



## SDK Manual

Tim Flohrer

(c) 2008 by zplane.development

August 1, 2008

## Contents

<b>1</b>	<b>élastique Pro V2.1 Direct API SDK Documentation</b>	<b>1</b>
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 . . . . .	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 . . . . .	4
1.4	Delivered Files (example project) . . . . .	10
1.4.1	File Structure . . . . .	10
1.5	Coding Style minimal overview . . . . .	10
1.6	Command Line Usage Example . . . . .	11
1.7	Support . . . . .	11
<b>2</b>	<b>élastique Pro V2.1 Direct API SDK Documentation Directory Hierarchy</b>	<b>11</b>
2.1	élastique Pro V2.1 Direct API SDK Documentation Directories . . . . .	11
<b>3</b>	<b>élastique Pro V2.1 Direct API SDK Documentation Hierarchical Index</b>	<b>12</b>
3.1	élastique Pro V2.1 Direct API SDK Documentation Class Hierarchy . . . . .	12
<b>4</b>	<b>élastique Pro V2.1 Direct API SDK Documentation Class Index</b>	<b>12</b>
4.1	élastique Pro V2.1 Direct API SDK Documentation Class List . . . . .	12
<b>5</b>	<b>élastique Pro V2.1 Direct API SDK Documentation File Index</b>	<b>12</b>
5.1	élastique Pro V2.1 Direct API SDK Documentation File List . . . . .	12
<b>6</b>	<b>élastique Pro V2.1 Direct API SDK Documentation Directory Documenta- tion</b>	<b>12</b>
6.1	incl/ Directory Reference . . . . .	12
<b>7</b>	<b>élastique Pro V2.1 Direct API SDK Documentation Class Documentation</b>	<b>13</b>
7.1	CElastiqueProDirectIf Class Reference . . . . .	13
7.1.1	Detailed Description . . . . .	13

7.1.2	Member Enumeration Documentation . . . . .	14
7.1.3	Constructor & Destructor Documentation . . . . .	14
7.1.4	Member Function Documentation . . . . .	14
<b>8</b>	<b>élastique Pro V2.1 Direct API SDK Documentation File Documentation</b>	<b>21</b>
8.1	docugen_ProDirect.txt File Reference . . . . .	21
8.2	élastiqueProDirectAPI.h File Reference . . . . .	21
8.2.1	Detailed Description . . . . .	21
8.2.2	Define Documentation . . . . .	21

# 1 élastique Pro V2.0 Direct API 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.

With Version 2 of élastiquePro and élastique efficient an alternative API has been introduced for both algorithms. The APIs for Pro and efficient are mostly the same, so this document describes the Pro Direct API but also applies to the efficient Direct API. The only difference is that the Pro API features some methods that are only supported by the élastiquePro timestretch engine (polyphonic formant preserving pitching).

The new Direct API is intended for two use cases:

- Decreasing performance peaks in realtime application especially those with low latency
- Gaining better control over setting parameters at a definite point in time

The API mostly corresponds to the standard API, so the methods that are the same are not described explicitly here, please refer to the standard API description for further information. This document will focus on the methods that differ and on how to use them for the two use cases described above.

One important concept of the Direct API is that the processing is split into smaller parts, giving the developer the opportunity to achieve better distribution of processing over time.

## 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.

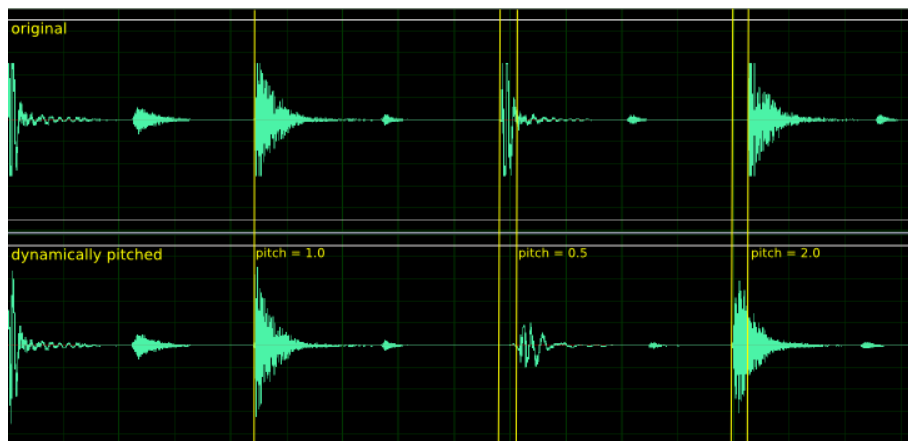


Figure 1: Original and pitched audio without sync

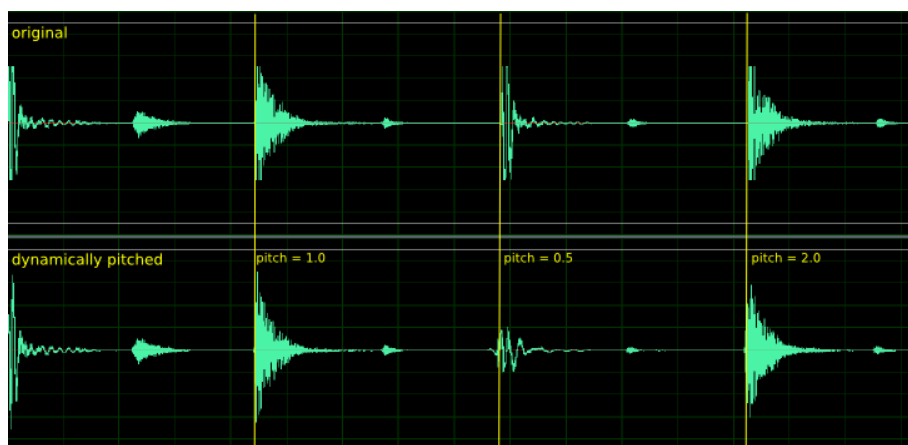


Figure 2: Original and pitched audio with sync

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 Direct C++-API of `élastique Pro` can be accessed via the file `élastiquePro-DirectAPI.h` resp. `élastiqueDirectAPI.h`, where the class `CElastiqueProDirectIf` resp. `CElastiqueDirectIf` provide the interface for Direct API. In the following we will only refer to the `CElastiqueProDirectIf`.

Error codes are defined in `zErrorCodes.h`.

### 1.3.1 Memory Allocation

The `élastique Pro` Direct 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. In contrast to the standard API the Direct API does not allow

the definition of a desired output block size. The output block size will vary with the stretch and pitch factor. On the other hand the input block size will be fixed except for the first call. 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 [CElastiqueProDirectIf::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 [CElastiqueProDirectIf::\\_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

For the *élastique* efficient Direct API it is defined in [CElastiqueDirectIf::\\_elastique\\_proc\\_mode](#)

- `kEfficientTransientMode`: this corresponds to the *élastique* time stretching default mode
- `kEfficientTonalMode` (**obsolete**): with Version 2 this mode is obsolete, but for compatibility reasons still in the API

### 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.

- **`CElastiqueProDirectIf::CreateInstance(.)`**  
description see below
- **`CElastiqueProDirectIf::PreFillData()` (alternative 1)**  
description see below
- **`CElastiqueProDirectIf::PreProcessData()` (alternative 2)**  
description see below
- **`CElastiqueProDirectIf::ProcessData()`**  
description see below
- **`CElastiqueProDirectIf::ProcessData(.)`**  
description see below
- **`CElastiqueProDirectIf::GetProcessedData(.)`**  
description see below
- **`CElastiqueProDirectIf::DestroyInstance(.)`**  
description see standard API

### 1.3.5.2 Functions that differ from the standard API

#### Instance Handling Functions

- **`int CElastiqueProDirectIf::CreateInstance (CElastiqueProDirectIf*& c-CElastique, 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 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](#)). Note that in contrast to the standard API the parameter `output blocksize` is missing, which is not required for the Direct API  
If the function fails, the return value is not 0.  
The use of this function is required.

#### Timestretching and Pitch Shifting Functions

- **`int CElastiqueProDirectIf::GetFramesNeeded ()`**  
see standard API. The main difference to the standard API is, that `GetFramesNeeded` will except for the first call (with alternative 2, see below) always return the same value

- **int CElastiqueProDirectIf::GetPreFramesNeeded ()**  
returns the number of frames to prefill buffers needed either by PreFillData(.) or PreProcessData(.).
- **int CElastiqueProDirectIf::GetNumOfInitialUnusedFrames()**  
returns the number of frames that have to be discarded in the beginning if using PreFillData(.).
- **int CElastiqueProDirectIf::PreFillData (float\*ppInSampleData, int iNumOfInFrames)**  
This function should be used if you want equidistant input hops, giving you the opportunity to have definite points in time to change the stretch and/or pitch parameters. The input data passed in has to have the size as retrieved by GetPreFramesNeeded(.). Also note that a certain amount of frames (as retrieved by GetNumOfInitialUnusedFrames(.)) in the beginning has to be discarded.  
The use of this function is required or alternatively PreProcessData(.) may be used.
- **int CElastiqueProDirectIf::PreProcessData (float\*ppInSampleData, int iNumOfInFrames, float\*ppInSampleData,)**  
This function should be used for better distribution of processing over time. It also returns some number of unprocessed frames in order to start the audio output and use the time for processing the next output data. The function takes care of the overlapping. Please note that the Process call after that will be more time consuming and will need more input data than later on.  
The use of this function is required or alternatively PreFillData(.) may be used.
- **int CElastiqueProDirectIf::GetNumOfProcessCalls ()**  
This function returns how often ProcessData() has to be called in order to complete processing.  
The use of this function is required.
- **int CElastiqueProDirectIf::ProcessData (float\*ppInSampleData, int iNumOfInFrames)**  
Pushes the input data into the algorithm. This is always the first step in processing and has to be followed by calls to Processdata() (no arguments) and finally one call to GetProcessedData(.). The input 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 +-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 CElastiqueProDirectIf::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 CElastiqueProDirectIf::ProcessData ()**

This function does the separate processing steps. Has to be called as often as retrieved by GetNumOfProcessCalls ().

The use of this function is required.

- **int CElastiqueProDirectIf::GetProcessedData (float\*ppInSampleData)**

This functions returns the processed data into the pointer passed and has the number of frames as a return value.

The use of this function is required.

### 1.3.5.3 C++ Usage example for distribution over time using PreProcessData(.)

The complete code can be found in the example source file `elastiqueProDirectTest-CL.cpp`. We focus on the difference to the standard API here, for instance creation etc. please refer to the standard API documentation.

The basic idea of using this API is to get some unprocessed data with a call to `PreProcessData(.)` and use that time to do the processing of the next output data. This decreases the computational overhead when starting output. To illustrate this we use in this example a simple double buffering, so that the previous output is written while the current is processed.

After instance creation and buffer allocation, first the stretch and pitch factor for the first buffer has to be set.

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

Then, we ask the interface for the number of frames needed for the preprocessing.

```
iNumOfInFrames = pcElastiqueHandle->GetPreFramesNeeded();
```

Now, we call the preprocessing and receive a buffer of frames that we keep in order to write out while processing the next buffer.

```
// Keep in mind that the first "real" processing after the PreProcessData call will always be r
// the subsequent processing calls.
iFrameCnt[0] = pcElastiqueHandle->PreProcessData(      apfProcessInputData,
                                                       iNumOfInFrames,
                                                       ppfProcessOutputData1);
```

In the next step, the processing loop can be executed

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

After reading the audio data, the process function can be called with the required number of input frames, this will not return any frames but rather initiate a new process step.

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

The next line tells us how often ProcessData() has to be called. Please note that this has to be saved in a separate variable and not used directly in the for() statement, as the output of that function may vary during the process steps.

```
iCalls = pcElastiqueHandle->GetNumOfProcessCalls();
```

Then we call ProcessData() as often as retrieved by GetNumOfProcessCalls. In real applications this may be used to distribute the processing over time.

```
for (iSteps = 0; iSteps < iCalls; iSteps++)
{
    pcElastiqueHandle->ProcessData();
}
```

Finally we get the processed output data. The function returns the number of frames returned.

```
// 4th step: get the result
iFrameCnt[iCurrentFrame%2] = pcElastiqueHandle->GetProcessedData(ppfProcessOutputData2);
```

after saving the data, we are at the end of the processing loop:

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

After the processing loop was exited the rest of the example basically follows the example of the standard API.

**1.3.5.4 C++ Usage example for equidistant processing using PreFillData(.)** The complete code can be found in the example source file elastiqueProDirectTest-EquidistantCL.cpp. We focus on the difference to the standard API here, for instance creation etc. please refer to the standard API documentation.

The basic idea of using this API is to always have a fixed input buffer size, so that there are predefined points in time to change the stretch and/or pitch factor. This is especially useful for drawing stretch or pitch curves. The fixed input size can be retrieved by GetFramesNeeded(). The only exception of the fixed input size is the first block of input audio data. The size for that is retrieved by GetPreFramesNeeded(). After that PreFillData has to be called. Please note that this kind of operation produces some invalid audio frames that have to be omitted by the application developer. The number of frames to be omitted can be accessed by calling GetNumOfInitialUnusedFrames().

After instance creation and buffer allocation, first the stretch and pitch factor for the first buffer has to be set.

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

To get the number of frames to be omitted call

```
iTotalFrames = -pcElastiqueHandle->GetNumOfInitialUnusedFrames();
```

Then, we ask the interface for the number of frames needed for the preprocessing.

```
iNumOfInFrames = pcElastiqueHandle->GetPreFramesNeeded();
```

Now, we call the prefill function.

```
iFrameCnt = pcElastiqueHandle->PreFillData(    apfProcessInputData,
                                              iNumOfInFrames,
                                              apfProcessOutputData);
```

In the next step, the processing loop can be executed, `GetFramesNeeded` now always returns the same value

```
while (bReadNextFrame)
{
    // check how much frames elastique expects at this run

    iNumOfInFrames = pcElastiqueHandle->GetFramesNeeded();
```

After reading the audio data, the process function can be called with the required number of input frames, this will not return any frames but rather initiate a new process step. In order to change the stretch or pitch factor call `SetStretchPitchQFactor(.)` or `SetStretchQPitchFactor(.)` before:

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

The next line tells us how often `ProcessData()` has to be called. Please note that this has to be saved in a separate variable and not used directly in the `for()` statement, as the output of that function may vary during the process steps.

```
zINT32 iCalls = pcElastiqueHandle->GetNumOfProcessCalls();
```

Then we call `ProcessData()` as often as retrieved by `GetNumOfProcessCalls`. In real applications this may be used to distribute the processing over time.

```
for (iSteps = 0; iSteps < iCalls; iSteps++)
{
    pcElastiqueHandle->ProcessData();
}
```

Finally we get the processed output data. The function returns the number of frames returned.

```
iFrameCnt = pcElastiqueHandle->GetProcessedData(apfProcessOutputData);
```

after saving the data, we are at the end of the processing loop:

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

After the processing loop was exited the rest of the example basically follows the example of the standard API.

## 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](mailto:info@zplane.de)

katzbachstr. 21

D-10965 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: [info@zplane.de](mailto:info@zplane.de)

## 2 élastique Pro V2.1 Direct API SDK Documentation Directory Hierarchy

### 2.1 élastique Pro V2.1 Direct API SDK Documentation Directories

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

**incl**

**12**

## 3 **élastique Pro V2.1 Direct API SDK Documentation Hierarchical Index**

### 3.1 **élastique Pro V2.1 Direct API SDK Documentation Class Hierarchy**

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

[CElastiqueProDirectIf](#) 13

## 4 **élastique Pro V2.1 Direct API SDK Documentation Class Index**

### 4.1 **élastique Pro V2.1 Direct API SDK Documentation Class List**

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

[CElastiqueProDirectIf](#) 13

## 5 **élastique Pro V2.1 Direct API SDK Documentation File Index**

### 5.1 **élastique Pro V2.1 Direct API SDK Documentation File List**

Here is a list of all files with brief descriptions:

[elastiqueProDirectAPI.h](#) (Interface of the [CElastiqueProDirectIf](#) class ) 21

## 6 **élastique Pro V2.1 Direct API SDK Documentation Directory Documentation**

### 6.1 **incl/ Directory Reference**

#### Files

- file [elastiqueProDirectAPI.h](#)  
*interface of the [CElastiqueProDirectIf](#) class.*

## 7 élastique Pro V2.1 Direct API SDK Documentation Class Documentation

### 7.1 CElastiqueProDirectIf Class Reference

```
#include <elastiqueProDirectAPI.h>
```

#### 7.1.1 Detailed Description

CLASS

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

USAGE

Definition at line 118 of file elastiqueProDirectAPI.h.

#### Public Types

- enum [\\_elastiquePro\\_proc\\_mode](#) { [kProDefaultMode](#) = 0, [kProTransientMode](#) }
- enum [\\_ElastiqueProStereoInputMode\\_](#) { [kPlainStereoMode](#) = 0, [kMSMode](#) }

#### Public Member Functions

- virtual [~CElastiqueProDirectIf](#) ()
- virtual int [PreFillData](#) (float \*\*ppInSampleData, int iNumOfInFrames, float \*\*ppOutSampleData)=0
- virtual int [GetNumOfInitialUnusedFrames](#) ()=0
- virtual int [PreProcessData](#) (float \*\*ppInSampleData, int iNumOfInFrames, float \*\*ppOutSampleData)=0
- virtual int [ProcessData](#) (float \*\*ppInSampleData, int iNumOfInFrames)=0
- virtual int [ProcessData](#) ()=0
- virtual int [GetProcessedData](#) (float \*\*ppOutSampleData)=0
- virtual int [GetNumOfProcessCalls](#) ()=0
- virtual int [GetFramesProcessed](#) ()=0
- virtual int [GetPreFramesNeeded](#) ()=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 double [GetCurrentTimePos](#) ()=0
- virtual int [FlushBuffer](#) (float \*\*ppfOutSampleData)=0

- virtual int [SetStereoInputMode](#) ([\\_ElastiqueProStereoInputMode\\_](#) eStereoInputMode)=0
- virtual int [SetEnvelopeFactor](#) (float fShiftFactor)=0
- virtual int [SetEnvelopeOrder](#) (int iOrder)=0

### Static Public Member Functions

- static char \* [GetVersionString](#) ()
- static char \* [GetBuildDateString](#) ()
- static int [CreateInstance](#) ([CElastiqueProDirectIf](#) \*&cCElastique, int iNumOfChannels, float fSampleRate, [\\_elastiquePro\\_proc\\_mode](#) eMode)
- static int [DestroyInstance](#) ([CElastiqueProDirectIf](#) \*&cCElastique)

## 7.1.2 Member Enumeration Documentation

### 7.1.2.1 enum [CElastiqueProDirectIf::\\_elastiquePro\\_proc\\_mode](#)

#### Enumerator:

- *kProDefaultMode* pro mode optimized for audio rich of tonal components, usually recommended
- *kProTransientMode* pro mode optimized for audio rich of transients, not necessary anymore, it's only there for legacy reasons

Definition at line 124 of file [elastiqueProDirectAPI.h](#).

### 7.1.2.2 enum [CElastiqueProDirectIf::\\_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 130 of file [elastiqueProDirectAPI.h](#).

## 7.1.3 Constructor & Destructor Documentation

### 7.1.3.1 virtual [CElastiqueProDirectIf::~CElastiqueProDirectIf](#) () [[inline](#), [virtual](#)]

Definition at line 122 of file [elastiqueProDirectAPI.h](#).

## 7.1.4 Member Function Documentation

### 7.1.4.1 static int [CElastiqueProDirectIf::CreateInstance](#) ([CElastiqueProDirectIf](#) \*&cCElastique, 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  
*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 CElastiqueProDirectIf::DestroyInstance (CElastiqueProDirectIf \*& 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

**7.1.4.3 virtual int CElastiqueProDirectIf::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\* CElastiqueProDirectIf::GetBuildDateString () [static]**

returns the build date as string

**Returns:**

static char\* : returns a pointer to build date string

**7.1.4.5 virtual double CElastiqueProDirectIf::GetCurrentTimePos () [pure virtual]**

returns the current input time position

**Returns:**

int : returns the current input time position

**7.1.4.6 virtual int CElastiqueProDirectIf::GetFramesNeeded ()** [pure virtual]

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

**Parameters:**

*none*

**Returns:**

int : returns the number of frames required

**7.1.4.7 virtual int CElastiqueProDirectIf::GetFramesProcessed ()** [pure virtual]

returns the number of frames that will be put out by the ProcessData function

**Parameters:**

*none*

**Returns:**

int : returns the number of frames required for the output buffer

**7.1.4.8 virtual int CElastiqueProDirectIf::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

**7.1.4.9 virtual int CElastiqueProDirectIf::GetNumOfInitialUnusedFrames ()** [pure virtual]

when using [PreFillData\(\)](#) for equidistant processing this returns the number of frames that have to be omitted in the beginning

**Returns:**

virtual int : returns the number of frames to be ommitted

**7.1.4.10 virtual int CElastiqueProDirectIf::GetNumOfProcessCalls ()** [pure virtual]

returns the number of process calls needed

**Parameters:**

*none*

**Returns:**

virtual int : returns the number of process calls required

**7.1.4.11 virtual int CElastiqueProDirectIf::GetPreFramesNeeded ()** [pure virtual]

returns the number of frames needed for the first pre-processing step, this function should be always called directly before CElastiqueDirectIf::PreProcessData(..)

**Parameters:**

*none*

**Returns:**

int : returns the number of frames required

**7.1.4.12 virtual int CElastiqueProDirectIf::GetProcessedData (float \*\* ppOutSampleData)** [pure virtual]

after processing cal that function to retrieve the processed data

**Parameters:**

*ppOutSampleData* :

**Returns:**

virtual int : returns the number of frames returned

**7.1.4.13 static char\* CElastiqueProDirectIf::GetVersionString ()** [static]

returns the version as string

**Returns:**

static char\* : returns a pointer to version string

**7.1.4.14 virtual int CElastiqueProDirectIf::PreFillData (float \*\* ppInSampleData, int iNumOfInFrames, float \*\* ppOutSampleData) [pure virtual]**

does the initial buffer filling if the number of frames provided is as retrieved by CElastiqueDirectIf::GetPreFramesNeeded() returns first output buffer should be used for equidistant stretch/pitch factor setting

**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 number of processed frames

**7.1.4.15 virtual int CElastiqueProDirectIf::PreProcessData (float \*\* ppInSampleData, int iNumOfInFrames, float \*\* ppOutSampleData) [pure virtual]**

does the initial buffer filling if the number of frames provided is as retrieved by CElastiqueDirectIf::GetPreFramesNeeded() immediately returns some unprocessed frames to give some time to do the first processing, overlapping is done internally should be used for distributing processing over time

**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 number of processed frames

**7.1.4.16 virtual int CElastiqueProDirectIf::ProcessData () [pure virtual]**

call as often as [GetNumOfProcessCalls\(\)](#) tells you

**Parameters:**

*none*

**Returns:**

virtual int : returns the error code, otherwise 0

**7.1.4.17 virtual int CElastiqueProDirectIf::ProcessData (float \*\* *ppInSampleData*, int *iNumOfInFrames*)** [pure virtual]

does the actual processing if the number of frames provided is as retrieved by CElastiqueDirectIf::GetFramesNeeded()

**Parameters:**

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

*iNumOfInFrames* : the number of input frames

**Returns:**

virtual int : returns the error code, otherwise 0

**7.1.4.18 virtual void CElastiqueProDirectIf::Reset ()** [pure virtual]

resets the internal state of the élastique algorithm

**Parameters:**

*none*

**Returns:**

int : returns some error code otherwise NULL

**7.1.4.19 virtual int CElastiqueProDirectIf::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.

@return int: returns some error code otherwise NULL

**7.1.4.20 virtual int CElastiqueProDirectIf::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:**

int: returns some error code otherwise NULL

**7.1.4.21** `virtual int CElastiqueProDirectIf::SetStereoInputMode (\_Elastique-ProStereoInputMode\_eStereoInputMode)` [pure virtual]

sets the stereo input mode

**Parameters:**

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

**Returns:**

int : returns some error code otherwise NULL

**7.1.4.22** `virtual int CElastiqueProDirectIf::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)

**Returns:**

int : returns the error code, otherwise 0

**7.1.4.23** `virtual int CElastiqueProDirectIf::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)

**Returns:**

int : returns the error code, otherwise 0

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

- [elastiqueProDirectAPI.h](#)

## 8 élastique Pro V2.1 Direct API SDK Documentation File Documentation

### 8.1 docugen\_ProDirect.txt File Reference

### 8.2 elastiqueProDirectAPI.h File Reference

#### 8.2.1 Detailed Description

interface of the [CElastiqueProDirectIf](#) class.

:

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

#### Classes

- class [CElastiqueProDirectIf](#)

#### Defines

- `#define __libElastiqueProDirectIf_HEADER_INCLUDED__`

#### 8.2.2 Define Documentation

##### 8.2.2.1 `#define __libElastiqueProDirectIf_HEADER_INCLUDED__`

Definition at line 105 of file [elastiqueProDirectAPI.h](#).

## Index

- ~CElastiqueProDirectIf
    - CElastiqueProDirectIf, 14
  - \_ElastiqueProStereoInputMode\_
    - CElastiqueProDirectIf, 14
  - \_\_libElastiqueProDirectIf\_HEADER\_-  
INCLUDED\_
    - elastiqueProDirectAPI.h, 21
  - \_elastiquePro\_proc\_mode
    - CElastiqueProDirectIf, 14
  - CElastiqueProDirectIf, 13
    - kMSMode, 14
    - kPlainStereoMode, 14
    - kProDefaultMode, 14
    - kProTransientMode, 14
  - CElastiqueProDirectIf
    - ~CElastiqueProDirectIf, 14
    - \_ElastiqueProStereoInputMode\_, 14
    - \_elastiquePro\_proc\_mode, 14
    - CreateInstance, 14
    - DestroyInstance, 15
    - FlushBuffer, 15
    - GetBuildDateString, 15
    - GetCurrentTimePos, 15
    - GetFramesNeeded, 15
    - GetFramesProcessed, 16
    - GetMaxFramesNeeded, 16
    - GetNumOfInitialUnusedFrames, 16
    - GetNumOfProcessCalls, 16
    - GetPreFramesNeeded, 17
    - GetProcessedData, 17
    - GetVersionString, 17
    - PreFillData, 17
    - PreProcessData, 18
    - ProcessData, 18
    - Reset, 19
    - SetEnvelopeFactor, 19
    - SetEnvelopeOrder, 19
    - SetStereoInputMode, 20
    - SetStretchPitchQFactor, 20
    - SetStretchQPitchFactor, 20
  - CreateInstance
    - CElastiqueProDirectIf, 14
  - DestroyInstance
    - CElastiqueProDirectIf, 15
  - docugen\_ProDirect.txt, 21
  - elastiqueProDirectAPI.h, 21
  - elastiqueProDirectAPI.h
    - \_\_libElastiqueProDirectIf\_-  
HEADER\_INCLUDED\_\_, 21
  - FlushBuffer
    - CElastiqueProDirectIf, 15
  - GetBuildDateString
    - CElastiqueProDirectIf, 15
  - GetCurrentTimePos
    - CElastiqueProDirectIf, 15
  - GetFramesNeeded
    - CElastiqueProDirectIf, 15
  - GetFramesProcessed
    - CElastiqueProDirectIf, 16
  - GetMaxFramesNeeded
    - CElastiqueProDirectIf, 16
  - GetNumOfInitialUnusedFrames
    - CElastiqueProDirectIf, 16
  - GetNumOfProcessCalls
    - CElastiqueProDirectIf, 16
  - GetPreFramesNeeded
    - CElastiqueProDirectIf, 17
  - GetProcessedData
    - CElastiqueProDirectIf, 17
  - GetVersionString
    - CElastiqueProDirectIf, 17
- incl/ Directory Reference, 12
- kMSMode
    - CElastiqueProDirectIf, 14
  - kPlainStereoMode
    - CElastiqueProDirectIf, 14
  - kProDefaultMode
    - CElastiqueProDirectIf, 14
  - kProTransientMode
    - CElastiqueProDirectIf, 14
  - PreFillData
    - CElastiqueProDirectIf, 17
  - PreProcessData
    - CElastiqueProDirectIf, 18
  - ProcessData
    - CElastiqueProDirectIf, 18

- Reset
  - [CElastiqueProDirectIf](#), [19](#)
- SetEnvelopeFactor
  - [CElastiqueProDirectIf](#), [19](#)
- SetEnvelopeOrder
  - [CElastiqueProDirectIf](#), [19](#)
- SetStereoInputMode
  - [CElastiqueProDirectIf](#), [20](#)
- SetStretchPitchQFactor
  - [CElastiqueProDirectIf](#), [20](#)
- SetStretchQPitchFactor
  - [CElastiqueProDirectIf](#), [20](#)