



SDK Manual

Tim Flohrer

(c) 2008 by zplane.development

August 1, 2008

Contents

1	élastique 2.1 Documentation	1
1.1	Introduction	1
1.2	What's new in V2.1	1
1.2.1	Pitch synchronization explained	2
1.3	API Documentation	3
1.3.1	Memory Allocation	4
1.3.2	Naming Conventions	4
1.3.3	Stereo Processing	4
1.3.4	C++ API description	4
1.3.5	C++ DirectAPI description	8
1.4	Coding Style minimal overview	8
1.5	Command Line Usage Example	9
1.6	Support	9
2	élastique Time Stretching 2.1 SDK Directory Hierarchy	10
2.1	élastique Time Stretching 2.1 SDK Directories	10
3	élastique Time Stretching 2.1 SDK Hierarchical Index	10
3.1	élastique Time Stretching 2.1 SDK Class Hierarchy	10
4	élastique Time Stretching 2.1 SDK Class Index	10
4.1	élastique Time Stretching 2.1 SDK Class List	10
5	élastique Time Stretching 2.1 SDK File Index	10
5.1	élastique Time Stretching 2.1 SDK File List	10
6	élastique Time Stretching 2.1 SDK Directory Documentation	11
6.1	incl/ Directory Reference	11
7	élastique Time Stretching 2.1 SDK Class Documentation	11
7.1	CElastiqueIf Class Reference	11
7.1.1	Detailed Description	11
7.1.2	Member Enumeration Documentation	12
7.1.3	Member Function Documentation	12

8 élastique Time Stretching 2.1 SDK File Documentation	17
8.1 docugen.txt File Reference	17
8.1.1 Detailed Description	17
8.2 elastiqueAPI.h File Reference	17
8.2.1 Detailed Description	17

1 élastique 2.1 Documentation

1.1 Introduction

élastique is a completely new time-stretch algorithm that outperforms other algorithms in quality and performance. Due to new technologies élastique allows time compression and expansion of any audio material without receivable loss in quality. Of élastique is capable of running in realtime.

Since version 1.1, élastique is not only able to timescale the input data, but also to pitch shift it. Since élastique is a general music algorithm, it should be noted that the pitch shifting cannot be formant preserving. For formant preserving pitchshifting, zplane's élastique SOLOIST SDK would be the best choice.

The TrueFreq and TrueTrans technologies developed by zplane render a realtime time-stretching quality never heard before. One disadvantage of other algorithms is always the insufficient frequency resolution that usually leads to frequency smearing.

TrueFreq encounters this problem by using a unique technique to determine the true frequencies independently of the frequency resolution. This enables élastique to generate a flawless sound even at high stretch factors.

TrueTrans analyses the audio matrial, automatically detects transients and leaves them unchanged. Thus, TrueTrans provides sharp attacks and perfectly stretched drum-loops with perfectly accurate timing.

The project contains the required libraries for operating system élastique was licensed for with the appropriate header files.

The structure of this document is as following: First the API of the élastique library is described. The API documentation contains naming conventions, function descriptions of the C++-API as well as the C-API. The following usage examples (available as source code for compiling the test application) give a clear example on how to use the API in a real world application. Afterwards, a short description of the usage of the compiled example application is given.

1.2 What's new in V2.1

- improved performance (~5-10%) for the normal API of élastiquePro due to internal structural changes
- improved performance (~15-25%) for all élastique efficient modes when using either pitching or SetStretchPitchQFactor(.)

- reduced occasional performance peak when calling Reset() (this was most probably due to some cache issues)
- new feature allowing improved synchronization of the stretch and resample engine (see below)

1.2.1 Pitch synchronization explained

Pitchshifting in *élastique* (Pro/efficient) is done by combining the time stretch engine with a resampler. So, for example, for pitch shifting one octave up, the resampler downsamples the signal to half the rate resulting in pitch and speed doubling when played at the original sample rate. The time stretch engine now stretches the result by a factor of two, so that the final output has the original tempo but the pitch is doubled.

The resampler is able to switch the samplerate immediately while due to the block based overlap-and-add procedure the time stretch engine smoothes the transition. At that point the resampler and the time stretch engine are not synchronized leading to variable (positive or negative) latency depending on the pitch factor. The following two pictures illustrate the behavior. Please note that this behavior only occurs when dynamic pitching is used. With a constant pitch factor both mode yield the same result.

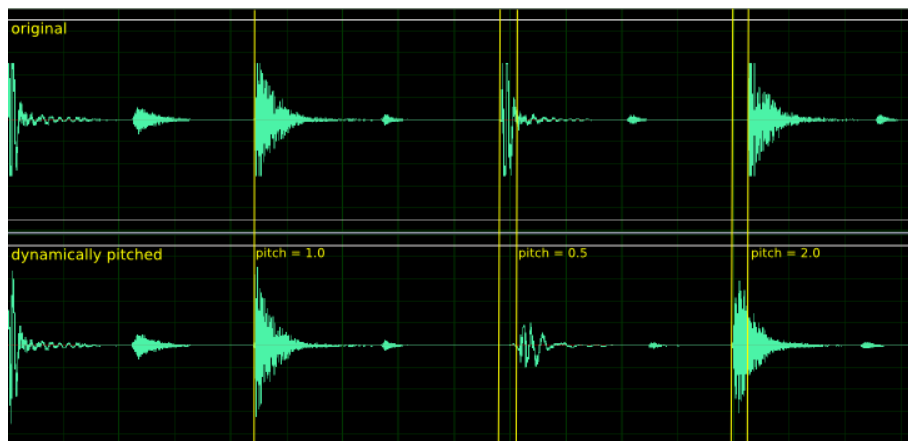


Figure 1: Original and pitched audio without sync

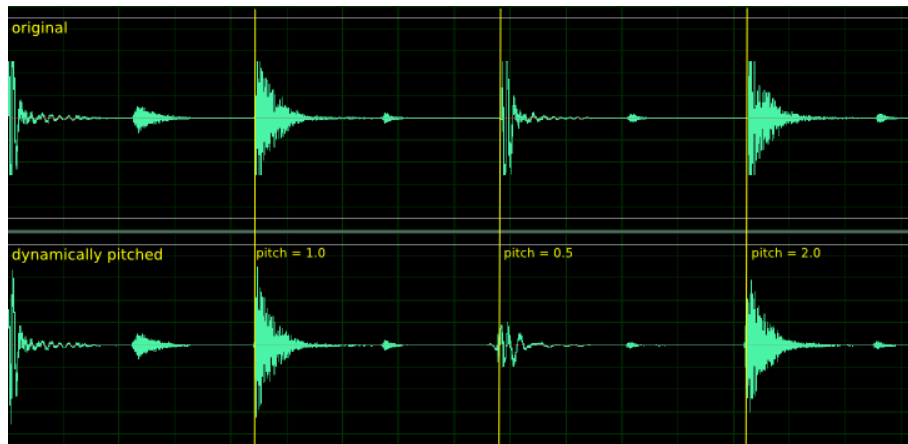


Figure 2: Original and pitched audio with sync

While the advantages of the synchronized mode are obvious, the un-synchronized mode still has some advantages over the synchronized mode, when the introduced latency is negligible (e.g. when only using small pitch variations).

- in un-synchronized mode the pitching is done immediately, while in the synched mode it will need some time to reach the desired pitch.
- using the DirectAPI only for pitching the un-synchronized mode maintains constant output blocksizes, while in synchronized mode the output blocksizes may vary during the synchronization process requiring a larger buffer to compensate these variations in a realtime application.

The API has been updated for this. `SetStretchPitchQFactor(.)` and `SetStretchQPitchFactor(.)` have a third boolean parameter, that when set to true enables the pitch synchronization. Otherwise when not set the default is "no-sync" maintaining the behavior of previous versions of the API.

1.3 API Documentation

élastique offers two different APIs, one in ANSI-C and the other in ANSI-C++. The C++-API can be accessed via the file [elastiqueAPI.h](#), where the class `CElastiqueIf` provides the interface for élastique. The C-API can be found in the file `elastiqueAPI_C.h`. The C-API differs from the C++-API mainly in two properties: Firstly, every API-function name is extended with a preceding `Elastique_`, and secondly, every C-API function has an additional parameter `void* pElastiqueHandle`.

Since version 1.3 there is an alternative API called `elastiqueDirectAPI`. This alternative API offers the possibility to split up the processing calls in order to avoid peak performance. This API has been improved with version 2.0. The downside of that is that a lot of buffer handling is left to user.

All variable types needed are either defined in file [elastiqueAPI.h](#) or standard C++-types. Error codes are defined in `zErrorCodes.h`.

1.3.1 Memory Allocation

The élastique SDK does not allocate any buffers handled by the calling application. The input buffer as well as the output buffer has to be allocated by the calling application. The exact size of the output buffer is defined by the user with the function call of [CElastiqueIf::CreateInstance](#) (.) (C++-API, see [C++ API description](#) for details) resp. `Elastique_CreateInstance` (.) (C-API, see `apic` for details). The maximum size of the input buffer in frames depends on the output buffer size via the formula:

```
InputBufferSizeInFrames = m_pCMyElastiqueInstance->GetMaxFramesNeeded();
```

1.3.2 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.

1.3.3 Stereo Processing

When processing stereo input, it is strongly recommended to use élastique with a stereo instance, not with two mono instances. This has two reasons: quality and performance. The quality will be better, since the content of both channels is taken into account for the analysis, and the stereo processing is linked between both channels. The performance will be better since many analysis steps can be combined for both channels.

1.3.4 C++ API description

1.3.4.1 Required Functions The following functions have to be called when using the élastique library. Description see below.

- [CElastiqueIf::CreateInstance](#)(.)
description see below
- [CElastiqueIf::ProcessData](#)(.)
description see below
- [CElastiqueIf::DestroyInstance](#)(.)
description see below

1.3.4.2 Complete Function Description

Instance Handling Functions

- `int CElastiqueIf::CreateInstance (CElastiqueIf*& cCElastique, int i-OutputBufferSize, int iNumOfChannels, float SampleRate, _elastique_-proc_mode eMode)`

Creates a new instance of the *élastique* time stretcher. The handle to the new instance is returned in parameter **cCElastique*. The requested size of the output buffer is given in frames in parameter *iOutputBufferSize*. The output buffer size must not exceed the value of 4096 frames. The parameter *iNumOfChannels* describes the number of channels with which *élastique* is used. Possible are either 1 channel (Mono) or two channels (stereo). *fSampleRate* denotes the input samplerate and *eMode* chooses the processing mode. The processing mode is either *_TRANSIENT_MODE* which should be used as default or *_TONAL_MODE* which is appropriate if one experiences artifacts in tonal material.

If the function fails, the return value is not 0.

The use of this function is required.

- **[int CElastiqueIf::DestroyInstance](#)** (*CElastiqueIf* cCElastique*)

Destroys the instance of the *élastique* time stretcher given in parameter **cCElastique*.

If the function fails, the return value is not 0.

The use of this function is required.

Timestretching and Pitch Shifting Functions

- **[int CElastiqueIf::SetStretchQPitchFactor](#)** (*float& fStretchFactor*, *float fPitchFactor*)

Sets the stretch and pitch factors for the *élastique* timestretching engine instance. The parameter *fStretchFactor* is given in percent of the output length. A stretch factor of 1 (=100%) does not alter the output length. A stretch factor of 0.5 (=50%) will result in an output signal with doubled speed and halved size. The allowed range for the stretch factor is from 0.1 (=10%) up to 10.0 (=1000%). The parameter *fPitchFactor* is given in percent of the input pitch. The allowed range for the pitch factor is from 0.25 (=25%) up to 4.0 (=400%).

The function may be called as often as needed, but not during the processing of a block. Only calls before or after processing a single block are allowed.

Due to the algorithmic properties of the *élastique* time stretching engine, the stretch factor has to be slightly quantized. Therefore, the quantized stretch factor is given back to the calling application. The maximum quantization error will be max. 0.02% and therefore hardly recognisable. However, over a long time the quantization of the stretch factor can be an issue, e.g. when using *élastique* for the time alignment of very long signals. Two different workarounds are applicable to circumvent these issues:

- use the function [CElastiqueIf::SetStretchPitchQFactor](#) to quantize the pitch factor instead of the stretch factor. This will lead to a small pitch shift in the signal.
- vary the stretch factor slightly over time (e.g. every 100th block) to get the overall stretch factor accurate. Since the quantization is on a very low level, these variations will not be recognizable at all.

If the function fails, the return value is not 0.

- **int [CElastiqueIf::SetStretchPitchQFactor](#)** (float fStretchFactor, float& fPitchFactor)

This function will do exactly the same as [CElastiqueIf::SetStretchQPitchFactor](#) (.), but quantize the pitch factor instead of the stretch factor. For a more detailed description, see the description of [CElastiqueIf::SetStretchQPitchFactor](#) (.).

- **int [CElastiqueIf::SetCutOffFreq](#)** (float fFreq)

In order to save processing time one may reduce the audio bandwidth. fFreq is the cutoff frequency in Hz. Usually Fs/4 gives a speed increase of about 20~25%.

- **int [CElastiqueIf::GetFramesNeeded](#)** ()

Returns the required number of input samples for the upcoming processing block. This function has to be called before each processing step to assure correct input buffer sizes.

- **int [CElastiqueIf::ProcessData](#)** (float** ppInSampleData, int iNumOfInFrames, float** ppOutSampleData)

Processes the input data and returns the stretched output data. The input as well as the output data is given as an array of pointers to the data. This means that, for example, ppInSampleData[0] is a pointer to the float input data buffer of the first channel while ppInSampleData[1] is the pointer to the float input data buffer of the second input channel. The range of the audio input data is +-1.0F. The parameter iNumOfInFrames describes the number of input frames. Please make sure that this parameter iNumOfInFrames equals the value returned by the function [CElastiqueIf::GetFramesNeeded](#) () to assure the requested output buffer size.

If no input data is available anymore, the last output blocks in the internal buffers can requested with the call of [CElastiqueIf::ProcessData](#) (.) with parameter iNumOfInFrames set to 0, until the function returns -1. If the function fails, the return value is not 0.

The use of this function is required.

- **void [CElastiqueIf::FlushBuffer](#)** (float** ppfOutSampleData)

Gets all the remaining internal frames that do not fit into the output buffer size and write them into parameter ppfOutSampleData. Returns the number of written samples.

The use of this function is optional.

- **void [CElastiqueIf::Reset](#)** ()

Sets all internal buffers to their initial state. The call of this function is needed before using the same instance of *élastique* for a different input signal.

The use of this function is optional.

1.3.4.3 C++ Usage example The complete code can be found in the example source file `élastiqueTestCLMain.cpp`.

In the first step, a handle to the *élastique* instance has to be declared:

```
CElastiqueIf *pcElastiqueHandle = 0;
```

Then, an instance of `élastique` object is created. The output size in frames for this example is defined in `_OUTPUT_BUFFER_SIZE`.

```
iError = CElastiqueIf::CreateInstance( pcElastiqueHandle,
    _OUTPUT_BUFFER_SIZE,
    iNumOfChannels,
    (float)iSampleRate,
    (CElastiqueIf::_elastique_proc_mode) iMode);

if (iError)
{
    fprintf(stderr, "Instance Create Error!\n");
    return -1;
}
```

Now, let's see how much memory we have to allocate.

```
iMaxBufferSize = pcElastiqueHandle->GetMaxFramesNeeded();
```

Afterwards, the stretch and pitch factors are set to the values in variables `fStretchRatio` and `fPitchRatio`.

```
if (pcElastiqueHandle->SetStretchQPitchFactor( fStretchRatio, fPitchFactor) != 0)
```

In the next step, the processing loop can be executed

```
while (bReadNextFrame)
{
    // check how much frames elastique expects at this run
    iNumOfInFrames = pcElastiqueHandle->GetFramesNeeded();
```

In the first step, we have to ask `élastique` how much input frames are needed in this processing step to get the required number of output frames, as already defined in `CreateInstance`.

In this example, the input data is read from a WAV file with 16bit interleaved audio data. The Audio File I/O library is provided with the example, but is not part of the SDK.

```
iNumSamplesRead = pcInputFile->Read(apfProcessInputData, iNumOfInFrames);
```

Afterwards, the process function can be called with the required number of input frames:

```
iError = pcElastiqueHandle->ProcessData( apfProcessInputData,
```

If there was no processing error, we know the exact output buffer size and therefore can simply resort the output data and write it to an output file

```
if (iTotalFrames > iFinalFrames)
{
    pCOutputFile->Write(apfProcessOutputData,
        (_OUTPUT_BUFFER_SIZE - (iTotalFrames - iFinalFrames)));
    break;
}
else
    pCOutputFile->Write(apfProcessOutputData, (_OUTPUT_BUFFER_SIZE));
```

and are at the end of the processing loop:

```
} // while (bReadNextFrame)
```

After the processing loop was exited, there will be remaining internal buffers in the `elastic` instance that we want to get. To get these remaining frames, the function `FlushBuffer` can be called:

```
if (iTotalFrames < iFinalFrames)
    while ((iNumOfFrames = pcElasticHandle->FlushBuffer(apfProcessOutputData)) > 0)
    {

        iTotalFrames += iNumOfFrames;
        if (iTotalFrames > iFinalFrames)
        {
            pCOutputFile->Write(apfProcessOutputData,
                (iNumOfFrames - (iTotalFrames - iFinalFrames)));
            break;
        }
        else
            pCOutputFile->Write(apfProcessOutputData, (iNumOfFrames));
    }
```

After the successful processing, the instance of `elastic` can be destroyed

```
// delete instance
CElasticIf::DestroyInstance(pcElasticHandle);
```

1.3.5 C++ DirectAPI description

Since version 2.0 there is a separate API description for the Direct Interface for `elastic` efficient and `elastic Pro`. This API description is written for the `Pro` interface but also applies to the efficient interface. So, please refer to that manual.

1.4 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
class/struct:	c

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

1.5 Command Line Usage Example

The compiled example is a command line application that reads and writes audio files in PCM (16bit RAW) format. RAW format means that the file has no header but contains only the pure data.

Since the example application has no sophisticated command line parser, so the order of the arguments is crucial. The command line synopsis is:

```
elastiqueTest input_file output_file StretchRatio [PitchRatio] [InputSampleRate] [NumberOfChannels]
```

The following command line will result in an output file of the same format as the input file (RAW PCM, 16bit, 48kHz, stereo interleaved, name: `input.pcm`) that is 10% longer as the original (i.e. stretch factor 1.1) and its pitch is 1% lower than in the original (i.e. pitch factor 0.99):

```
elastiqueTest input48.pcm output48.pcm 1.1 0.99 48000 2
```

1.6 Support

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

zplane.development

tim flohrer

holsteinische str. 39-42

aufgang 8

D-12161 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: flohrrer@zplane.de

2 élastique Time Stretching 2.1 SDK Directory Hierarchy

2.1 élastique Time Stretching 2.1 SDK Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

[incl](#) **11**

3 élastique Time Stretching 2.1 SDK Hierarchical Index

3.1 élastique Time Stretching 2.1 SDK Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

[CElastiqueIf](#) **11**

4 élastique Time Stretching 2.1 SDK Class Index

4.1 élastique Time Stretching 2.1 SDK Class List

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

[CElastiqueIf](#) **11**

5 élastique Time Stretching 2.1 SDK File Index

5.1 élastique Time Stretching 2.1 SDK File List

Here is a list of all documented files with brief descriptions:

[elastiqueAPI.h](#) **17**

6 élastique Time Stretching 2.1 SDK Directory Documentation

6.1 incl/ Directory Reference

Files

- file [elastiqueAPI.h](#)

7 élastique Time Stretching 2.1 SDK Class Documentation

7.1 CElastiqueIf Class Reference

```
#include <elastiqueAPI.h>
```

7.1.1 Detailed Description

CLASS

This class provides the interface for zplane's élastique class.

USAGE

Definition at line 129 of file elastiqueAPI.h.

Public Types

- enum [_elastique_proc_mode](#) { [_TRANSIENT_MODE](#), [_TONAL_MODE](#) }
- enum [_ElastiqueStereoInputMode_](#) { [_PLAIN_STEREO](#) = 0, [_MS_STEREO](#) }

Public Member Functions

- virtual [~CElastiqueIf](#) ()
- virtual int [ProcessData](#) (float **ppInSampleData, int iNumOfInFrames, float **ppOutSampleData)=0
- virtual int [GetFramesNeeded](#) (int iOutBufferSize)=0
- virtual int [GetFramesNeeded](#) ()=0
- virtual int [GetMaxFramesNeeded](#) (float fMinStretchFactor=0.1, float fMinPitchFactor=1.0)=0
- virtual int [SetStretchQPitchFactor](#) (float &fStretchFactor, float fPitchFactor, bool bUsePitchSync=false)=0
- virtual int [SetStretchPitchQFactor](#) (float fStretchFactor, float &fPitchFactor, bool bUsePitchSync=false)=0
- virtual void [Reset](#) ()=0

- virtual int [FlushBuffer](#) (float **ppfOutSampleData)=0
- virtual int [GetFramesBuffered](#) ()=0
- virtual int [SetStereoInputMode](#) ([_ElastiqueStereoInputMode_](#) eStereoInputMode)=0
- virtual int [SetCutOffFreq](#) (float fFreq)=0
- virtual int [GetTimePos](#) ()=0

Static Public Member Functions

- static char * [GetVersionString](#) ()
- static char * [GetBuildDateString](#) ()
- static int [CreateInstance](#) ([CElastiqueIf](#) *&cCElastique, int iOutputBufferSize, int iNumOfChannels, float SampleRate, [_elastique_proc_mode](#) eMode)
- static int [DestroyInstance](#) ([CElastiqueIf](#) *cCElastique)

7.1.2 Member Enumeration Documentation

7.1.2.1 enum [CElastiqueIf::_elastique_proc_mode](#)

Enumerator:

- [_TRANSIENT_MODE](#) mode optimized for audio rich of transients
- [_TONAL_MODE](#) DEPRECATED: mode optimized for audio rich of tonal components, only here for legacy reasons.

Definition at line 135 of file [elastiqueAPI.h](#).

7.1.2.2 enum [CElastiqueIf::_ElastiqueStereoInputMode_](#)

Enumerator:

- [_PLAIN_STEREO](#) normal LR stereo mode
- [_MS_STEREO](#) MS stereo mode M must be in channel 0 and S in channel 1.

Definition at line 141 of file [elastiqueAPI.h](#).

7.1.3 Member Function Documentation

7.1.3.1 static int [CElastiqueIf::CreateInstance](#) ([CElastiqueIf](#) *& *cCElastique*, int *iOutputBufferSize*, int *iNumOfChannels*, float *SampleRate*, [_elastique_proc_mode](#) *eMode*) [static]

creates an instance of zplane's *élastique* class

Parameters:

- cCElastique* : returns a pointer to the class instance
- iOutputBufferSize* : desired number of frames at the output

iNumOfChannels : number of channels (1..2)
fSampleRate : input samplerate
eMode : either transient or tonal optimized mode

Returns:

static int : returns some error code otherwise NULL

7.1.3.2 static int CElastiqueIf::DestroyInstance (CElastiqueIf * cCElastique)
 [static]

destroys an instance of the zplane's élastique class

Parameters:

cCElastique : pointer to the instance to be destroyed

Returns:

static int : returns some error code otherwise NULL

7.1.3.3 virtual int CElastiqueIf::FlushBuffer (float ** ppfOutSampleData)
 [pure virtual]

gets the last frames in the internal buffer, ProcessData must have returned -1 before.

Parameters:

ppfOutSampleData, double pointer to the output buffer of samples
 [channels][samples]

Returns:

int : returns some error code otherwise NULL

7.1.3.4 static char* CElastiqueIf::GetBuildDateString () [static]

returns the build date as string

Returns:

static char* : returns a pointer to build date string

7.1.3.5 virtual int CElastiqueIf::GetFramesBuffered () [pure virtual]

returns the number of frames already buffered in the output buffer. The number of frames is divided by the stretch factor, so that one is able to calculate the current position within the source file.

Parameters:

none

Returns:

virtual int : returns the number of frames buffered

7.1.3.6 virtual int CElastiqueIf::GetFramesNeeded () [pure virtual]

returns the number of frames needed for the next processing step, this function should be always called directly before [CElastiqueIf::ProcessData\(..\)](#)

Parameters:

none

Returns:

virtual int : returns the number of frames required

7.1.3.7 virtual int CElastiqueIf::GetFramesNeeded (int iOutBufferSize) [pure virtual]

returns the number of frames needed for the next processing step according to the required buffersize, this function should be always called directly before [CElastiqueIf::ProcessData\(..\)](#) this function changes the internal output buffer size as specified when calling [CElastiqueIf::CreateInstance\(..\)](#) use this function if want to change the output size.

Parameters:

iOutBufferSize : required output buffer size

Returns:

virtual int : returns the number of frames required

7.1.3.8 virtual int CElastiqueIf::GetMaxFramesNeeded (float fMinStretchFactor = 0.1, float fMinPitchFactor = 1.0) [pure virtual]

returns the maximum number of frames needed for a given minimum factor

Parameters:

float fMinStretchFactor : the minimum stretch factor to be used

float fMinPitchFactor : the minimum pitch factor to be used

Returns:

virtual int : returns number of frames

7.1.3.9 virtual int CElastiqueIf::GetTimePos () [pure virtual]

gets the current input time position

Returns:

virtual int : returns the time position

7.1.3.10 static char* CElastiqueIf::GetVersionString () [static]

returns the version as string

Returns:

static char* : returns a pointer to version string

7.1.3.11 virtual int CElastiqueIf::ProcessData (float ** ppInSampleData, int i-NumOfInFrames, float ** ppOutSampleData) [pure virtual]

does the actual processing if the number of frames provided is as retrieved by [CElastiqueIf::GetFramesNeeded\(\)](#) this function always returns the number of frames as specified when calling [CElastiqueIf::CreateInstance\(..\)](#)

Parameters:

ppInSampleData : double pointer to the input buffer of samples
[channels][samples]

iNumOfInFrames : the number of input frames

ppOutSampleData : double pointer to the output buffer of samples
[channels][samples]

Returns:

virtual int : returns the error code, otherwise 0

7.1.3.12 virtual void CElastiqueIf::Reset () [pure virtual]

resets the internal state of the élastique algorithm

Parameters:

none

Returns:

int : returns some error code otherwise NULL

7.1.3.13 virtual int CElastiqueIf::SetCutOffFreq (float fFreq) [pure virtual]

sets a cutoff frequency in order to save processing time

Parameters:

fFreq : cutoff freq in Hz

Returns:

virtual int : returns some error code otherwise NULL

7.1.3.14 virtual int CElastiqueIf::SetStereoInputMode ([_ElastiqueStereoInputMode](#), [eStereoInputMode](#)) [pure virtual]

sets the stereo input mode

Parameters:

eStereoInputMode : sets the mode according to [_ElastiqueStereoInputMode](#)

Returns:

static int : returns some error code otherwise NULL

7.1.3.15 virtual int CElastiqueIf::SetStretchPitchQFactor (float *fStretchFactor*, float & *fPitchFactor*, bool *bUsePitchSync* = false) [pure virtual]

sets the internal stretch & pitch factor. A value between 0.1 and 10.0 for stretching is valid. The pitch factor is quantized. The product of the stretch factor and the pitch factor must be between 0.1 and 10.0.

Parameters:

fStretchFactor : stretch factor 0.1 - 10.0 (1.0 is default and does nothing)

fPitchFactor,: pitch factor 0.25 - 4.0 (1.0 is default and does nothing)

bUsePitchSync,: synchronizes timestretch and pitchshifting (default is set to `_FALSE` in order to preserve old behavior, see documentation for V2.1), this is ignored in SOLO modes

Returns:

int : returns the error code, otherwise 0

7.1.3.16 virtual int CElastiqueIf::SetStretchQPitchFactor (float & *fStretchFactor*, float *fPitchFactor*, bool *bUsePitchSync* = false) [pure virtual]

sets the internal stretch & pitch factor. A value between 0.1 and 10.0 for stretching is valid. The stretch factor is quantized. The product of the stretch factor and the pitch factor must be between 0.1 and 10.0.

Parameters:

fStretchFactor : stretch factor 0.1 - 10.0 (1.0 is default and does nothing)

fPitchFactor,: pitch factor 0.25 - 4.0 (1.0 is default and does nothing)

bUsePitchSync,: synchronizes timestretch and pitchshifting (default is set to `_FALSE` in order to preserve old behavior, see documentation for V2.1), this is ignored in SOLO modes

Returns:

int : returns the error code, otherwise 0

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

- [elastiqueAPI.h](#)

8 élastique Time Stretching 2.1 SDK File Documentation

8.1 docugen.txt File Reference

8.1.1 Detailed Description

source documentation main file

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

8.2 elastiqueAPI.h File Reference

8.2.1 Detailed Description

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

Classes

- class [CElastiqueIf](#)

Defines

- #define [__libELASTIQUEAPI_HEADER_INCLUDED__](#)

Index

- [_ElastiqueStereoInputMode_](#)
 - [CElastiqueIf, 12](#)
 - [_MS_STEREO](#)
 - [CElastiqueIf, 12](#)
 - [_PLAIN_STEREO](#)
 - [CElastiqueIf, 12](#)
 - [_TONAL_MODE](#)
 - [CElastiqueIf, 12](#)
 - [_TRANSIENT_MODE](#)
 - [CElastiqueIf, 12](#)
 - [_elastique_proc_mode](#)
 - [CElastiqueIf, 12](#)
- [CElastiqueIf, 11](#)
 - [_MS_STEREO, 12](#)
 - [_PLAIN_STEREO, 12](#)
 - [_TONAL_MODE, 12](#)
 - [_TRANSIENT_MODE, 12](#)
- [CElastiqueIf](#)
 - [_ElastiqueStereoInputMode_, 12](#)
 - [_elastique_proc_mode, 12](#)
 - [CreateInstance, 12](#)
 - [DestroyInstance, 13](#)
 - [FlushBuffer, 13](#)
 - [GetBuildDateString, 13](#)
 - [GetFramesBuffered, 13](#)
 - [GetFramesNeeded, 13, 14](#)
 - [GetMaxFramesNeeded, 14](#)
 - [GetTimePos, 14](#)
 - [GetVersionString, 14](#)
 - [ProcessData, 15](#)
 - [Reset, 15](#)
 - [SetCutOffFreq, 15](#)
 - [SetStereoInputMode, 15](#)
 - [SetStretchPitchQFactor, 16](#)
 - [SetStretchQPitchFactor, 16](#)
- [CreateInstance](#)
 - [CElastiqueIf, 12](#)
- [DestroyInstance](#)
 - [CElastiqueIf, 13](#)
- [docugen.txt, 17](#)
- [elastiqueAPI.h, 17](#)
- [FlushBuffer](#)
 - [CElastiqueIf, 13](#)
- [GetBuildDateString](#)
 - [CElastiqueIf, 13](#)
- [GetFramesBuffered](#)
 - [CElastiqueIf, 13](#)
- [GetFramesNeeded](#)
 - [CElastiqueIf, 13, 14](#)
- [GetMaxFramesNeeded](#)
 - [CElastiqueIf, 14](#)
- [GetTimePos](#)
 - [CElastiqueIf, 14](#)
- [GetVersionString](#)
 - [CElastiqueIf, 14](#)
- [incl/ Directory Reference, 11](#)
- [ProcessData](#)
 - [CElastiqueIf, 15](#)
- [Reset](#)
 - [CElastiqueIf, 15](#)
- [SetCutOffFreq](#)
 - [CElastiqueIf, 15](#)
- [SetStereoInputMode](#)
 - [CElastiqueIf, 15](#)
- [SetStretchPitchQFactor](#)
 - [CElastiqueIf, 16](#)
- [SetStretchQPitchFactor](#)
 - [CElastiqueIf, 16](#)