeteringIf::kPpmTp);
137
138     // set processing parameters
139     pCMeteringHandle->SetAddDenormalNoise(false);
140     pCMeteringHandle->SetOutputInDB (false);
141     //pCMeteringHandle->SetParam (CMeteringIf::kParamTime1InMs, 5.0F);
142
143     // now we can begin to process!
144     while(bReadNextFrame)
145     {
146         int iNumFrames2Read = 441;
147         // read new frames
148         iNumFramesRead = pCInputFile->Read(apfSplitInputData, iNumFrames2Read);
149
150         if(iNumFramesRead<iNumFrames2Read)
151         {
152             for (int ch=0; ch<pCInputFile->GetNumOfChannels(); ch++)
153                 memset (&apfSplitInputData[ch][iNumFramesRead],
154                     0,
155                     (iNumFrames2Read-iNumFramesRead)*pCInputFile->GetNumOfChannels ()*sizeof(float));
156             bReadNextFrame = false;
157         }
158
159         CLShowProcessTime(iCurrentFrame++, pCInputFile->GetSampleRate (),iNumFrames2Read);
160
161         // process the current block, and sum up the processing times for each block
162         clStartTime = clock();
163         pCMeteringHandle->Process (apfSplitInputData, apfSplitInputData, iNumFramesRead);
164         clTotalTime += (clock() - clStartTime);
165
166         // write output data
167         if (pcOutputPath)
168         {
169             for(i = 0; i < iNumFramesRead; i++)
170             {
171                 for(int j = 0; j < pCInputFile->GetNumOfChannels (); j++)
172                     FOutputFile << afOutputData[i*pCInputFile->GetNumOfChannels () + j] << endl;
173             }
174         }
175     }
176
177     std::cout << std::endl << "Input file processed!" << std::endl << std::endl;
178
179     CLShowProcessedTime(clTotalTime);
```

```
180
181     for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
182         delete [] apfSplitInputData[i];
183
184     // destroy instance
185     CMeteringIf::DestroyInstance (pCMeteringHandle);
186
187     // close files
188     pCInputFile->CloseFile ();
189     delete pCInputFile;
190     FOutputFile.close ();
191
192
193     return 0;
194
195 }
```

5.5 MeteringIf.h File Reference

interface of the CMeteringIf class.

Classes

- class CMeteringIf

5.5.1 Detailed Description

interface of the CMeteringIf class.
: