espace 1.1.0

by zplane.development

April 14, 2021

Contents

1 espace parametric convolution SDK documentation **2**
- 1.1 Introduction . . . 2
  - 1.1.1 Features on a view . . . 2
- 1.2 API Documentation . . . 3
  - 1.2.1 Memory Allocation . . . 3
  - 1.2.2 Naming Conventions . . . 3
  - 1.2.3 Required Functions . . . 3
  - 1.2.4 Parameter Control Functions . . . 4
  - 1.2.5 Usage example . . . 6
  - 1.2.6 Parameter Description . . . 7
- 1.3 Licensing Issues . . . 9
- 1.4 Support . . . 9

2 Class Index **9**
- 2.1 Class List . . . 9

3 File Index **9**
- 3.1 File List . . . 9

4 Class Documentation **9**
- 4.1 _EnvelopePoint_ Struct Reference . . . 9
- 4.2 espaceIf Class Reference . . . 10
  - 4.2.1 Detailed Description . . . 11
  - 4.2.2 Member Enumeration Documentation . . . 11
  - 4.2.3 Constructor & Destructor Documentation . . . 11
  - 4.2.4 Member Function Documentation . . . 11

5 File Documentation **20**
- 5.1 docugen.txt File Reference . . . 20
  - 5.1.1 Detailed Description . . . 20
- 5.2 espaceAPI.h File Reference . . . 20
  - 5.2.1 Detailed Description . . . 20
  - 5.2.2 Typedef Documentation . . . 21
  - 5.2.3 Enumeration Type Documentation . . . 21

1 espace parametric convolution SDK documentation

1.1 Introduction

Using the espace SDK, it is possible to process convolution in realtime and for multi-channel files. Besides the efficient convolution, espace allows to set parameters for the impulse response just like with conventional artificial reverbs - with the difference that no artificial reverberation is created but only the impulse response itself is modified. This combines the high quality of convolution reverbs with the flexibility of artificial reverbs and makes espace a powerful and intuitive reverb in nearly all areas of application. An overview of the features is given below.
The SDK comes with a C++-style API.
Apart from the delivered libraries, no additional libraries are needed. The SDK is delivered with the libraries, the corresponding header files, this documentation and a small example application.
The first part of this document explains the API of the SDK. Then, a small usage example explaining the basic usage of the SDK is given. After some additional notes, some automatically generated documentation follows.

1.1.1 Features on a view

- high performance due to assembler optimizations
- no restrictions on impulse response regarding sample rate, number of channels, length
- automatic analysis of impulse response to achieve a correct crosspoint between early reflections (ER) and late reverberation (LR)
- automatic sample rate conversion if the sample rate of the impulse response does not equal the processing sample rate
- modification of complete impulse response with the following parameters:
  - Pre-Delay
  - N-point envelope
- modification of ER and LR parts of the impulse response individually with the following parameters:
  - Length without changing pitch
  - Pitch without changing length
  - up to ten filters with different settings/characteristics (Lowpass, Highpass, Bandpass, Low-shelving, High-shelving, Peak)
- user adjustable cross-point and cross-fade time between ER and LR part of impulse response
- automatic volume compensation to avoid clipping
- setting of processing block size (latency), wetness, input downmixing to mono
- optional reverse impulse response processing

- possibility to calculate parameter changes in background while processing convolution

1.2 API Documentation

The interface of the espace parametric convolution SDK is given in the the file espaceAPI.h. There are no additional headers required.

1.2.1 Memory Allocation

The espace SDK does not allocate buffers handled by the calling application. The input and output buffers have to be allocated by the calling application. Audio buffers are allocated as double arrays of [channels][SamplesPerChannel].

1.2.2 Naming Conventions

Before using the SDK, an **instance** has to be created. After successful creation, the other API functions can be called.
When talking about frames, the number of audio samples per channel is meant. For mono signals, the number of frames equals the number of samples. For stereo signals, the number of samples is twice the number of frames. So, in general, the number of samples equals the number of frames times the number of channels. If the sample size is 16bit, one sample has a memory usage of 2 byte.

1.2.3 Required Functions

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

- **zERROR espaceIf::CreateInstance (espaceIf∗& pCespace)**
  This function creates a new instance of espace and has to be called before anything can happen. The parameter pCespace is the handle to the new instance and is set in the function.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::DestroyInstance (espaceIf∗& pCespace)**
  This function destroys an instance of espace and has to be called after everything has been done to free the allocated memory etc. The parameter pCespace is the handle to the instance to be destroyed and is set in the function to 0.

- **zERROR espaceIf::Initialize (int iSampleRate, int iNumOfChannels, int i↩ BlockSize)**
  This function initializes an instance of espace and has to be called after the instace has been created. The parameter iSampleRate is the processing sample rate

in Hz, the parameter iNumOfChannels is the number of processing channels, and the parameter iBlockSize is the processing frame size that has to be a power of two.
The function returns 0 when no error occurred.

- **zERROR espaceIf::Process (const float ∗∗ppfInputData, float ∗∗ppfOutput↩ Data, int iNumOfFrames)**
  This function does the actual convolution. The pointer ppfInputData contains in input data in an array [channels][samples], the output data is written to the buffer pointed to in parameter ppfOutputData of the same array format. Note the the parameter ppfOutputData may equal the parameter ppfInputData for inplace processing. The parameter iNumOfFrames is the number of frames to process. Please note that before this function can be called, the impulse response has to be set via the function espaceIf::SetImpulseResponse.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::SetImpulseResponse (const float ∗∗ppfImpulseResponse, int iNumOfIRFrames, int iNumOfIRChannels, int iIRSampleRate)**
  This function is used to set the impulse response. The impulse response data is in an array pointed to by parameter ppfImpulseResponse in the format [channels][samples], the parameter iNumOfIRFrames is the number of frames of the impulse response, the parameters iNumOfIRChannels and iIRSampleRate contain the number of impulse response channels resp. the sample rate of the impulse response. The number of channels must equal the number of process channels if this number is not 1. If the sample rate of the impulse response does not equal the processing sample rate, it will be automatically converted to the processing sample rate.
  A call of this function has to be followed by a call of the function espaceIf::ApplyParameterChanges.
  The function returns 0 when no error occurred.

1.2.4 Parameter Control Functions

The following functions are to set or get the individual parameter values. Most parameters can be controlled via the function espaceIf::SetParameter, however, some parameters are controlled via extra functions.

- **zERROR espaceIf::SetParameter (int iIndex, float fValue)**
  This function allows to set nearly all available parameter (see section Parameter Description).
  The parameter iIndex gives the parameter index, and the parameter fValue the value of the parameter.
  The function returns 0 when no error occurred.

- **float espaceIf::GetParameter (int iIndex)**
  This function is to get the available parameter values (see section Parameter Description).
  The parameter iIndex gives the parameter index, its value is returned.

- **zERROR espaceIf::SetStretchnPitch (float fStretchFactor, float fPitchFactor, ::*IRPart* ePart)**
  This function can be used to set the stretch and/or pitch factor for either the ER or the LR part of the impulse response. The parameter fStretchFactor is the stretch factor in percent (range 0.5...1.5), representing the resulting length of the impulse response, the parameter fPitchFactor is the pitch shifting in percent with

the same range, and the parameter ePart is to define if these changes should take effect on the ER or the LR part of the impulse response.
The function returns 0 when no error occurred.

- **zERROR espaceIf::CreateFilter (void **phFilterHandle, ::*FilterTypes* eType, ::*IRPart* ePart)**
  This function is to create a new filter to be applied to either the ER or the LR part of the impulse response. The parameter phFilterHandle is the handle to the new filter (to be written), the parameter eType defines the filter type (see ::*FilterTypes*), and the parameter ePart is to define if these changes should take effect on the ER or the LR part of the impulse response.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::DestroyFilter (void **phFilterHandle)**
  This function is to destroy a previously created filter. The parameter phFilterHandle is the handle to the filter.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::SetFilterParameter (void *phFilterHandle, float fFilterFreq, float fFilterGain = 0, float fFilterQ = 1)**
  This function sets the parameters of a previously created filter. The parameter phFilterHandle is the handle to the filter and the parameter fFilterFreq specifies, dependent on the selected filter type, the mid or cut-off frequency of the filter in Hz. The parameters fFilterGain and fFilterQ have to be set only in the case that the selected filter type requires them; they correspond to the gain in dB and the Q of the filter.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::GetFilterParameter (void *phFilterHandle, float *pfFilterFreq, float *pfFilterGain = 0, float *pfFilterQ = 0)**
  This function is to return the parameters of a previously created filter. For the parameter description, see function espaceIf::SetfilterParameter, exept that in this case all parameters are pointer that are written by this function.
  The function returns 0 when no error occurred.

- **zERROR espaceIf::SetEnvelope (*EnvelopePoint* **ppstEnvelopePoints, int iNumOfEnvelopePoints)**
  This function allows to set an envelope for the impulse response. The parameter ppstEnvelopePoints contains pointers to structures where the envelope points and their values are defined (see also ::*EnvelopePoint*), and the parameter iNumOfEnvelopePoints specifies the number of envelope points. The number of envelope points must exceed a value of two (beginning and end). The envelope itself is a piecewise linear function that goes through all specified points.
  The function returns 0 when no error occurred.

- **int espaceIf::GetEnvelope (*EnvelopePoint* **ppstEnvelopePoints)**
  This function returns the specified envelope for the impulse response. The parameter ppstEnvelopePoints is exactly the same as described in the function espaceIf::SetEnvelope; its memory has to be allocated by the user beforehand. The function returns the number of envelope points; if the argument ppstEnvelopePoints is 0, only the number of envelope points is returned.

- **zERROR espaceIf::ApplyParameterChanges (PFUNCallback pCallback↩ WhenReady = 0, void ∗pUserData = 0)**
  This function has to be called after every parameter change to let the changes take effect. Currently this function may not be called while processing is done. Thread safety needs to be taken care of. To be notified when the background process has finished (optionally), it is possible to optionally specify a function pointer of type PFUNCallback in the parameter pCallbackWhenReady and some user defined pointer pUserData. then, after the background process has finished, this function is called with the specified user data pointer.
  The function returns 0 when no error occurred.

Other Functions

- **zERROR espaceIf::Reset ()**
  Upon the call of this function, all internal buffers are flushed to 0.
  The function returns 0 when no error occurred.

1.2.5 Usage example

The complete code can be found in the example source file ConvolverTestCLMain.cpp. The example application uses the open source library libSndFile for audio file format parsing.
In the first step, a pointer to the needed espaceIf instance has to be declared. This can be done with the following code snippet:

```
espaceIf            *espaceInstance  = 0;                    // instance
   handle for espace
```

First, a new instance has to be created:

```
// create a new instance of espace and do initialization
if (espaceIf::CreateInstance (espaceInstance) != 0)
{
    fprintf(stdout,"Instance creation failed!\n");
    delete inputFile;
    delete irFile;
    delete outputFile;
    return -1;
}
```

which has to be initialized with sample rate, number of channels, and the required blocksize

```
espaceInstance->Initialize ( (int)inputFile->GetSampleRate(),
                             inputFile->GetNumOfChannels(),
                             0);

// allocate memory for sorted inputdata
for (i = 0; i < inputFile->GetNumOfChannels(); i++)
    ioData[i]   = (float*)malloc(sizeof(float)*_BLOCKSIZE);
```

Now, the impulse response can be read from file or memory, sorted to the correct input format and handed over to the SDK:

```
// read impulse response data
```

At this stage, parameters can be set, as can be seen in the following examples. In the first example, an envelope with only three points is defined and set (actually, this envelope consists only of a flat line and does nothing at all)

Setting the time stretching and pitch shifting factors for the early reflections can be done with the following code (which in this special case results in no alteration of the impulse response again):

To create a new filter and set its parameters (in this for the early reflections part), the code would look like

Other parameters can be set via the ::SetParameter function, here is an example for the PreDelay

After successfully setting the parameters, the function ::ApplyParameterChanges has to be called to let the changes take effect. This function can be called in a separate thread to allow processing while calculating the new impulse response in background.

To request either information about the impulse response analysis or the current parameter state, the function ::GetParameter can be called

To process a new block of audio data, the process function has to be called

After the processing, the created filters and the instance of espace realtime convolution have to be destroyed

The above code snippets demonstrated the basic functionality of the espace library. Most of the additional functions can be used similar to the given code examples. The exact functionality of the functions is described above.

1.2.6 Parameter Description

Despite some special parameters like time stretching, filtering and evelope, the parameters can be set via the function ::SetParameter which first parameter is the index of the parameter itself. The parameter indices are listed in the enum ::*Parameter_Indices* in the header file espaceAPI.h. Note that there are two distinct types of parameters, those that can be used to set some values and those that can only be requested because they are analysis results.
All parameter values are in float format; if the parameter is only on/off, then the allowed values are 0.0 (false/off) and 1.0 (true/on)

Changeable Parameters (alphabetical)

- kParamAutoVolumeScaleEnabled: enable/disable automatic volume scale to avoid clipping (allowed values 0 (off) and 1 (on), default: 1); please note that the scaling is dependent on the input signal and a slight correction of the output gain may be necessary nevertheless
- kParamBlockSize: set/get the processing block size in frames, has to be a power of two
- kParamBypass: enable/disable bypass (allowed values 0 (off) and 1 (on), default: 0)
- kParamInputMode: enable/disable downmixing of input to mono (allowed values 0 (no downmix) or 1 (downmix), default: 0)
- kParamPreDelay: set/get Pre-Delay (delay of impulse response) in ms (range 0...300, default: 0)
- kParamPreserveLength: enable/disable the truncation of ”unnecessary” frames at the begin and end of the impulse response (allowed values 0 (off) and 1 (on), default: 1)

- kParamResLPCutOff: set/get cut-off frequency of resonator low pass to be applied to the output signal in Hz (default: half sample rate)
- kParamResLPModDepth: set/get second cut-off frequency of resonator low pass to be applied to the output signal in Hz (default: half sample rate), the cut-off frequency is modulated between kParamResLPCutOff and kParamResLPModDepth
- kParamResLPModFreq: set/get modulation frequency of resonator low pass to be applied to the output signal in Hz (range 0...20, default: 0)
- kParamResLPResonance: set/get resonance of resonator low pass to be applied to the output signal (range 0 (no resonance)...4 (resonance), default: 0)
- kParamReverseIR: enable/disable to swap the whole impulse response from end to start (allowed values 0 (off) and 1 (on), default: 0)
- kParamXFadeLength: set/get crossfade length between ER and LR in ms (default: 50ms)
- kParamXPoint: set/get crosspoint between ER and LR in percent (range 0...1, 0 means start, 1 means end, default: impulse response dependent)
- kParamWetness: set/get wetness (dry/wet) in percent (range 0...1, 0 means dry, 1 means wet, default: 1)

Non-Changeable Parameters (alphabetical)

- kParamProcessLatency: get processing latency (equals block size) in s (see kParamBlockSize)
- kParamLengthOfOrigIR: get length of loaded impulse response in s (cannot be set)
- kParamLengthOfCurrentIR: get length of current processed (modified) impulse response in s (can be set only implicitely via stretch factors)
- kParamReverberationTime: get measured reverberation time in s (cannot be set)
- kParamNumOfTailSamples : get length of tail in frames (i.e. how many zeros have to be fed to the library to get the complete reverberation tail)
- kParamStretchER: get the stretch factor of early reflections in percent (can be set via ::SetStretchnPitch)
- kParamPitchER: get the pitch factor of early reflections in percent (can be set via ::SetStretchnPitch)
- kParamStretchLR: get the stretch factor of late reverberation in percent (can be set via ::SetStretchnPitch)
- kParamPitchLR: get the pitch factor of late reverberation in percent (can be set via ::SetStretchnPitch)

1.3 Licensing Issues

Please note that there might be intellectual property rights on the algorithms used. zplane cannot be held responsible for any possible infringements.

1.4 Support

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

zplane.development
grunewaldstr. 83
D-10823 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:

EnvelopePoint 9

espaceIf 10

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

espaceAPI.h
Interface of the espaceIf class 20

4 Class Documentation

4.1 _EnvelopePoint_ Struct Reference

`#include <espaceAPI.h>`

Collaboration diagram for _EnvelopePoint_:

4.2 espaceIf Class Reference

`#include <espaceAPI.h>`

Collaboration diagram for espaceIf:

Public Types

- enum Version_t {
  kMajor, kMinor, kPatch, kRevision,
  kNumVersionInts }

Public Member Functions

- espaceIf ()
- virtual ~espaceIf ()
- virtual zERROR Initialize (int iSampleRate, int iNumOfChannels, int iBlock↩ Size)=0
- virtual zERROR Process (const float **ppfInputData, float **ppfOutputData, int iNumOfFrames)=0
- virtual zERROR Reset ()=0
- virtual zERROR SetImpulseResponse (const float **ppfImpulseResponse, int iNumOfIRFrames, int iNumOfIRChannels, int iIRSampleRate)=0
- virtual zERROR GetProcessImpulseResponse (float **ppfImpulseResponse, int *piNumOfIRFrames, int *piNumOfIRChannels)=0
- virtual zERROR SetParameter (int iIndex, float fValue)=0
- virtual float GetParameter (int iIndex)=0
- virtual zERROR SetStretchnPitch (float fStretchFactor, float fPitchFactor, _IRPart_ ePart)=0
- virtual zERROR CreateFilter (void **phFilterHandle, _FilterTypes_ eType, _IRPart_ ePart)=0
- virtual zERROR DestroyFilter (void **phFilterHandle)=0
- virtual zERROR SetFilterParameter (void *phFilterHandle, float fFilterFreq, float fFilterGain=0, float fFilterQ=1)=0
- virtual zERROR GetFilterParameter (void *phFilterHandle, float *pfFilterFreq, float *pfFilterGain=0, float *pfFilterQ=0)=0
- virtual zERROR SetEnvelope (_EnvelopePoint_ **ppstEnvelopePoints, int i↩ NumOfEnvelopePoints)=0
- virtual int GetEnvelope (_EnvelopePoint_ **ppstEnvelopePoints=0)=0
- virtual zERROR ApplyParameterChanges (PFUNCallback pCallbackWhenReady=0, void *pUserData=0)=0

Static Public Member Functions

- static zERROR CreateInstance (espaceIf *&pCespace)
- static zERROR DestroyInstance (espaceIf *&pCespace)
- static const int GetVersion (const Version_t eVersionIdx)
- static const char * GetBuildDate ()

4.2.1 Detailed Description

CLASS

This class provides the interface for espace realtime parametric convolution.

4.2.2 Member Enumeration Documentation

Version_t `enum espaceIf::Version_t`

Enumerator

kMajor	
kMinor	
kPatch	
kRevision	
kNumVersionInts	

4.2.3 Constructor & Destructor Documentation

espaceIf() `espaceIf::espaceIf ( ) [inline]`

~espaceIf() `virtual espaceIf::~espaceIf ( ) [inline], [virtual]`

4.2.4 Member Function Documentation

CreateInstance() `static zERROR espaceIf::CreateInstance ( espaceIf *& pCespace ) [static]`

creates a new instance of espace

Parameters

pCespace	: handle to new instance, to be written

Returns

static zERROR : 0 when no error

DestroyInstance() `static zERROR espaceIf::DestroyInstance ( espaceIf *& pCespace ) [static]`

destroys an instance of espace

Parameters

pCespace	: handle to the instance to be destroyed

Returns

static zERROR : 0 when no error

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

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

Initialize()
```
virtual zERROR espaceIf::Initialize (
    int iSampleRate,
    int iNumOfChannels,
    int iBlockSize ) [pure virtual]
```
instance initialization and allocation of IR-independent memory

Parameters

iSampleRate	: sample rate of instance in Hz
iNumOfChannels	: number of channels of instance
iBlockSize	: processing block size in frames

Returns

virtual zERROR : 0 when no error

Process()
```
virtual zERROR espaceIf::Process (
    const float ** ppfInputData,
    float ** ppfOutputData,
    int iNumOfFrames ) [pure virtual]
```
process function (does the actual convolution), if the instance is not ready for processing, the output buffer is set to 0

Parameters

∗∗*ppfInputData*	: double pointer to the input buffer of samples [channels][samples]
∗∗*ppfOutputData*	: double pointer to the output buffer of samples [channels][samples], may be the same as the input buffer
iNumOfFrames	: number of input frames, must not exceed 16384

Returns

virtual zERROR : 0 when no error

Reset() `virtual zERROR espaceIf::Reset ( ) [pure virtual]`

reset processing buffers (but not impulse response/settings)

Parameters

none	

Returns

virtual zERROR : 0 when no error

SetImpulseResponse()
```
virtual zERROR espaceIf::SetImpulseResponse (
        const float ** ppfImpulseResponse,
        int iNumOfIRFrames,
        int iNumOfIRChannels,
        int iIRSampleRate ) [pure virtual]
```

set new impulse response for processing, this function has to be followed by a call of the function ::ApplyParameterChanges, except SetParameter/SetStretchnPitch/SetEnvelope/SetFilterParameter is called afterwards with other parameters

Parameters

***ppfImpulseResponse*	: double pointer to the impulse response buffer of samples [channels][samples]
iNumOfIRFrames	: number of frames
iNumOfIRChannels	: number of IR channels (has to equal iNumOfChannels or 1)
iIRSampleRate	: sample rate of IR (IR will be converted if iIRSampleRate != iSampleRate)

Returns

virtual zERROR : 0 when no error

GetProcessImpulseResponse()
```
virtual zERROR espaceIf::GetProcessImpulseResponse (
        float ** ppfImpulseResponse,
        int * piNumOfIRFrames,
        int * piNumOfIRChannels ) [pure virtual]
```

get the current impulse response (e.g. for display)

Parameters

∗∗*ppfImpulseResponse*	: double pointer to the impulse response buffer of samples [channels][samples], to be written. NOTE THAT THE MEMORY HAS TO BE ALLOCATED BY THE USER!!!
∗*piNumOfIRFrames*	: number of impulse response frames, to be written
∗*piNumOfIRChannels*	: number of impulse response channels, to be written

Returns

virtual zERROR : 0 when no error

SetParameter() `virtual zERROR espaceIf::SetParameter (`
`int iIndex,`
`float fValue ) [pure virtual]`

set parameter with index iIndex to value fValue, this function has to be followed by a call of the function ::ApplyParameterChanges, except SetParameter/SetStretchn↩ Pitch/SetEnvelope/SetFilterParameter is called afterwards with other parameters

Parameters

iIndex	: parameter index, enum ::*Parameter_Indices*
fValue	: value of parameter

Returns

virtual zERROR : 0 when no error

GetParameter() `virtual float espaceIf::GetParameter (`
`int iIndex ) [pure virtual]`

return value of parameter with index iIndex

Parameters

iIndex	: parameter index

Returns

virtual float : value of parameter with index iIndex

SetStretchnPitch() `virtual zERROR espaceIf::SetStretchnPitch (`
`float fStretchFactor,`

```
        float fPitchFactor,
        _IRPart_ ePart )  [pure virtual]
```

set stretch and pitch factor for early reflectionsor late reverberation, this function has to be followed by a call of the function ::ApplyParameterChanges, except Set↩ Parameter/SetStretchnPitch/SetEnvelope/SetFilterParameter is called afterwards with other parameters

Parameters

fStretchFactor	: stretch factor in percent (1.0 means no alteration)
fPitchFactor	: pitch factor in percent (1.0 means no alteration)
ePart	: part of the IR (early reflections or late reverberation)

Returns

virtual zERROR : 0 when no error

CreateFilter() `virtual zERROR espaceIf::CreateFilter (`

```
        void ** phFilterHandle,
        _FilterTypes_ eType,
        _IRPart_ ePart )  [pure virtual]
```

add a new filter either to early reflections or to late reverberation (up to ten filters per IR part are allowed), this function should be followed by a call of ::SetFilterParameter

Parameters

∗∗*phFilterHandle*	: handle to the filter, to be written
eType	: the type of the filter to be applied see ::*FilterTypes*
ePart	: the part of the IR where the filter should be applied (early reflections or late reverberation)

Returns

virtual zERROR : 0 when no error

DestroyFilter() `virtual zERROR espaceIf::DestroyFilter (`

```
        void ** phFilterHandle )  [pure virtual]
```

destroy a filter

Parameters

∗∗*phFilterHandle*	: handle to the filter to be destroyed

Returns

virtual zERROR : 0 when no error

SetFilterParameter() `virtual zERROR espaceIf::SetFilterParameter (`
`void * phFilterHandle,`
`float fFilterFreq,`
`float fFilterGain = 0,`
`float fFilterQ = 1 ) [pure virtual]`

set filter parameter for a special filter, this function has to be followed by a call of the function ::ApplyParameterChanges, except SetParameter/SetStretchnPitch/Set↩ Envelope/SetFilterParameter is called afterwards with other parameters

Parameters

∗phFilterHandle	: handle to the filter
fFilterFreq	: depending on the selected filter type the cut-off frequency or the mid frequency in Hz
fFilterGain	: gain of the filter in dB, only to be set if required by the filter type
fFilterQ	: Q of the filter, only to be set if required by the filter type

Returns

virtual zERROR : 0 when no error

GetFilterParameter() `virtual zERROR espaceIf::GetFilterParameter (`
`void * phFilterHandle,`
`float * pfFilterFreq,`
`float * pfFilterGain = 0,`
`float * pfFilterQ = 0 ) [pure virtual]`

returns the parameters of the filter

Parameters

∗phFilterHandle	: handle to the filter
pfFilterFreq	: depending on the selected filter type the cut-off frequency or the mid frequency in Hz
pfFilterGain	: gain of the filter in dB, only set if required by the filter type
pfFilterQ	: Q of the filter, only set if required by the filter type

Returns

virtual zERROR : 0 when no error

SetEnvelope() `virtual zERROR espaceIf::SetEnvelope ( _EnvelopePoint_ ** ppstEnvelopePoints, int iNumOfEnvelopePoints ) [pure virtual]`

set an envelope for the (whole) IR

Parameters

***ppstEnvelopePoints*	: array of pointers to structures containing amplitude and time points of envelope
iNumOfEnvelopePoints	: number of array elements

Returns

virtual zERROR : 0 when no error

GetEnvelope() `virtual int espaceIf::GetEnvelope ( _EnvelopePoint_ ** ppstEnvelopePoints = 0 ) [pure virtual]`

returns the current envelope

Parameters

***ppstEnvelopePoints*	: array of pointers to structures containing amplitude and time points of envelope, if 0, only the number of envelope points is returned

Returns

virtual int : number of envelope points

ApplyParameterChanges() `virtual zERROR espaceIf::ApplyParameterChanges ( PFUNCallback pCallbackWhenReady = 0, void * pUserData = 0 ) [pure virtual]`

this function has to be called after the call of the functions SetParameter/Set↩ Evelope/SetFilterParameter/SetStretchnPitch to apply the parameter changes

Parameters

pCallbackWhenReady	: pointer to callback function to be called if the parameters are applied
pUserData	: pointer to any user data that can be returned with the callback function

Returns

virtual zERROR : 0 when no error

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

- espaceAPI.h

5 File Documentation

5.1 docugen.txt File Reference

5.1.1 Detailed Description

source documentation main file

5.2 espaceAPI.h File Reference

interface of the espaceIf class.

Classes

- struct _EnvelopePoint_
- class espaceIf

Typedefs

- typedef int zERROR
- typedef void(∗ PFUNCallback) (void ∗pUserData)

Enumerations

- enum _FilterTypes_ {
  FilterTypeLowPass, FilterTypeHighPass, FilterTypeBandPass, FilterTypePeak,
  FilterTypeLowShelving, FilterTypeHighShelving }
- enum _IRPart_ { ER, LR }
- enum _Parameter_Indices_ {
  kParamIndexStart = 0, kParamInformalStart = 0, kParamProcessLatency, kParamLengthOfOrigIR,
  kParamLengthOfCurrentIR, kParamReverberationTime, kParamNumOfTailSamples,
  kParamStretchER,
  kParamPitchER, kParamStretchLR, kParamPitchLR, kParamFramesOfCurrentIR,
  kParamChannelsOfCurrentIR, kNumParamsInformal, kParamSettableStart = 100,
  kParamXPoint,
  kParamWetness, kParamPreDelay, kParamAutoVolumeScaleEnabled, kParamReverseIR,
  kParamPreserveLength, kParamBypass, kParamBlockSize, kParamProcessSampleRate,
  kParamInputMode, kParamXFadeLength, kParamResLPCutOff, kParamResLPResonance,
  kParamResLPModFreq, kParamResLPModDepth, kParamIndexEnd, kNumParamsSettable
  = (kParamResLPModDepth + 1 - kParamSettableStart) }

5.2.1 Detailed Description

:

5.2.2 Typedef Documentation

zERROR `typedef int zERROR`

PFUNCallback `typedef void(* PFUNCallback) (void *pUserData)`

this is the callback function prototype that can be called when all parameter changes are applied

5.2.3 Enumeration Type Documentation

FilterTypes `enum _FilterTypes_`

These are the allowed filter types applieable to either early reflections or late reverberation

Enumerator

FilterTypeLowPass	standard two pole low pass filter
FilterTypeHighPass	standard two pole high pass filter
FilterTypeBandPass	standard two pole band pass filter
FilterTypePeak	peak filter
FilterTypeLowShelving	low shelving filter
FilterTypeHighShelving	high shelving filter

IRPart `enum _IRPart_`

The impulse response (IR) is split in these distinct parts

Enumerator

ER	early reflections part (first part of IR)
LR	late reverberation part (second part of IR)

_Parameter_Indices_ `enum _Parameter_Indices_`

this enumeration holds the defines for all parameter indices

Enumerator

kParamIndexStart	not to be used
kParamInformalStart	not to be used

Enumerator

kParamProcessLatency	not settable (only informative), in s
kParamLengthOfOrigIR	not settable (only informative), in s
kParamLengthOfCurrentIR	not settable (only informative), in s
kParamReverberationTime	not settable (only informative), RT60 in s
kParamNumOfTailSamples	not settable (only informative), length of tail in samples (for last processing block)
kParamStretchER	not settable (only informative), in percent, use function SetStretchnPitch to set
kParamPitchER	not settable (only informative), in percent, use function SetStretchnPitch to set
kParamStretchLR	not settable (only informative), in percent, use function SetStretchnPitch to set
kParamPitchLR	not settable (only informative), in percent, use function SetStretchnPitch to set
kParamFramesOfCurrentIR	not settable (only informative), in frames
kParamChannelsOfCurrentIR	not settable (only informative)
kNumParamsInformal	not to be used
kParamSettableStart	not to be used
kParamXPoint	crosspoint between ER and LR in percent (range 0...1, default: IR dependent)
kParamWetness	wetness of output signal in percent (range 0...1, default: 1)
kParamPreDelay	predelay in ms (range 0...300, default: 0)
kParamAutoVolumeScaleEnabled	enabling/disabling of automatic volume scale (boolean, range 0 or 1, default: 1)
kParamReverseIR	bool if whole IR should be flipped (boolean, range 0 or 1, default: 0)
kParamPreserveLength	level threshold in dBFS to truncate IR length (<=-96: no truncation, default: -96)
kParamBypass	bool if input signal is bypassed (boolean, range 0 or 1, default: 0)
kParamBlockSize	length of process blocks in frames
kParamProcessSampleRate	processing sample rate in Hz, has to the input samplerate times 0.5, 0.25 or 0.125
kParamInputMode	bool if input is downmixed to mono or not (boolean, range 0 or 1, default: 0 equals no downmix)
kParamXFadeLength	length of crossfade between ER and LR in s
kParamResLPCutOff	cut-off frequency of resonator low pass to be applied to the output signal in Hz
kParamResLPResonance	resonance of resonator low pass to be applied to the output signal (value between 0 and 4)

Enumerator

kParamResLPModFreq	modulation frequency of resonator low pass to be applied to the output signal ($<$ 20Hz)
kParamResLPModDepth	second cut-off frequency of resonator low pass to be applied to the output signal, the modulation varies between kParamResLPCutOff and this parameter
kParamIndexEnd	not to be used
kNumParamsSettable	not to be used