



Metering SDK 2.0.0

by zplane.development

(c) 2011 zplane.development GmbH & Co. KG

August 2, 2011

Contents

1	Metering Documentation	2
1.1	Introduction	2
1.2	SDK Content	2
1.2.1	Folder Structure	2
1.2.2	Project Structure	3
1.2.3	Project Configurations	3
1.3	API Documentation	3
1.3.1	Naming Conventions	3
1.4	CMeteringIf - SDK-interface for PPM, VU, RMS and TruePeak metering	4
1.4.1	Instance Control Functions	4
1.4.2	Parameter Functions	4
1.4.3	Process Function	4
1.5	CLoudnessIf - SDK-interface for EBU R128 loudness metering	5
1.5.1	Instance Control Functions	5
1.5.2	Configuration Functions	5
1.5.3	Process Function	6
1.5.4	Result Functions	6
1.5.5	Usage Example	6
1.6	Third Party Libraries	7
1.7	Coding Style minimal overview	7
1.8	Support	7
2	Class Index	8
2.1	Class List	8
3	File Index	8
3.1	File List	8
4	Class Documentation	8
4.1	CLoudnessIf Class Reference	8
4.1.1	Detailed Description	10
4.1.2	Member Enumeration Documentation	10
4.1.3	Member Function Documentation	11
4.1.4	Member Data Documentation	15
4.2	CMeteringIf Class Reference	15
4.2.1	Detailed Description	16
4.2.2	Member Enumeration Documentation	16
4.2.3	Constructor & Destructor Documentation	18
4.2.4	Member Function Documentation	18
5	File Documentation	21
5.1	docugen.txt File Reference	21
5.1.1	Detailed Description	21
5.2	LoudnessIf.h File Reference	21
5.2.1	Detailed Description	22
5.3	LoudnessTestCLMain.cpp File Reference	22
5.3.1	Detailed Description	23
5.3.2	Define Documentation	23

5.3.3	Function Documentation	24
5.4	MeteringIf.h File Reference	28
5.4.1	Detailed Description	29
5.5	MeteringTestCLMain.cpp File Reference	29
5.5.1	Define Documentation	30
5.5.2	Function Documentation	31

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 4

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 number 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 parameter 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.

1.5 CLOUDNESSIF - SDK-INTERFACE FOR EBU R128 LOUDNESS METERING

The details of the loudness metering functionalities implemented in this interface are best explained on the EBU website:

<http://tech.ebu.ch/loudness>

"On the way to Loudness Nirvana" is a good short introduction to the key concepts:

http://tech.ebu.ch/webdav/site/tech/shared/techreview/trev_-2010-Q3_loudness_Camerer.pdf

The "Production Guidelines" contain all the details about the implementation and how to use the results for loudness normalization:

<http://tech.ebu.ch/webdav/site/tech/shared/tech/tech3343.pdf>

1.5.1 Instance Control Functions

The following functions have to be called when using the loudness metering interface:

- **CLOUDNESSIF::CREATEINSTANCE (CLOUDNESSIF* & PCINSTANCEPOINTER, int ISAMPLERATE, int INUMBEROFCHANNELS, unsigned long long ULMAXPROGRAMMELENGTHINFRAMES, int IMAXBLOCKSIZE)**

Create a new instance and starts integrated metering. The handle to the new instance is written to the variable PCINSTANCEPOINTER.

You have to provide all properties of the signal to be measured (sample rate in Hz, number of channels, total length in frames, block size in frames). None of the parameters can be changed afterwards, so you have to destroy and create a new new instance whenever the audio format changes.

The function returns 0 in case of no error.

- **CLOUDNESSIF::DESTROYINSTANCE (CMETERINGIF* & PCINSTANCEPOINTER)**

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. The function returns 0 in case of no error.

1.5.2 Configuration Functions

- **CLOUDNESSIF::SETCHANNELCONFIG (CLOUDNESSIF::CHANNELTYPE_t* aeCHANNELTYPES)**

You need to call this method only when metering multi-channel signals that have surround or LFE channels to specify the channel order of the audio signal. As required by EBU R 128, Surround channels are boosted by 1.5dB and the LFE channel is ignored. You have to create an array of CLOUDNESSIF::CHANNELTYPE_t enums with one item for each of your input channels (in the same order they appear in your input buffers).

1.5.3 Process Function

- **CLOUDNESSIF::PROCESS** (`float** ppfSampleData, int iNumFrames`)

Processes a block of multi-channel audio data, passed in as a two-dimensional array in the form [channels][frames]. The number of frames must not exceed the maximum block size that was specified when calling [CLOUDNESSIF::CREATEINSTANCE](#) (...).

1.5.4 Result Functions

- **CLOUDNESSIF::GETMOMENTARYLOUDNESS ()** and **CLOUDNESSIF::GETSHORTTERMLLOUDNESS ()**

These methods can be called for each input block to get an continuously updating value for the current loudness (in LUFS) of the signal. The difference between those method is the window time (400ms for momentary and 3s for short-term loudness).

- **CLOUDNESSIF::GETPROGRAMMELOUDNESS ()** and **CLOUDNESSIF::GETLOUDNESSRANGE ()**

These methods calculate the average gated loudness (in LUFS) and the loudness range (LU) for the overall audio signal. You should not call these methods from the audio-thread during real-time processing. They are usually meant to be called after the last block was processed to get the final results for a whole file. If you want to measure only a specific part of the signal, use the [CLOUDNESSIF::PAUSE \(\)](#), [CLOUDNESSIF::START \(\)](#) and [CLOUDNESSIF::RESET \(\)](#) methods.

1.5.5 Usage Example

The complete code to calculate the programme loudness and loudness range of an audio file can be found in the example source file [LoudnessTestCLMain.cpp](#). For audio file IO, the open source library `libSndFile` is used in this example.

After opening an file and obtaining the details of its audio format, a pointer to the [CLOUDNESSIF](#) instance has to be declared and instanciated:

```
CLOUDNESSIF          *pCLoudnessHandle          = 0;          // instance handle
// create loudness metering instance
CLOUDNESSIF::CREATEINSTANCE (pCLoudnessHandle, iSampleRate, iNumChannels, iNum
    mFrames, kBlockSize);
```

After instanciation and initialization, we can begin processing succeeding blocks of audio data:

```
// process next block with the loudness meter
pCLoudnessHandle->PROCESS (apfInputData, iNumFramesRead);
```

After processing the last block we can get the overall results:

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

At the end, destroy the instance to free all resources

```
CLOUDNESSIF::DestroyInstance(pCloudnessHandle);
```

1.6 Third Party Libraries

The example program uses the open source library libSndFile (LGPL). Please ensure the license compliance if using this library.

1.7 Coding Style minimal overview

Variable names have a preceding letter indicating their types:

unsigned:	u
pointer:	p
array:	a
class:	C
bool:	b
char:	c
short (int16):	s
int (int32):	i
__int64:	l
float (float32):	f
double (float64):	d
char:	c

For example, a pointer to a buffer of unsigned ints will be named `puiBufferName`.

1.8 Support

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

zplane.development

katzbachstr.21

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:

CLoudnessIf (Implements loudness metering according to the EBU R128 recommendation)	8
CMeteringIf	15

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

LoudnessIf.h (Contains the CLoudnessIf interface for loudness metering)	21
LoudnessTestCLMain.cpp (Usage example for the CLoudnessIf interface)	22
MeteringIf.h (Interface of the CMeteringIf class)	28
MeteringTestCLMain.cpp	29

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 **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 216 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

<i>pCINSTANCEPOINTER</i>	: handle to the new instance
<i>ISAMPLERATE</i>	: sample rate of the audio signal to be analyzed (in Hz)
<i>INUMBEROFCHANNELS</i>	: number of channels in the audio signal to be analyzed
<i>ULMAXPROGRAMMELENGTHINFRAMES</i>	: the length of the audio signal
<i>IMAXBLOCKSIZE</i>	: 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

<i>pInstance-Pointer</i>	: 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

<i>fLow</i>	: reference to return the lower bounds of the range in LUFS
<i>fHigh</i>	: 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 CCloudnessIf::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

4.1.3.7 `virtual float CCloudnessIf::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 [CCloudnessIf::kfUndefinedLUFS](#) to signal that there is no valid result available. It will return [CCloudnessIf::kfMinLUFS](#) if all input was below the absolute gating threshold.

Returns

the programme loudness in LUFS (or [CCloudnessIf::kfUndefinedLUFS](#) / [CCloudnessIf::kfMinLUFS](#))

Referenced by [main\(\)](#).

4.1.3.8 `virtual float CCloudnessIf::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

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

Referenced by [CLShowProgInfo\(\)](#).

4.1.3.10 `virtual bool CCloudnessIf::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

4.1.3.11 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()`.

4.1.3.12 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

<i>ppfSampleData</i>	: audio buffer of dimension [channels][frames]
<i>iNumFrames</i>	: number of frames in this block

Referenced by `main()`.

4.1.3.13 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()` and `GetLoudnessRange()` and start a new measurement beginning with the next incoming audio block. This method does not affect the results of `GetMomentaryLoudness()` and `GetShortTermLoudness()`.

4.1.3.14 virtual void CLOUDNESSIF::ResetInstance () [pure virtual]

Completely reset this instance to its initial state.

4.1.3.15 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

<i>aeChannel-Types</i>	: an array with a <code>ChannelType_t</code> enum for each channel of the audio signal
------------------------	--

4.1.3.16 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`, `kRms`, `kVu`, `kLeqa`,
`kBs1770`, `kTruePeak`, `kNumMeterTypes` }
- enum `Parameters_t` { `kParamTime1InMs`, `kParamTime2InMs`, `kNumOfParameters` }

- 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 [Reset](#) ()=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::MeterTypes_t](#)

defines the available metering types

Enumerator:

kPpm PPM style meter.

kRms RMS measurement.

kVu VU style meter.

kLeqa RMS based A-weighted loudness measurement.
kBs1770 ITU-R BS.1770 based loudness measurement.
kTruePeak "true-peak" measurement as described in ITU-R BS.1770 Annex B
 */
kNumMeterTypes

Definition at line 44 of file MeteringIf.h.

```
{
    kPpm,
    kRms,
    kVu,
    kLeqa,
    kBs1770,
    kTruePeak,

    kNumMeterTypes
};
```

4.2.2.2 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 57 of file MeteringIf.h.

```
{
    kParamTime1InMs,
    kParamTime2InMs,

    kNumOfParameters
};
```

4.2.2.3 enum CMeteringIf::Version_t

Enumerator:

kMajor
kMinor
kPatch
kBuild
kNumVersionInts

Definition at line 166 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 180 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

<i>*pfInputBuffer</i>	: pointer to interleaved audio data input in floating point format
<i>*pfOutputBuffer</i>	: pointer to interleaved audio data output in floating point format may be the same as input buffer for inplace processing)
<i>iNumberOfFrames</i>	: 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

<i>pCInstancePointer</i>	: handle to the new instance
<i>iSampleRate</i>	: sample rate of audio signal to be analyzed (in Hz)

<i>iNumberOfChannels</i>	: number of channels of audio signal to be analyzed
<i>eType</i>	: type of meter that is supposed to be used values: <code>_PPM</code> : peak program meter (works with default values, not for sure with longer attack times) <code>_RMS</code> : root mean square <code>_VU</code> : volume unit <code>_LEQA</code> : 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

<i>pCInstancePointer</i>	: 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 denormal noise is added internally

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

Referenced by CLShowProgInfo().

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

returns true if the output is in "amplitude"

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

returns the parameters of the metering type

Parameters

<i>eParamIndex</i>	: index of parameter
--------------------	----------------------

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

Referenced by CLShowProgInfo().

4.2.4.9 `virtual int CMeteringIf::Process (float * pInputBufferInterleaved, float * pOutputBufferInterleaved, int iNumberOfFrames)` `[pure virtual]`

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

Parameters

<code>*pInputBuffer</code>	: pointer to interleaved audio data input in floating point format
<code>*pOutputBuffer</code>	: pointer to interleaved audio data output in floating point format may be the same as input buffer for inplace processing)
<code>iNumberOfFrames</code>	: 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().

4.2.4.10 `virtual int CMeteringIf::Reset ()` `[pure virtual]`

reset internal buffers

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

allows to add small amount of noise to avoid denormals

Parameters

<code>bAddNoise</code>	: true if noise will be added
------------------------	-------------------------------

Referenced by main().

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

allows to select if output is in decibels or "amplitude"; $\text{dB} = 20 \cdot \log_{10}(\text{amplitude})$ for all scales

Parameters

<code>bOutputInDB</code>	: true if output is in dB
--------------------------	---------------------------

4.2.4.13 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

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

Referenced by main().

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

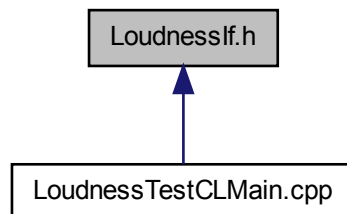
source documentation main file

Definition in file [docugen.txt](#).

5.2 LoudnessIf.h File Reference

Contains the [CLoudnessIf](#) interface for loudness metering.

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



Classes

- class [CLoudnessIf](#)

Implements loudness metering according to the EBU R128 recommendation.

5.2.1 Detailed Description

Contains the [CLoudnessIf](#) interface for loudness metering. :

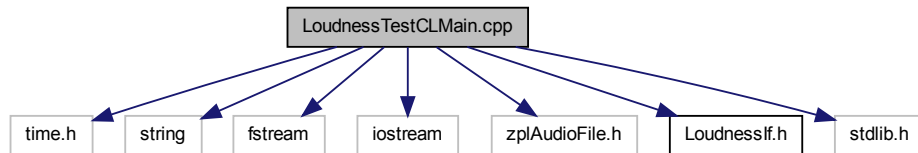
Definition in file [LoudnessIf.h](#).

5.3 LoudnessTestCLMain.cpp File Reference

usage example for the [CLoudnessIf](#) interface

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

Include dependency graph for LoudnessTestCLMain.cpp:



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 cTime)
- int [main](#) (int argc, char *argv[])

5.3.1 Detailed Description

usage example for the [CLoudnessIf](#) interface :

Definition in file [LoudnessTestCLMain.cpp](#).

5.3.2 Define Documentation

5.3.2.1 #define kBlockSize (8192)

Definition at line 33 of file [LoudnessTestCLMain.cpp](#).

Referenced by [main\(\)](#).

5.3.2.2 #define kNumMaxChannels (8)

Definition at line 36 of file [LoudnessTestCLMain.cpp](#).

Referenced by [main\(\)](#).

5.3.2.3 #define kNumMinCIArgs (2)

Definition at line 35 of file LoudnessTestCLMain.cpp.

Referenced by main().

5.3.3 Function Documentation

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

Definition at line 180 of file LoudnessTestCLMain.cpp.

Referenced by main().

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

5.3.3.2 static void CLShowProcessedTime (clock_t cITime) [static]

Definition at line 191 of file LoudnessTestCLMain.cpp.

Referenced by main().

```
{
    fprintf(stdout, "\nTime elapsed:\t%2.2f sec\n", (float)(cITime) / CLOCKS_PER_
        SEC);
    return;
}
```

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

Definition at line 185 of file LoudnessTestCLMain.cpp.

Referenced by main().

```
{
    fprintf(stderr, "\rProcessed:\t%2.2f seconds of audio data", iCurrentFrame*1.
        0F/fSampleRate);
    return;
}
```

5.3.3.4 static void CLShowProgInfo () [static]

Definition at line 165 of file LoudnessTestCLMain.cpp.

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

Referenced by main().

```

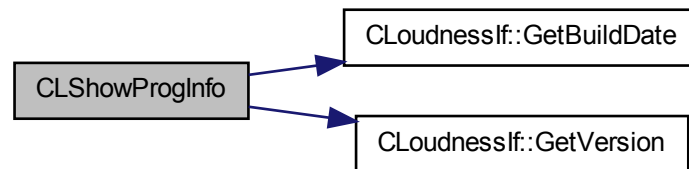
{
    std::cout << "zplane.development Loudness Metering App" << std::endl;
    std::cout << "(c) 2011 by zplane" << std::endl;

    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;
}

```

Here is the call graph for this function:



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

Definition at line 46 of file LoudnessTestCLMain.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           clStartTime     = 0;

    float              *apfInputData[kNumMaxChannels];

    CLoudnessIf       *pCLoudnessHandle = 0;           // instance handle

#ifdef 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 | _E
M_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, iNu
mFrames, kBlockSize);

// now we can begin to process!
clStartTime = clock();

```

```
while(iCurrentFrame < iNumFrames)
{
    // read next block from input file
    int iNumFramesRead = pCInputFile->Read(apfInputData, kBlockSize);

    // process next block with the loudness meter
    pCLoudnessHandle->Process(apfInputData, iNumFramesRead);

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

// 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::en
    dl;
std::cout << std::endl;

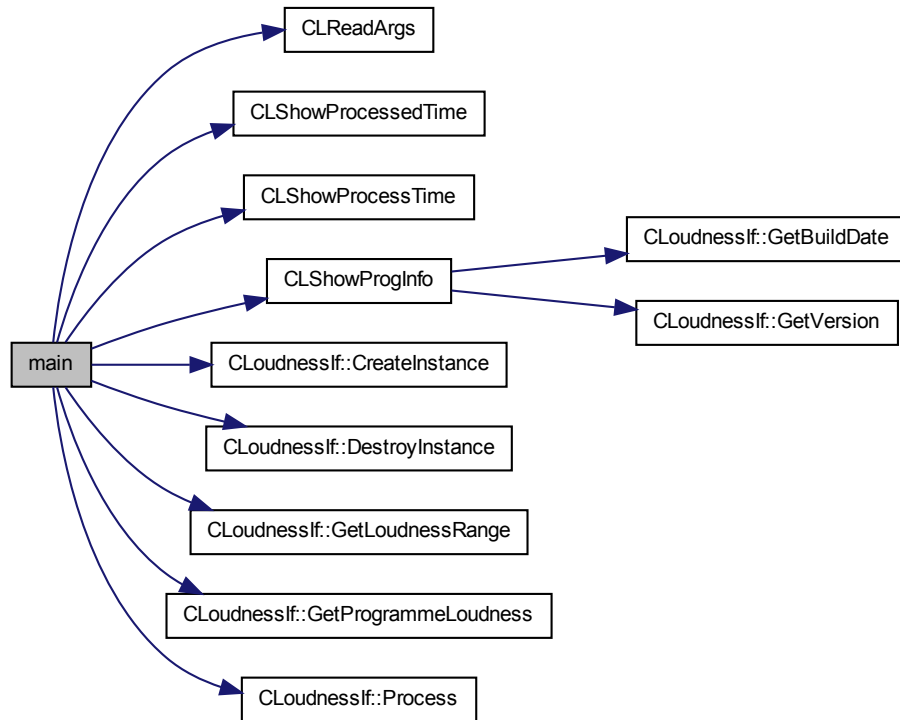
// free input sample buffers
for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
{
    delete [] apfInputData[i];
}

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

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

return 0;
}
```

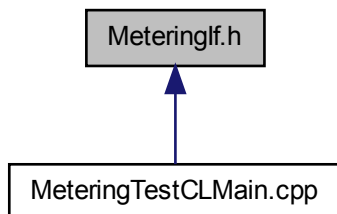
Here is the call graph for this function:



5.4 MeteringIf.h File Reference

interface of the [CMeteringIf](#) class.

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



Classes

- class [CMeteringIf](#)

5.4.1 Detailed Description

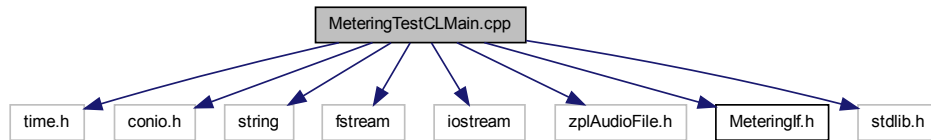
interface of the [CMeteringIf](#) class. :

Definition in file [MeteringIf.h](#).

5.5 MeteringTestCLMain.cpp File Reference

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

Include dependency graph for MeteringTestCLMain.cpp:



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 iBlock-sizeFile)
- static void [CLShowProcessedTime](#) (clock_t cTime)
- static void [CIUtilSplit2Interleaved](#) (float *pfInterleavedBuffer, float **ppfSingleChannels, int iNumChannels, int iNumFrames)
- int [main](#) (int argc, char *argv[])

5.5.1 Define Documentation

5.5.1.1 #define kBlockSize (8192)

Definition at line 34 of file MeteringTestCLMain.cpp.

Referenced by main().

5.5.1.2 #define kNumMaxChannels (8)

Definition at line 37 of file MeteringTestCLMain.cpp.

Referenced by main().

5.5.1.3 #define kNumMinCLArgs (2)

Definition at line 36 of file MeteringTestCLMain.cpp.

Referenced by main().

5.5.2 Function Documentation

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

Definition at line 216 of file MeteringTestCLMain.cpp.

Referenced by main().

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

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

Definition at line 228 of file MeteringTestCLMain.cpp.

Referenced by main().

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

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

Definition at line 222 of file MeteringTestCLMain.cpp.

Referenced by main().

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

5.5.2.4 static void CLShowProgInfo () [static]

Definition at line 201 of file MeteringTestCLMain.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"

```

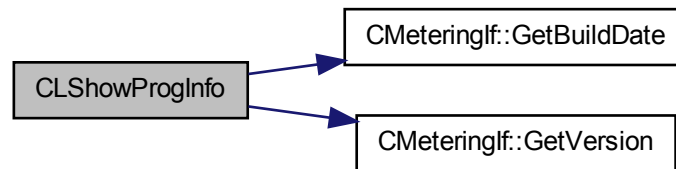
```

    << 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;
}

```

Here is the call graph for this function:



5.5.2.5 static void CIUtilSplit2Interleaved (float * *pfInterleavedBuffer*, float ** *ppfSingleChannels*, int *iNumChannels*, int *iNumFrames*) [static]

Definition at line 45 of file MeteringTestCLMain.cpp.

Referenced by main().

```

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

```

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

Definition at line 56 of file MeteringTestCLMain.cpp.

References CLReadArgs(), CLShowProcessedTime(), CLShowProcessTime(), CLShowProgInfo(), CIUtilSplit2Interleaved(), CMeteringIf::CreateInstance(), CMeteringIf::DestroyInstance(), kBlockSize, kNumMaxChannels, kNumMinClArgs, CMeteringIf::kParamTime1InMs, CMeteringIf::kPpm, CMeteringIf::Process(), CMeteringIf::SetAddDenormalNoise(), and CMeteringIf::SetParam().

```

{

```

```
char          *pcInputPath   = 0,
              *pcOutputPath = 0;

int           i,
              iNumFramesRead,
              iCurrentFrame  = 0;

bool          bReadNextFrame = true;

CzplAudioFile *pCInputFile   = 0;

std::ofstream FOutputFile;

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::kPpm);

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

clStartTime = clock();

// now we can begin to process!
while(bReadNextFrame)
{
    int iNumFrames2Read = 1024;
    // 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->Get
    NumOfChannels (), iNumFrames2Read);

    // process meter the current block
    pCMeteringHandle->Process (afInputData, afOutputData, iNumFramesRead);

    //if (pcOutputPath)
    //{
    //    for(i = 0; i < iNumFramesRead; i++)
    //        {
    //            for(int j = 0; j < pCInputFile->GetNumOfChannels (); j++)

    //                fprintf (pFOutputFile,"%1.10e\t", afOutputData[i*sfInputInfo.ch
    annels+j]);
    //            fprintf (pFOutputFile, "\n");
    //        }
    //}
}

```

```
}

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

CLShowProcessedTime(clock() - clStartTime);

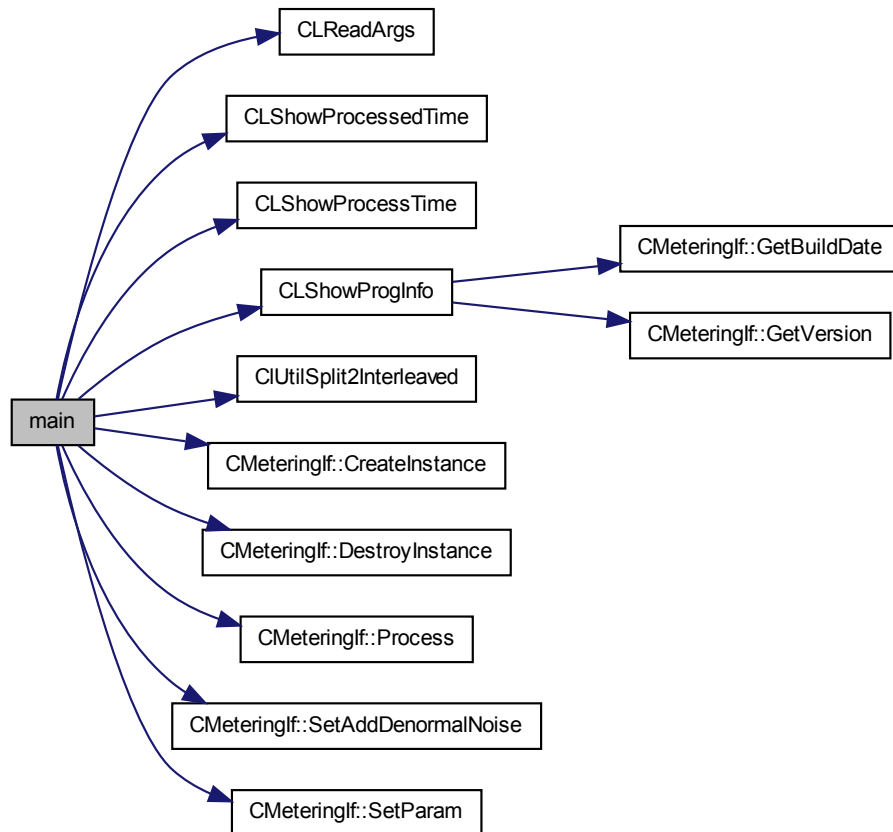
for (i = 0; i < pCInputFile->GetNumOfChannels (); i++)
    delete [] apfSplitInputData[i];

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

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

return 0;
}
```

Here is the call graph for this function:



Index

- ~CMeteringIf
 - CMeteringIf, 18
- ApplyFilterFunction
 - CMeteringIf, 18
- ChannelType_t
 - CLoudnessIf, 10
- CLoudnessIf, 8
 - ChannelType_t, 10
 - CreateInstance, 11
 - DestroyInstance, 11
 - GetBuildDate, 12
 - GetLoudnessRange, 12
 - GetMomentaryLoudness, 12
 - GetProgrammeLoudness, 13
 - GetShortTermLoudness, 13
 - GetVersion, 13
 - IsRunning, 13
 - kBuild, 10
 - kfMinLUFS, 15
 - kFront, 10
 - kfUndefinedLUFS, 15
 - kLFE, 10
 - kMajor, 10
 - kMinor, 10
 - kNumChannelTypes, 10
 - kNumVersionInts, 10
 - kPatch, 10
 - kSurround, 10
 - Pause, 14
 - Process, 14
 - Reset, 14
 - ResetInstance, 14
 - SetChannelConfig, 14
 - Start, 15
 - Version_t, 10
- CLReadArgs
 - LoudnessTestCLMain.cpp, 24
 - MeteringTestCLMain.cpp, 31
- CLShowProcessedTime
 - LoudnessTestCLMain.cpp, 24
 - MeteringTestCLMain.cpp, 31
- CLShowProcessTime
 - LoudnessTestCLMain.cpp, 24
 - MeteringTestCLMain.cpp, 31
- CLShowProgInfo
 - LoudnessTestCLMain.cpp, 24
 - MeteringTestCLMain.cpp, 31
- CIUtilSplit2Interleaved
 - MeteringTestCLMain.cpp, 32
- CMeteringIf, 15
 - ~CMeteringIf, 18
 - ApplyFilterFunction, 18
 - CreateInstance, 18
 - DestroyInstance, 19
 - GetAddDenormalNoise, 19
 - GetBuildDate, 19
 - GetOutputInDB, 19
 - GetParam, 19
 - GetVersion, 19
 - kBs1770, 17
 - kBuild, 17
 - kLeqa, 16
 - kMajor, 17
 - kMinor, 17
 - kNumMeterTypes, 17
 - kNumOfParameters, 17
 - kNumVersionInts, 17
 - kParamTime1InMs, 17
 - kParamTime2InMs, 17
 - kPatch, 17
 - kPpm, 16
 - kRms, 16
 - kTruePeak, 17
 - kVu, 16
 - MeterTypes_t, 16
 - Parameters_t, 17
 - Process, 20
 - Reset, 20
 - SetAddDenormalNoise, 20
 - SetOutputInDB, 20
 - SetParam, 20
 - Version_t, 17
- CreateInstance
 - CLoudnessIf, 11
 - CMeteringIf, 18
- DestroyInstance
 - CLoudnessIf, 11
 - CMeteringIf, 19
- docugen.txt, 21
- GetAddDenormalNoise

- CMeteringIf, 19
- GetBuildDate
 - CLoudnessIf, 12
 - CMeteringIf, 19
- GetLoudnessRange
 - CLoudnessIf, 12
- GetMomentaryLoudness
 - CLoudnessIf, 12
- GetOutputInDB
 - CMeteringIf, 19
- GetParam
 - CMeteringIf, 19
- GetProgrammeLoudness
 - CLoudnessIf, 13
- GetShortTermLoudness
 - CLoudnessIf, 13
- GetVersion
 - CLoudnessIf, 13
 - CMeteringIf, 19
- IsRunning
 - CLoudnessIf, 13
- kBlockSize
 - LoudnessTestCLMain.cpp, 23
 - MeteringTestCLMain.cpp, 30
- kBs1770
 - CMeteringIf, 17
- kBuild
 - CLoudnessIf, 10
 - CMeteringIf, 17
- kfMinLUFS
 - CLoudnessIf, 15
- kFront
 - CLoudnessIf, 10
- kfUndefinedLUFS
 - CLoudnessIf, 15
- kLeqa
 - CMeteringIf, 16
- kLFE
 - CLoudnessIf, 10
- kMajor
 - CLoudnessIf, 10
 - CMeteringIf, 17
- kMinor
 - CLoudnessIf, 10
 - CMeteringIf, 17
- kNumChannelTypes
 - CLoudnessIf, 10
- kNumMaxChannels
- LoudnessTestCLMain.cpp, 23
- MeteringTestCLMain.cpp, 30
- kNumMeterTypes
 - CMeteringIf, 17
- kNumMinCIArgs
 - LoudnessTestCLMain.cpp, 23
 - MeteringTestCLMain.cpp, 30
- kNumOfParameters
 - CMeteringIf, 17
- kNumVersionInts
 - CLoudnessIf, 10
 - CMeteringIf, 17
- kParamTime1InMs
 - CMeteringIf, 17
- kParamTime2InMs
 - CMeteringIf, 17
- kPatch
 - CLoudnessIf, 10
 - CMeteringIf, 17
- kPpm
 - CMeteringIf, 16
- kRms
 - CMeteringIf, 16
- kSurround
 - CLoudnessIf, 10
- kTruePeak
 - CMeteringIf, 17
- kVu
 - CMeteringIf, 16
- LoudnessIf.h, 21
- LoudnessTestCLMain.cpp, 22
 - CLReadArgs, 24
 - CLShowProcessedTime, 24
 - CLShowProcessTime, 24
 - CLShowProgInfo, 24
 - kBlockSize, 23
 - kNumMaxChannels, 23
 - kNumMinCIArgs, 23
 - main, 25
- main
 - LoudnessTestCLMain.cpp, 25
 - MeteringTestCLMain.cpp, 32
- MeteringIf.h, 28
- MeteringTestCLMain.cpp, 29
 - CLReadArgs, 31
 - CLShowProcessedTime, 31
 - CLShowProcessTime, 31
 - CLShowProgInfo, 31

- CIUtilSplit2Interleaved, [32](#)
- kBlockSize, [30](#)
- kNumMaxChannels, [30](#)
- kNumMinCIArgs, [30](#)
- main, [32](#)
- MeterTypes_t
 - CMeteringIf, [16](#)
- Parameters_t
 - CMeteringIf, [17](#)
- Pause
 - CLoudnessIf, [14](#)
- Process
 - CLoudnessIf, [14](#)
 - CMeteringIf, [20](#)
- Reset
 - CLoudnessIf, [14](#)
 - CMeteringIf, [20](#)
- ResetInstance
 - CLoudnessIf, [14](#)
- SetAddDenormalNoise
 - CMeteringIf, [20](#)
- SetChannelConfig
 - CLoudnessIf, [14](#)
- SetOutputInDB
 - CMeteringIf, [20](#)
- SetParam
 - CMeteringIf, [20](#)
- Start
 - CLoudnessIf, [15](#)
- Version_t
 - CLoudnessIf, [10](#)
 - CMeteringIf, [17](#)