Metering SDK 2.1.6

by zplane.development

September 22, 2016

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

1.2.1.1 Documentation

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

1.2.1.2 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.

1.2.1.3 Source Files

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

1.2.1.4 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**.

1.2.1.5 Resource Files

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

1.2.1.6 Library Files

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

1.2.1.7 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**.

1.2.1.8 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 TruePeak 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, 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 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

  Calculate the integrated loudness of the complete signal.
- 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

4.1.2.1 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

kNumChannelTypes

Definition at line 48 of file LoudnessIf.h.

```
{
    kFront,
    kSurround,
    kLFE,

    kNumChannelTypes
};
```

4.1.2.2 enum CLoudnessIf::Version_t

Enumerator:

kMajor

kMinor

kPatch

kBuild

kNumVersionInts

Definition at line 230 of file LoudnessIf.h.

```
{
    kMajor,
    kMinor,
    kPatch,
    kBuild,

    kNumVersionInts
};
```

4.1.3 Member Function Documentation

4.1.3.1 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

pCInstance-Pointer	: handle to the new instance
iSampleRate	: sample rate of the audio signal to be analyzed (in Hz)
iNumberOf-Channels	: number of channels in the audio signal to be analyzed
ulMax-Programme-LengthIn-Frames	: the length of the audio signal
iMaxBlock-Size	: the largest block size to expect

Returns

int : 0 when no error

Referenced by main().

4.1.3.2 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

pCInstance-Pointer	: handle to the instance to be destroyed

Returns

int : 0 when no error

Referenced by main().

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

Referenced by CLShowProgInfo().

4.1.3.4 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().

4.1.3.5 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)

4.1.3.6 virtual float CLoudnessIf::GetMaximumMomentaryLoudness ( ) 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)

4.1.3.7 virtual float CLoudnessIf::GetMaximumShortTermLoudness ( ) 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)

4.1.3.8 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

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

Calculate the integrated loudness 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. 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().

4.1.3.10 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

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

Referenced by CLShowProgInfo().

4.1.3.12 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

4.1.3.13 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().

4.1.3.14 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

ppfSample-Data	: audio buffer of dimension [channels][frames]
iNumFrames	: number of frames in this block

Referenced by main().

4.1.3.15 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 GetMomentary-Loudness() and GetShortTermLoudness().

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

Completely reset this instance to its initial state.

4.1.3.17 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

aeChannel-Types	: an array with a ChannelType_t enum for each channel of the audio signal

4.1.3.18 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

4.1.4.1 const float CLoudnessIf::kfMinLUFS [static]

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

Definition at line 61 of file LoudnessIf.h.

4.1.4.2 const float CLoudnessIf::kfUndefinedLUFS [static]

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

Definition at line 64 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 *pfInputBufferInterleaved, float *pfOutputBufferInterleaved, int iNumberOfFrames)=0
- virtual int ApplyFilterFunction (float *pfInputBufferInterleaved, float *pfOutputBufferInterleaved, 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

4.2.2.1 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.

```
{
    kLastBlock,
    kOverall,

    kNumMaxima
};
```

4.2.2.2 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.

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.

```
{
    kPpm,
    kPpmType1,
    kPpmType2,
    kRms,
    kVu,
    kLeqa,
    kBs1770,
    kPpmTp,
    kPpmTpHR,
    // obsolete: for compatibility with older versions
    kTruePeak = kPpmTpHR,

    kNumMeterTypes
};
```

4.2.2.3 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.

```
{
    kParamTime1InMs,
    kParamTime2InMs,

    kNumOfParameters
};
```

4.2.2.4 enum CMeteringIf::Version_t

Enumerator:

kMajor

kMinor

kPatch

kBuild

kNumVersionInts

Definition at line 187 of file MeteringIf.h.

```
{
    kMajor,
    kMinor,
    kPatch,
    kBuild,

    kNumVersionInts
};
```

4.2.3 Constructor & Destructor Documentation

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

Definition at line 201 of file MeteringIf.h.

```
{};
```

4.2.4 Member Function Documentation

4.2.4.1 virtual int CMeteringIf::ApplyFilterFunction (float * *pfInputBufferInterleaved,* float * *pfOutputBufferInterleaved,* 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

**pfInputBufferInterleaved*	: pointer to interleaved audio data input in floating point format
**pfOutputBufferInterleaved*	: pointer to interleaved 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

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

creates a new instance of Metering

Parameters

pCInstance-Pointer	: handle to the new instance
iSampleRate	: sample rate of audio signal to be analyzed (in Hz)
iNumberOf-Channels	: 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().

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

destroys an instance of Metering

Parameters

pCInstance-Pointer	: handle to the instance to be destroyed

Returns

static int : 0 when no error

Referenced by main().

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

returns true if denomral noise is added internally

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

Referenced by CLShowProgInfo().

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

get the maximum

Parameters

eMaxType	: get the block or the overall maximum
pfMaxIn-Channel	: maximum per channel

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

returns true if the output is in "amplitude"

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

returns the parameters of the metering type

Parameters

eParam-Index	: index of parameter

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

Referenced by CLShowProgInfo().

4.2.4.10 virtual int CMeteringIf::Process (float * *pfInputBufferInterleaved,* float * *pfOutputBufferInterleaved,* int *iNumberOfFrames*) [pure virtual]

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

Parameters

**pfInput-Buffer-Interleaved*	: pointer to interleaved audio data input in floating point format
**pfOutput-Buffer-Interleaved*	: pointer to interleaved audio data output in floating point format may be the same as input buffer for inplace processing)
iNumberOf-Frames	: 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

Referenced by main().

4.2.4.11 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

4.2.4.12 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().

4.2.4.13 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().

4.2.4.14 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

Definition in file docugen.txt.

5.2 LoudnessCLMain.cpp File Reference

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

Defines

- #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 Define Documentation

5.2.1.1 #define kBlockSize (8192)

Definition at line 33 of file LoudnessCLMain.cpp.

Referenced by main().

5.2.1.2 #define kNumMaxChannels (8)

Definition at line 36 of file LoudnessCLMain.cpp.

Referenced by main().

5.2.1.3 #define kNumMinClArgs (2)

Definition at line 35 of file LoudnessCLMain.cpp.

Referenced by main().

5.2.2 Function Documentation

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

Definition at line 182 of file LoudnessCLMain.cpp.

Referenced by main().

```
{
    pcInputPath     = argv[1];
    return;
}
```

5.2.2.2 static void CLShowProcessedTime (clock_t *clTime*) [static]

Definition at line 193 of file LoudnessCLMain.cpp.

Referenced by main().

```
{
    fprintf(stdout, "\nTime elapsed:\t%2.2f sec\n", (float)(clTime) /
      CLOCKS_PER_SEC);

    return;
}
```

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

Definition at line 187 of file LoudnessCLMain.cpp.

Referenced by main().

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

5.2.2.4 static void CLShowProgInfo ( ) [static]

Definition at line 167 of file LoudnessCLMain.cpp.

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

Referenced by main().

```
{
    std::cout << "zplane.development Loudness Metering App" << std::endl;
    std::cout << "(c) 2011 by zplane" << std::endl;

    std::cout << "V"
```

```
        << CLoudnessIf::GetVersion (CLoudnessIf::kMajor) << "."
        << CLoudnessIf::GetVersion (CLoudnessIf::kMinor) << "."
        << CLoudnessIf::GetVersion (CLoudnessIf::kPatch) << " build: "
        << CLoudnessIf::GetVersion (CLoudnessIf::kBuild) << ", date: "
        << CLoudnessIf::GetBuildDate () << std::endl;

    return;
}
```

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

Definition at line 46 of file LoudnessCLMain.cpp.

References CLReadArgs(), CLShowProcessedTime(), CLShowProcessTime(), CLShowProgInfo(), CLoudnessIf::CreateInstance(), CLoudnessIf::DestroyInstance(), CLoudnessIf::GetLoudnessRange(), CLoudnessIf::GetProgrammeLoudness(), kBlockSize, kNumMaxChannels, kNumMinClArgs, and CLoudnessIf::Process().

```
{

    char                *pcInputPath    = 0;

    int                 i,
                        iCurrentFrame   = 0;

    CzplAudioFile       *pCInputFile    = 0;

    clock_t             clTotalTime     = 0;
    clock_t             clStartTime     = 0;

    float               *apfInputData[kNumMaxChannels];

    CLoudnessIf         *pCLoudnessHandle       = 0;        // instance handle

#if (!defined(WITHOUT_MEMORY_CHECK) && defined(_DEBUG) && defined (WIN32))
    // set memory checking flags
    int iDbgFlag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
    iDbgFlag       |= _CRTDBG_CHECK_ALWAYS_DF;
    iDbgFlag       |= _CRTDBG_LEAK_CHECK_DF;
    _CrtSetDbgFlag( iDbgFlag );
#endif

#if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined (WIN32))
    // enable check for exceptions (don't forget to enable stop in MSVC!)
    _controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW | _EM_UNDERFLOW |
      _EM_DENORMAL), _MCW_EM) ;
#endif // #ifndef WITHOUT_EXCEPTIONS

    //check for correct number of command line arguments
    if (argc < kNumMinClArgs)
    {
        std::cout << "Wrong number of command line arguments!" << std::endl
                  << "Usage: LoudnessTestCL inputfile" << std::endl;
        return -1;
    }

    CLShowProgInfo();

    CLReadArgs( pcInputPath,
                argc,
```

```
            argv);

    // open soundfile
    pCInputFile = new CzplAudioFile();
    pCInputFile->OpenReadFile(pcInputPath, kBlockSize);

    if (!pCInputFile->IsFileOpen())
    {
        std::cout << "Input file could not be opened!" << std::endl;
        delete pCInputFile;
        return -1;
    }
    else if (pCInputFile->GetNumOfChannels() > kNumMaxChannels)
    {
        std::cout << "Invalid input channel count!" << std::endl;
        pCInputFile->CloseFile();
        delete pCInputFile;
        return -1;
    }

    // allocate input sample buffers
    for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
    {
        apfInputData[i] = new float [kBlockSize];
    }

    // get properties of the input file
    int iSampleRate  = static_cast<int> (pCInputFile->GetSampleRate ());
    int iNumChannels = pCInputFile->GetNumOfChannels();
    int iNumFrames   = pCInputFile->GetFileSize();

    // create loudness metering instance
    CLoudnessIf::CreateInstance (pCLoudnessHandle, iSampleRate, iNumChannels,
      iNumFrames, kBlockSize);

    // now we can begin to process!
    while(iCurrentFrame < iNumFrames)
    {
        // read next block from input file
        int iNumFramesRead = pCInputFile->Read(apfInputData, kBlockSize);

        // process next block with the loudness meter, and sum up the
       processing times for all blocks
        clStartTime = clock();
        pCLoudnessHandle->Process(apfInputData, iNumFramesRead);
        clTotalTime += (clock() - clStartTime);

        // increase current frame pointer and show progress
        iCurrentFrame += iNumFramesRead;
        CLShowProcessTime(iCurrentFrame, pCInputFile->GetSampleRate ());
    }
    CLShowProcessedTime(clTotalTime);

    // calculate the overall programme loudness and range
    float fProgrammeLoudnessLUFS = pCLoudnessHandle->GetProgrammeLoudness();
    float fLoudnessRangeLU = pCLoudnessHandle->GetLoudnessRange();

    // display results
    std::cout << std::endl;
    std::cout << "Programme loudness: " << (fProgrammeLoudnessLUFS) << " LUFS"
      << std::endl;
    std::cout << "    Loudness range: " << (fLoudnessRangeLU) << " LU" <<
```

```
      std::endl;
    std::cout << std::endl;

    // free input sample buffers
    for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
    {
        delete [] apfInputData[i];
    }

    // destroy instance
    CLoudnessIf::DestroyInstance(pCLoudnessHandle);

    // close files
    pCInputFile->CloseFile ();
    delete pCInputFile;

    return 0;

}
```

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. :

Definition in file LoudnessIf.h.

5.4 MeteringCLMain.cpp File Reference

```
#include <time.h> #include <cstring> #include <fstream> ×
#include <iostream> #include "zplAudioFile.h" #include "Metering-
If.h" #include <stdlib.h>
```

Defines

- #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 iBlocksizeFile)
- static void CLShowProcessedTime (clock_t clTime)
- static void ClUtilSplit2Interleaved (float ∗pfInterleavedBuffer, float ∗∗ppfSingleChannels, int iNumChannels, int iNumFrames)
- int main (int argc, char ∗argv[ ])

5.4.1 Define Documentation

5.4.1.1 #define kBlockSize (8192)

Definition at line 33 of file MeteringCLMain.cpp.

Referenced by main().

5.4.1.2 #define kNumMaxChannels (8)

Definition at line 36 of file MeteringCLMain.cpp.

Referenced by main().

5.4.1.3 #define kNumMinClArgs (2)

Definition at line 35 of file MeteringCLMain.cpp.

Referenced by main().

5.4.2 Function Documentation

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

Definition at line 216 of file MeteringCLMain.cpp.

Referenced by main().

```
{
    pcInputPath     = argv[1];
    pcOutputPath    = argv[2];
    return;
}
```

5.4.2.2 static void CLShowProcessedTime (clock_t *clTime*) [static]

Definition at line 228 of file MeteringCLMain.cpp.

Referenced by main().

```
{
    fprintf(stdout, "\nTime elapsed:\t%2.2f sec\n", (float)(clTime) /
      CLOCKS_PER_SEC);

    return;
}
```

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

Definition at line 222 of file MeteringCLMain.cpp.

Referenced by main().

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

5.4.2.4 static void CLShowProgInfo ( ) [static]

Definition at line 201 of file MeteringCLMain.cpp.

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

Referenced by main().

```
{
    cout << "zplane.development Metering App" << endl;
    cout << "(c) 2000-2011 by zplane" << endl;
    cout    << "V"
        << CMeteringIf::GetVersion (CMeteringIf::kMajor) << "."
        << CMeteringIf::GetVersion (CMeteringIf::kMinor) << "."
        << CMeteringIf::GetVersion (CMeteringIf::kPatch) << " build: "
        << CMeteringIf::GetVersion (CMeteringIf::kBuild) << ", date: "
        << CMeteringIf::GetBuildDate () << endl;
    cout << "Press Escape to cancel..." << endl << endl;

    return;
}
```

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

Definition at line 44 of file MeteringCLMain.cpp.

Referenced by main().

```
{
    int iIlIdx = 0;
    for (int i = 0; i < iNumFrames; i++)
        for (int c = 0; c < iNumChannels; c++,iIlIdx++)
            pfInterleavedBuffer[iIlIdx] = ppfSingleChannels[c][i];
    return;
}
```

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

Definition at line 55 of file MeteringCLMain.cpp.

References CLReadArgs(), CLShowProcessedTime(), CLShowProcessTime(), CLShowProgInfo(), ClUtilSplit2Interleaved(), CMeteringIf::CreateInstance(), CMeteringIf::DestroyInstance(), kBlockSize, kNumMaxChannels, kNumMinClArgs, CMeteringIf::kPpmTp, CMeteringIf::Process(), CMeteringIf::SetAddDenormalNoise(), and CMeteringIf::SetOutputInDB().

```
{

    char                *pcInputPath    = 0,
                        *pcOutputPath   = 0;

    int                 i,
                        iNumFramesRead,
                        iCurrentFrame   = 0;

    bool                bReadNextFrame  = true;

    CzplAudioFile       *pCInputFile    = 0;

    std::ofstream       FOutputFile;

    clock_t             clTotalTime = 0;
    clock_t             clStartTime     = 0;

    float               *apfSplitInputData[kNumMaxChannels],
                        afInputData[kNumMaxChannels*kBlockSize],
                        afOutputData[kNumMaxChannels*kBlockSize];

    CMeteringIf         *pCMeteringHandle       = 0;        // instance handles


    //check for correct number of command line arguments
    if (argc < kNumMinClArgs)
    {
        fprintf (stdout,"Wrong number of command line arguments!\n");
        return -1;
    }

    CLShowProgInfo();

    CLReadArgs( pcInputPath,
                pcOutputPath,
                argc,
                argv);



    // open soundfiles
    pCInputFile = new CzplAudioFile();

    pCInputFile->OpenReadFile(pcInputPath, kBlockSize);

    if (!pCInputFile->IsFileOpen())
    {
        std::cout << "Input file could not be opened!" << std::endl;
        delete pCInputFile;
        return -1;
```

```
}
else if (pCInputFile->GetNumOfChannels() > kNumMaxChannels)
{
    std::cout << "Invalid input channel count!" << std::endl;
    pCInputFile->CloseFile();
    delete pCInputFile;
    return -1;
}

if (pcOutputPath)
{
    FOutputFile.open(pcOutputPath);
    if (!FOutputFile.is_open ())
    {
        std::cout << "Output file could not be opened!" << std::endl;
        pCInputFile->CloseFile ();
        delete pCInputFile;
        return -1;
    }
}


for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
    apfSplitInputData[i]     = new float [kBlockSize];

// create instance
CMeteringIf::CreateInstance (   pCMeteringHandle,
                                static_cast<int>(pCInputFile->GetSampleRate
   () + .1F),
                                pCInputFile->GetNumOfChannels (),
                                CMeteringIf::kPpmTp);

// set processing parameters
pCMeteringHandle->SetAddDenormalNoise(false);
pCMeteringHandle->SetOutputInDB (false);
//pCMeteringHandle->SetParam (CMeteringIf::kParamTime1InMs, 5.0F);

// now we can begin to process!
while(bReadNextFrame)
{
    int iNumFrames2Read = 441;
    // read new frames
    iNumFramesRead = pCInputFile->Read(apfSplitInputData, iNumFrames2Read);

    if(iNumFramesRead<iNumFrames2Read)
    {
        for (int ch=0; ch<pCInputFile->GetNumOfChannels(); ch++)
            memset (&apfSplitInputData[ch][iNumFramesRead],
                0,
                (iNumFrames2Read-iNumFramesRead)*pCInputFile->
  GetNumOfChannels ()*sizeof(float));
        bReadNextFrame = false;
    }

    CLShowProcessTime(iCurrentFrame++, pCInputFile->GetSampleRate (),
  iNumFrames2Read);

    // convert data format
    ClUtilSplit2Interleaved (afInputData, apfSplitInputData, pCInputFile->
  GetNumOfChannels (), iNumFrames2Read);

    // process the current block, and sum up the processing times for each
```

```
        block
         clStartTime = clock();
         pCMeteringHandle->Process (afInputData, afOutputData, iNumFramesRead);
         clTotalTime += (clock() - clStartTime);

         // write output data
         if (pcOutputPath)
         {
             for(i = 0; i < iNumFramesRead; i++)
             {
                 for(int j = 0; j < pCInputFile->GetNumOfChannels (); j++)
                     FOutputFile << afOutputData[i*pCInputFile->GetNumOfChannels
    () + j] << endl;
             }
         }
    }

    std::cout << std::endl << "Input file processed!" << std::endl << std::endl
      ;

    CLShowProcessedTime(clTotalTime);

    for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
        delete [] apfSplitInputData[i];

    // destroy instance
    CMeteringIf::DestroyInstance (pCMeteringHandle);

    // close files
    pCInputFile->CloseFile ();
    delete pCInputFile;
    FOutputFile.close ();


    return 0;

}
```

5.5 MeteringIf.h File Reference

interface of the CMeteringIf class.

Classes

- class CMeteringIf

5.5.1 Detailed Description

interface of the CMeteringIf class. :

Definition in file MeteringIf.h.