



SDK Manual

Tim Flohrer

(c) 2008 by zplane.development

August 1, 2008

Contents

1	élastique Pro V2.1 Time-Stretching/Pitch-Shifting SDK Documentation	1
1.1	Introduction	1
1.2	What's new in V2.1	2
1.2.1	Pitch synchronization explained	2
1.3	API Documentation	3
1.3.1	Memory Allocation	3
1.3.2	Naming Conventions	4
1.3.3	Stereo and Multichannel-Processing	4
1.3.4	Processing Modes	4
1.3.5	C++ API description	5
1.4	Delivered Files (example project)	9
1.4.1	File Structure	9
1.5	Coding Style minimal overview	10
1.6	Command Line Usage Example	10
1.7	Support	11
2	élastique Pro V2.1 SDK Documentation Directory Hierarchy	11
2.1	élastique Pro V2.1 SDK Documentation Directories	11
3	élastique Pro V2.1 SDK Documentation Hierarchical Index	11
3.1	élastique Pro V2.1 SDK Documentation Class Hierarchy	11
4	élastique Pro V2.1 SDK Documentation Class Index	11
4.1	élastique Pro V2.1 SDK Documentation Class List	11
5	élastique Pro V2.1 SDK Documentation File Index	12
5.1	élastique Pro V2.1 SDK Documentation File List	12
6	élastique Pro V2.1 SDK Documentation Directory Documentation	12
6.1	incl/ Directory Reference	12
7	élastique Pro V2.1 SDK Documentation Class Documentation	12
7.1	CElastiqueProIf Class Reference	12
7.1.1	Detailed Description	12
7.1.2	Member Enumeration Documentation	13

1 élastique Pro V2.1 Time-Stretching/Pitch-Shifting SDK Documentation **1**

7.1.3	Constructor & Destructor Documentation	14
7.1.4	Member Function Documentation	14
8	élastique Pro V2.1 SDK Documentation File Documentation	19
8.1	docugen.txt File Reference	19
8.2	élastiqueProAPI.h File Reference	19
8.2.1	Detailed Description	19
8.2.2	Define Documentation	19

1 élastique Pro V2.1 Time-Stretching/Pitch-Shifting SDK Documentation

1.1 Introduction

With élastique Pro, zplane introduces a completely new general purpose time stretch engine. It offers a unmatched quality and easily exceeds the high quality demands of professional productions and broadcast applications while still being very efficient.

Furthermore, the élastique Pro SDK package includes the "classic" élastique engine (now called élastique efficient) for highest efficiency and the élastique SOLOIST engine for special monophonic treatment. All algorithms are accessible through the same API making integration as easy as possible.

élastique Pro V2.0 offers additional APIs for the élastique Pro and efficient modes. These APIs offer better realtime performance and equidistant stretch and pitch factor control. Please refer to the additional documentation.

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

zplane has put more than two years of research into the development of élastique Pro. The new algorithm is based on state-of-the-art psychoacoustic knowledge and sophisticated signal processing theory. With its completely new approach to time-stretching, élastique Pro makes stretching artifacts obsolete and provides sharp transients and crystal clear vocals.

Naturally, élastique Pro offers perfectly stable timing and sample accurate stretching.

The structure of this document is as following: First the API of the élastique Pro library is described. The API documentation contains naming conventions, function descriptions of 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.

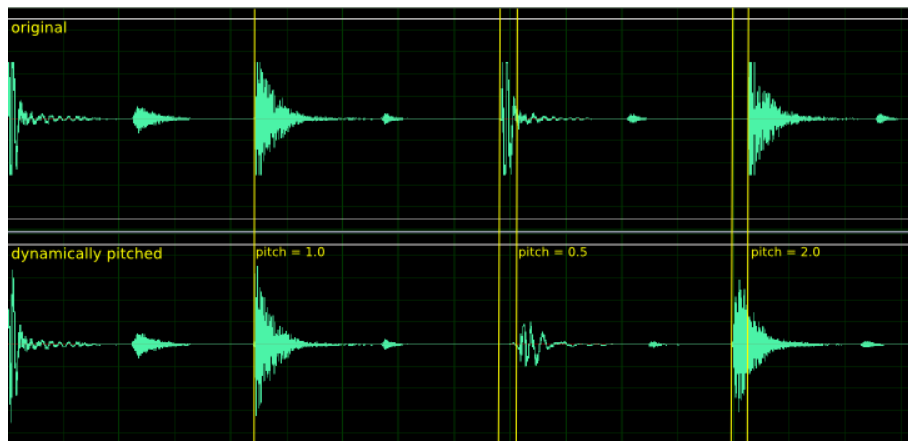


Figure 1: Original and pitched audio without sync

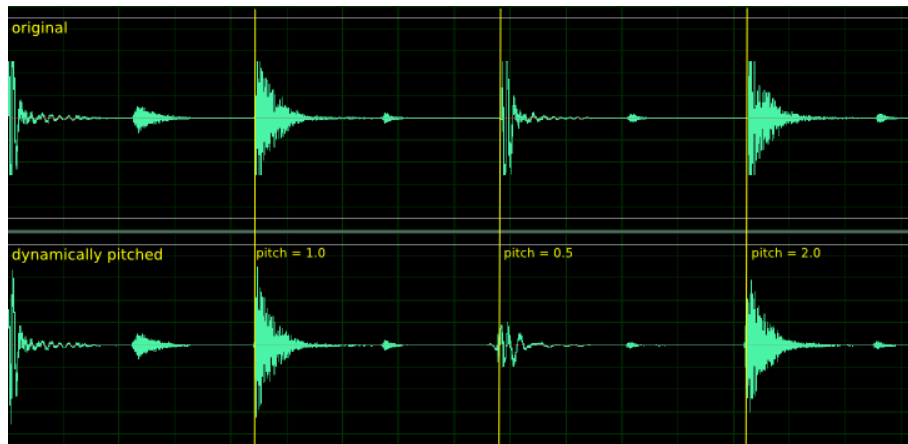


Figure 2: Original and pitched audio with sync

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

The standard C++-API of `élastique Pro` can be accessed via the file [elastiqueProAPI.h](#), where the class `CelastiqueProIf` provides the interface for `élastique Pro`.

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

1.3.1 Memory Allocation

The `élastique Pro` SDK does not allocate any buffers handled by the calling application. Therefore, 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 `CelastiqueProIf::CreateInstance(.)` (C++-API, see [C++ API description](#) for details). The maximum size of the input buffer can be retrieved by the API after instance creation. Please note that this has to be done right after instance creation:

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

The current size of the input buffer can (and should) be requested before each `Process-Call` with the function `CelastiqueProIf::GetFramesNeeded(.)`.

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 and Multichannel-Processing

When the input signal consists of two or more channels, it is strongly recommended to use `élastique` with a stereo/multichannel instance, not with single instances for each channel. This has two reasons: quality and performance. The quality will be better, since the content of all channels is taken into account for the analysis, and the stereo processing is linked between all channels, preserving phase coherence between all channels. The performance will be better since several analysis steps can be combined for both channels.

1.3.4 Processing Modes

`élastique Pro` offers several modes to allow best results in all use cases. The provided modes are defined in `CelastiqueProIf::_elastiquePro_proc_mode`:

- `kProDefaultMode`: this is the mode that should be used as general mode for most kinds of input signals
- `kProTransientMode` (**obsolete**): with Version 2 this mode is obsolete, but for compatibility reasons still in the API
- `kEfficientTransientMode`: this corresponds to the `élastique` time stretching default mode; it is usually about factor 2 less cpu expensive
- `kEfficientTonalMode` (**obsolete**): with Version 2 this mode is obsolete, but for compatibility reasons still in the API
- `kSoloMonophonicMode`: this is the special mode for single-voiced input signals. This mode provides highest output quality for monophonic input signals and allow to do formant-preserving pitch shifting.
- `kSoloSpeechMode`: this mode is optimized for speech signals

1.3.5 C++ API description

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

- **CelastiqueProIf::CreateInstance(.)**
description see below
- **CelastiqueProIf::ProcessData(.)**
description see below
- **CelastiqueProIf::DestroyInstance(.)**
description see below

1.3.5.2 Complete Function Description

Instance Handling Functions

- **int CelastiqueProIf::CreateInstance (CelastiqueProIf*& cCElastique, int iOutputBufferSize, 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 1024 frames. The parameter `iNumOfChannels` contains the number of channels with which `élastique` is instantiated. Possible is a channel number between 1 and 48 channels. `fSampleRate` denotes the input (and output) samplerate and `eMode` chooses the processing mode (see [Processing Modes](#)).

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

The use of this function is required.

- **int CelastiqueProIf::DestroyInstance (CelastiqueProIf*& 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 CelastiqueProIf::SetStretchQPitchFactor (float& fStretchFactor, float fPitchFactor)**

Sets the stretch and pitch factors for the `élastique Pro` 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 product of the stretch factor and the pitch factor must be between 0.1 and 10.0. Note that the parameter `fStretchFactor` is modified during the function call to the quantized value.

The function may be called as often as needed, but must not be called 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 Pro* 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 `CelastiqueProIf::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 CelastiqueProIf::SetStretchPitchQFactor (float fStretchFactor, float& fPitchFactor)`**

This function will do exactly the same as `CelastiqueProIf::SetStretchQPitchFactor (.)`, but quantize the pitch factor instead of the stretch factor. In this case, the parameter `fPitchFactor` is modified during the function call to the quantized value. For a more detailed description, see the description of `CelastiqueProIf::SetStretchQPitchFactor (.)`.

- **`int CelastiqueProIf::GetFramesNeeded ()`**

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

- **`int CelastiqueProIf::GetFramesNeeded (int iOutBufferSize)`**

Sets the output buffer size to a new value and returns the required number of input frames for the upcoming processing block. This function can be called alternatively to `CelastiqueProIf::GetFramesNeeded ()`.

- **`int CelastiqueProIf::GetFramesBuffered ()`**

Returns the number of output frames that have already been calculated and are in the internal output buffer.

- **int CelastiqueProf::ProcessData (float** ppInSampleData, int iNumOfInFrames, float** ppOutSampleData)**

Processes the input data and returns the stretched and pitched 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. All buffers have to be allocated by the calling application. The range of the audio input data is $\pm 1.0F$. The parameter iNumOfInFrames describes the number of input frames. Please make sure that this parameter iNumOfInFrames equals the value returned by the function CelastiqueProf::GetFramesNeeded () to assure the requested output buffer size. If the function fails, the return value is not 0.

The use of this function is required.

- **int CelastiqueProf::FlushBuffer (float** ppfOutSampleData)**

Gets all the remaining internal frames and write them into parameter ppfOutSampleData. Returns the number of written samples.

The use of this function is optional.

- **void CelastiqueProf::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.

- **int CelastiqueProf::SetEnvelopeFactor ()**

Sets a spectral envelope shift factor. If the shift factor is the same as the pitch shift factor formant preserving pitch shifting is performed. Otherwise one may shift the formants (envelope) separately. The envelope shifting is performed before the pitch shifting. That means if you have a factor larger than one the formants are shifted down otherwise up, i.e. it is reciprocal to the pitch factor.

This function is only available in either one of the "Pro" modes.

The use of this function is optional.

- **int CelastiqueProf::SetEnvelopeOrder ()**

Sets the order of the spectral envelope estimation. The default is set to 128 which works fine for most material. If the input audio is really high pitched the order should be lowered (lowest value is 8) otherwise if the input audio is low pitched the value should be raised (up to 512).

As a rule of thumb the order may be set according to highest pitch present in the audio material. 44100 divided by the order must result in the highest pitch. The Order is always normalized to 44100 Hz, so the true sample rate doesn't have to be considered.

This function is only available in either one of the "Pro" modes.

The use of this function is optional.

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

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

```
CElastiqueProIf *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 = CElastiqueProIf::CreateInstance( pcElastiqueHandle,
    _OUTPUT_BUFFER_SIZE,
    iNumOfChannels,
    (float)iSampleRate,
    (CElastiqueProIf::_elastiquePro_proc_mode) iMode);

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

Now, let's see how much memory we have to allocate. This doesn't work for the Solo modes, so in that case we set it to a fixed value.

```
iMaxBufferSize = pcElastiqueHandle->GetMaxFramesNeeded();
if (iMaxBufferSize == -1) // this is true for the Solo modes
    iMaxBufferSize = _INPUT_BUFFER_SIZE;
```

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

Optionally, one can set the envelope shifting for all Pro modes. If the the envelope shift factor is set to the same factor as the the pitch factor formant preserving pitch shifting is performed. The order of envelope estimation is set by default to 128. If one experiences disappearing tones, lower the order value, if on the other hand the result still sounds too "mickey-mousy" raise the value.

In the next step, the processing loop can be executed

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.

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

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

and are at the end of the processing loop:

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

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

1.4 Delivered Files (example project)

1.4.1 File Structure

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

1.4.1.2 Project Files The Workspaces, Projectfiles and or Makefiles can be found in the directory `./build` and its subfolders, where the subfolders names correspond to the project names.

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

1.4.1.4 Include Files Include files can be found in `./incl`.

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

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

1.4.1.7 Binary Files The final executable can be found in the directory `./bin`. In debug-builds, the binary files are in the subfolder `/Debug`.

1.4.1.8 Temporary Files The directory `./tmp` is for all temporary files while building the projects. In debug-builds, the temporary files can be found in the subfolder `/Debug`.

1.5 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`.

1.6 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:

```
elastiqueProCl 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):

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

1.7 Support

Support for the SDK 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 élastique Pro V2.1 SDK Documentation Directory Hierarchy

2.1 élastique Pro V2.1 SDK Documentation Directories

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

[incl](#) 12

3 élastique Pro V2.1 SDK Documentation Hierarchical Index

3.1 élastique Pro V2.1 SDK Documentation Class Hierarchy

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

[CElastiqueProIf](#) 12

4 élastique Pro V2.1 SDK Documentation Class Index

4.1 élastique Pro V2.1 SDK Documentation Class List

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

[CElastiqueProIf](#) 12

5 élastique Pro V2.1 SDK Documentation File Index

5.1 élastique Pro V2.1 SDK Documentation File List

Here is a list of all files with brief descriptions:

[elastiqueProAPI.h](#) (Interface of the [CElastiqueProIf](#) class) 19

6 élastique Pro V2.1 SDK Documentation Directory Documentation

6.1 incl/ Directory Reference

Files

- file [elastiqueProAPI.h](#)
interface of the [CElastiqueProIf](#) class.

7 élastique Pro V2.1 SDK Documentation Class Documentation

7.1 CElastiqueProIf Class Reference

```
#include <elastiqueProAPI.h>
```

7.1.1 Detailed Description

Definition at line 112 of file [elastiqueProAPI.h](#).

Public Types

- enum [_elastiquePro_proc_mode](#) {
[kProDefaultMode](#) = 0, [kProTransientMode](#), [kEfficientTransientMode](#), [kEfficientTonalMode](#),
[kSoloMonophonicMode](#), [kSoloSpeechMode](#) }
- enum [_ElastiqueProStereoInputMode_](#) { [kPlainStereoMode](#) = 0, [kMSMode](#) }

Public Member Functions

- virtual [~CElastiqueProIf](#) ()

- 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](#) ([_ElastiqueProStereoInputMode_](#) eStereoInputMode)=0
- virtual int [SetEnvelopeFactor](#) (float fShiftFactor)=0
- virtual int [SetEnvelopeOrder](#) (int iOrder)=0
- virtual double [GetCurrentTimePos](#) ()=0

Static Public Member Functions

- static char * [GetVersionString](#) ()
- static char * [GetBuildDateString](#) ()
- static int [CreateInstance](#) ([CElastiqueProIf](#) *&cCElastique, int iOutputBufferSize, int iNumOfChannels, float fSampleRate, [_elastiquePro_proc_mode](#) eMode)
- static int [DestroyInstance](#) ([CElastiqueProIf](#) *&cCElastique)

7.1.2 Member Enumeration Documentation

7.1.2.1 enum [CElastiqueProIf::_elastiquePro_proc_mode](#)

Enumerator:

kProDefaultMode pro mode optimized for audio rich of tonal components, usually recommended

kProTransientMode DEPRECATED: pro mode optimized for audio rich of transients, not necessary anymore, it's only there for legacy reasons.

kEfficientTransientMode classic mode optimized for audio rich of transients, usually recommended

kEfficientTonalMode DEPRECATED: classic mode optimized for audio rich of tonal components, not necessary anymore, it's only there for legacy reasons.

kSoloMonophonicMode SOLOIST mode optimized for monophonic audio with formant preserving pitch shifting.

kSoloSpeechMode SOLOIST mode optimized for single speech audio.

Definition at line 118 of file [elastiqueProAPI.h](#).

7.1.2.2 enum CElastiqueProIf::_ElastiqueProStereoInputMode_

Enumerator:

kPlainStereoMode normal LR stereo mode

kMSMode MS stereo mode M must be in channel 0 and S in channel 1.

Definition at line 128 of file elastiqueProAPI.h.

7.1.3 Constructor & Destructor Documentation

7.1.3.1 virtual CElastiqueProIf::~CElastiqueProIf () [inline, virtual]

Definition at line 116 of file elastiqueProAPI.h.

7.1.4 Member Function Documentation

7.1.4.1 static int CElastiqueProIf::CreateInstance (CElastiqueProIf *& c-CElastique, int iOutputBufferSize, int iNumOfChannels, float fSampleRate, _elastiquePro_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, shall not exceed 512 and not be smaller than 128

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

7.1.4.2 static int CElastiqueProIf::DestroyInstance (CElastiqueProIf *& c-CElastique) [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

7.1.4.3 virtual int CElastiqueProIf::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 number of frames returned

7.1.4.4 static char* CElastiqueProIf::GetBuildDateString () [static]

returns the build date as string

Returns:

static char* : returns a pointer to build date string

7.1.4.5 virtual double CElastiqueProIf::GetCurrentTimePos () [pure virtual]

returns the current input time position

Returns:

returns the current input time position

7.1.4.6 virtual int CElastiqueProIf::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

7.1.4.7 virtual int CElastiqueProIf::GetFramesNeeded () [pure virtual]

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

Parameters:

none

Returns:

virtual int : returns the number of frames required

7.1.4.8 virtual int CElastiqueProIf::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 [CElastiqueProIf::ProcessData\(..\)](#) this function changes the internal output buffer size as specified when calling [CElastiqueProIf::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

7.1.4.9 virtual int CElastiqueProIf::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

7.1.4.10 static char* CElastiqueProIf::GetVersionString () [static]

returns the version as string

Returns:

static char* : returns a pointer to version string

7.1.4.11 virtual int CElastiqueProIf::ProcessData (float ** *ppInSampleData*, int *iNumOfInFrames*, float ** *ppOutSampleData*) [pure virtual]

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

7.1.4.12 virtual void CElastiqueProIf::Reset () [pure virtual]

resets the internal state of the élastique algorithm

Parameters:

none

Returns:

int : returns some error code otherwise NULL

7.1.4.13 virtual int CElastiqueProIf::SetEnvelopeFactor (float fShiftFactor)

[pure virtual]

Sets a spectral envelope shift factor. If the shift factor is the same as the pitch shift factor formant preserving pitch shifting is performed. Otherwise one may shift the formants (envelope) separately. The envelope shifting is performed before the pitch shifting. That means if you have a factor larger than one the formants are shifted down otherwise up, i.e. it is reciprocal to the pitch factor. This is only available in either one of the "Pro" modes.

Parameters:

fShiftFactor The envelope shift factor.

Returns:

returns some error code otherwise NULL

7.1.4.14 virtual int CElastiqueProIf::SetEnvelopeOrder (int iOrder) [pure

virtual]

Sets the order of the spectral envelope estimation. The default is set to 128 which works fine for most material. If the input audio is really high pitched the order should be lowered (lowest value is 8) otherwise if the input audio is low pitched the value should be raised (up to 512). This is only available in either one of the "Pro" modes.

Parameters:

iOrder Sets the order of the spectral envelope estimation

Returns:

returns some error code otherwise NULL

7.1.4.15 virtual int CElastiqueProIf::SetStereoInputMode (_ElastiqueProStereoInputMode_ 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

7.1.4.16 virtual int CElastiqueProIf::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

7.1.4.17 virtual int CElastiqueProIf::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:

- [elastiqueProAPI.h](#)

8 élastique Pro V2.1 SDK Documentation File Documentation

8.1 docugen.txt File Reference

8.2 élastiqueProAPI.h File Reference

8.2.1 Detailed Description

interface of the [CElastiqueProIf](#) class.

:

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

Classes

- class [CElastiqueProIf](#)

Defines

- #define [__libElastiqueProIf_HEADER_INCLUDED__](#)

8.2.2 Define Documentation

8.2.2.1 #define [__libElastiqueProIf_HEADER_INCLUDED__](#)

Definition at line 109 of file [élastiqueProAPI.h](#).

Index

- ~CElastiqueProIf
 - CElastiqueProIf, [14](#)
 - _ElastiqueProStereoInputMode_
 - CElastiqueProIf, [13](#)
 - __libElastiqueProIf_HEADER_
INCLUDED__, [19](#)
 - elastiqueProAPI.h, [19](#)
 - _elastiquePro_proc_mode
 - CElastiqueProIf, [13](#)
 - CElastiqueProIf, [12](#)
 - kEfficientTonalMode, [13](#)
 - kEfficientTransientMode, [13](#)
 - kMSMode, [14](#)
 - kPlainStereoMode, [14](#)
 - kProDefaultMode, [13](#)
 - kProTransientMode, [13](#)
 - kSoloMonophonicMode, [13](#)
 - kSoloSpeechMode, [13](#)
 - CElastiqueProIf
 - ~CElastiqueProIf, [14](#)
 - _ElastiqueProStereoInputMode_, [13](#)
 - _elastiquePro_proc_mode, [13](#)
 - CreateInstance, [14](#)
 - DestroyInstance, [14](#)
 - FlushBuffer, [14](#)
 - GetBuildDateString, [15](#)
 - GetCurrentTimePos, [15](#)
 - GetFramesBuffered, [15](#)
 - GetFramesNeeded, [15](#), [16](#)
 - GetMaxFramesNeeded, [16](#)
 - GetVersionString, [16](#)
 - ProcessData, [16](#)
 - Reset, [17](#)
 - SetEnvelopeFactor, [17](#)
 - SetEnvelopeOrder, [17](#)
 - SetStereoInputMode, [17](#)
 - SetStretchPitchQFactor, [18](#)
 - SetStretchQPitchFactor, [18](#)
 - CreateInstance
 - CElastiqueProIf, [14](#)
 - DestroyInstance
 - CElastiqueProIf, [14](#)
 - docugen.txt, [19](#)
 - elastiqueProAPI.h, [19](#)
 - elastiqueProAPI.h
 - __libElastiqueProIf_HEADER_
INCLUDED__, [19](#)
 - FlushBuffer
 - CElastiqueProIf, [14](#)
 - GetBuildDateString
 - CElastiqueProIf, [15](#)
 - GetCurrentTimePos
 - CElastiqueProIf, [15](#)
 - GetFramesBuffered
 - CElastiqueProIf, [15](#)
 - GetFramesNeeded
 - CElastiqueProIf, [15](#), [16](#)
 - GetMaxFramesNeeded
 - CElastiqueProIf, [16](#)
 - GetVersionString
 - CElastiqueProIf, [16](#)
- incl/ Directory Reference, [12](#)
- kEfficientTonalMode
 - CElastiqueProIf, [13](#)
 - kEfficientTransientMode
 - CElastiqueProIf, [13](#)
 - kMSMode
 - CElastiqueProIf, [14](#)
 - kPlainStereoMode
 - CElastiqueProIf, [14](#)
 - kProDefaultMode
 - CElastiqueProIf, [13](#)
 - kProTransientMode
 - CElastiqueProIf, [13](#)
 - kSoloMonophonicMode
 - CElastiqueProIf, [13](#)
 - kSoloSpeechMode
 - CElastiqueProIf, [13](#)
- ProcessData
 - CElastiqueProIf, [16](#)
 - Reset
 - CElastiqueProIf, [17](#)
 - SetEnvelopeFactor
 - CElastiqueProIf, [17](#)
 - SetEnvelopeOrder

CElastiqueProIf, [17](#)
SetStereoInputMode
CElastiqueProIf, [17](#)
SetStretchPitchQFactor
CElastiqueProIf, [18](#)
SetStretchQPitchFactor
CElastiqueProIf, [18](#)