



ELASTIQUE PRO time stretching SDK DirectAPI 3.2.5

by zplane.development

(c) 2018 zplane.development GmbH & Co. KG

February 9, 2018

Contents

1	elastique Pro DirectAPI 3.x Documentation	2
1.1	Introduction	2
1.2	What's new in V3	2
1.2.1	What's new in V3.0.4	2
1.2.2	What's new in V3.2.1	2
1.3	DirectAPI operation explained	3
1.3.1	PreProcessData and avoiding performance peaks	3
1.3.2	PreFillData and equally sized input hops	4
1.4	API Documentation	4
1.4.1	Memory Allocation	5
1.4.2	Naming Conventions	5
1.4.3	Stereo Processing	5
1.4.4	C++ API description	5
1.5	Command Line Usage Example	11
1.6	Support	11
2	Class Index	11
2.1	Class List	11
3	File Index	11
3.1	File List	11
4	Class Documentation	12
4.1	CElastiqueProV3DirectIf Class Reference	12
4.1.1	Detailed Description	13
4.1.2	Member Enumeration Documentation	13
4.1.3	Constructor & Destructor Documentation	14
4.1.4	Member Function Documentation	14
5	File Documentation	22
5.1	docugen_ProDirect.txt File Reference	22
5.2	elastiqueProV3DirectAPI.h File Reference	22

1 elastique Pro DirectAPI 3.x Documentation

1.1 Introduction

elastique is one of the most used time stretching and pitch shifting algorithm in the market. elastique is able to run in realtime which makes it the perfect solution for DJing, performance and music production.

The Direct API is an alternative API for elastique and 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 V3

- Highly improved transient preservation engine working even at high stretch ratios
- `infiniStretch` technology to allow arbitrary high stretch ratios up to infinity
- `Hold` function to do infinity stretching
- better memory footprint

1.2.1 What's new in V3.0.4

This version of elastique Pro offers two new methods for warping. See

- [CElastiqueProV3DirectIf::SetNextSyncPoint\(.\)](#)
- [CElastiqueProV3DirectIf::IsReadyForNextSyncPoint\(.\)](#)

and the changelog for details.

1.2.2 What's new in V3.2.1

This version of elastique Pro offers this added functionality. See

- improved performance at `CElastiqueProV3DirectIf::PreProcessData(.)`
- additional mode for `CElastiqueProV3DirectIf::SetNextSyncPoint(.)`

and the changelog for details.

1.3 DirectAPI operation explained

The following sections explain the two operation mode of the DirectAPI.

1.3.1 PreProcessData and avoiding performance peaks

Using `CElastiqueProV3DirectIf::PreProcessData(.)` is targeted at reducing the peak loads that occur when working with low latency buffer sizes in the audio callback. Due to the internal buffer sizes of elastique it may happen that for each output buffer of elastique one may need several audio callbacks thus resulting in a performance peak each time elastique generates a new output buffer. To overcome this problem `PreProcessData` will produce an output buffer with almost no calculation overhead. Now, the time while this output buffer is being sent to the soundcard can be used to generate the next split output buffer. In order to have further control of the performance the processing is split into multiple parts so that the processing can be better distributed over time. The following figure shows that process.

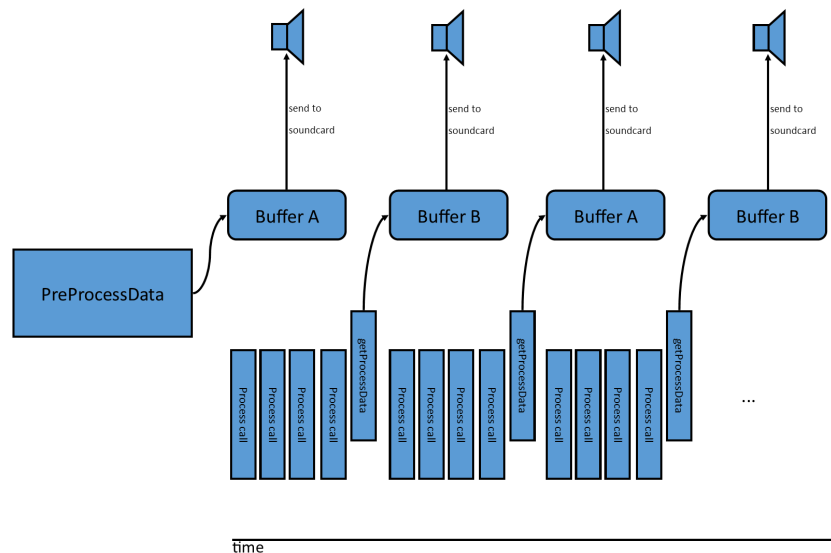


Figure 1: `PreProcessData` illustrated

Refer to section [C++ Usage example for using PreProcessData\(.\)](#) for further information.

1.3.2 PreFillData and equally sized input hops

Using `CElasticProV3DirectIf::PreFillData(.)` has the advantage of giving a direct relationship from input to output buffers. So, each input buffer of a fixed size generates an output buffer of the input size times the stretch factor. This provides a better control of timing. It is important to note that this will produce some initial unused frames that have to be discarded. The following figure shows that process.

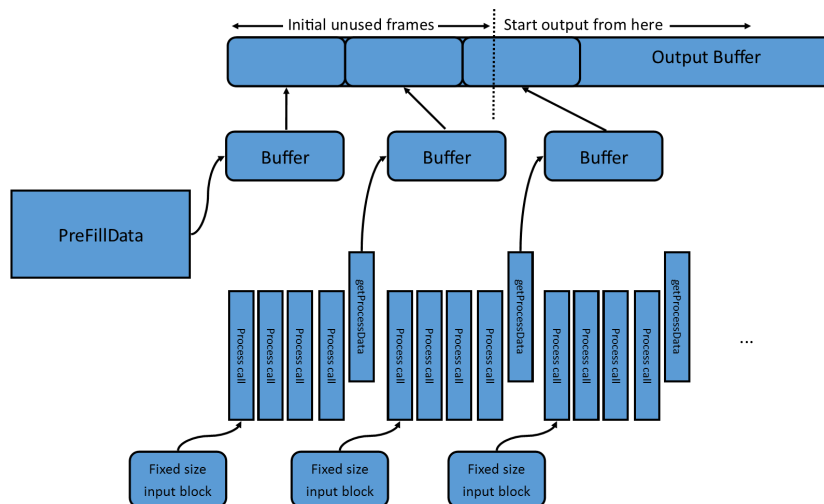


Figure 2: PreFillData illustrated

Refer to section [C++ Usage example for equidistant processing using PreFillData\(.\)](#) for further information.

1.4 API Documentation

elasticque offers two different APIs. The normal and easy to use API can be accessed via the file `elasticqueV3API.h`, where the class `CElasticProV3DirectIf` provides the interface for elasticque.

This alternative API offers the possibility to split up the processing calls in order to avoid performance peaks. The downside of that is that a lot of buffer handling is left to user.

All variable types needed are either defined in file `elastiqueV3API.h` or standard C++ types.

1.4.1 Memory Allocation

The elastique 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 `CElastiqueProV3DirectIf::CreateInstance` (.). The maximum size of the input buffer in frames depends on the output buffer size via the formula:

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

1.4.2 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.

1.4.3 Stereo Processing

When processing stereo input, it is strongly recommended to use elastique 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.

1.4.4 C++ API description

1.4.4.1 Required Functions

The following functions have to be called when using the elastique library. Description see below.

- `CElastiqueProV3DirectIf::CreateInstance`(.)
description see below
- `CElastiqueProV3DirectIf::PreFillData`() (alternative 1)
description see below
- `CElastiqueProV3DirectIf::PreProcessData`() (alternative 2)
description see below
- `CElastiqueProV3DirectIf::ProcessData`()
description see below
- `CElastiqueProV3DirectIf::ProcessData`(.)
description see below

- **CElastiqueProV3DirectIf::GetProcessedData(.)**
description see below
- **CElastiqueProV3DirectIf::DestroyInstance(.)**
description see standard API

1.4.4.2 Functions that differ from the standard API

1.4.4.2.1 Instance Handling Functions

- **int CElastiqueProV3DirectIf::CreateInstance (CElastiqueProV3DirectIf*& cCElastique, int iNumOfChannels, float fSampleRate, ElastiqueMode_t eElastiqueMode = kV3, float fMinCombinedFactor = 0.1f)**

Creates a new instance of the \tilde{A} l'astique time stretcher. The handle to the new instance is returned in parameter *cCElastique. The parameter iNumOfChannels contains the number of channels with which \tilde{A} l'astique 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 modes). Note that in contrast to the standard API the parameter output blocksize is missing, which is not required for the Direct API. The processing mode is either kV3 which should be used as default or kV3mobile which should be used for devices with less computational resources (e.g. mobile devices, thus the name). The fMinCombinedFactor defines the minimum product of pitch and Stretch factor to be used and determines the memory allocated by elastique.

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

The use of this function is required.

1.4.4.2.2 Timestretching and Pitch Shifting Functions

- **int CElastiqueProV3DirectIf::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 CElastiqueProV3DirectIf::GetPreFramesNeeded ()**
returns the number of frames to prefill buffers needed either by PreFillData(.) or PreProcessData(.)
- **int CElastiqueProV3DirectIf::GetNumOfInitialUnusedFrames()**
returns the number of frames that have to be discarded in the beginning if using PreFillData(.)
- **int CElastiqueProV3DirectIf::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 `CElastiqueProV3DirectIf::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 `CElastiqueProV3DirectIf::GetNumOfProcessCalls` ()**

This function returns how often `ProcessData()` has to be called in order to complete processing.

The use of this function is required.

- **int `CElastiqueProV3DirectIf::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 $\pm 1.0F$. The parameter `iNumOfInFrames` describes the number of input frames. Please make sure that this parameter `iNumOfInFrames` equals the value returned by the function `CElastiqueProV3DirectIf::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 `CElastiqueProV3DirectIf::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 `CElastiqueProV3DirectIf::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.

1.4.4.3 C++ Usage example for using `PreProcessData(.)`

The complete code can be found in the example source file `elastiqueProDirectTestCL-V3.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,
true) != 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 more computationally intensive than
// the subsequent processing calls.
iFrameCnt[0] = pcElastiqueHandle->PreProcessData(apfProcessInputData,
                                                iNumOfInFrames,
                                                ppfProcessOutputData1,
false);
```

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 into another buffer. The function returns the number of frames returned.

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

after saving the data and switching the buffers 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.

1.4.4.4 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,
```

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. (In order to change the stretch or pitch factor call `SetStretchPitchQFactor(.)` or `SetStretchQPitchFactor(.)` before `GetFramesNeeded(.)`)

```
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, iNumOfInFrames);
```

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.

```
int 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.

1.5 Command Line Usage Example

The compiled example is a command line application that reads and writes audio files in WAV or AIFF format.

The command line synopsis is:

```
elastiqueProTestCL -i input_file -o output_file -s StretchRatio -p PitchRatio
```

The following command line will result in an output file of the same format as the input file (WAV PCM, 16bit, 48kHz, stereo interleaved, name: input.wav) 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):

```
elastiqueProTestCL -i input48.wav -o output48.wav -s 1.1 -p 0.99
```

1.6 Support

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

[zplane.development GmbH & Co KG](#)

tim flohrer

grunewaldstr. 83

D-10823 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: flohrer@zplane.de

2 Class Index

2.1 Class List

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

[CElastiqueProV3DirectIf](#) 12

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

[elastiqueProV3DirectAPI.h](#) 22

4 Class Documentation

4.1 CElastiqueProV3DirectIf Class Reference

```
#include <elastiqueProV3DirectAPI.h>
```

Public Types

- enum [ElastiqueStereoInputMode_t](#) { [kPlainStereoMode](#) = 0, [kMSMode](#) }
- enum [ElastiqueVersion_t](#) { [kV3Pro](#) = 0, [kV3Eff](#), [kV3mobile](#) }
- enum [Version_t](#) { [kMajor](#), [kMinor](#), [kPatch](#), [kRevision](#), [kNumVersionInts](#) }

Public Member Functions

- virtual [~CElastiqueProV3DirectIf](#) ()
- virtual int [PreFillData](#) (float **ppInSampleData, int iNumOfInFrames, float **ppOutSampleData)=0
- virtual int [GetNumOfInitialUnusedFrames](#) ()=0
- virtual int [PreProcessData](#) (float **ppInSampleData, int iNumOfInFrames, float **ppOutSampleData, bool bDisableStartupPerfOptimization=false)=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](#) ()=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 long long [GetCurrentTimePos](#) ()=0
- virtual int [FlushBuffer](#) (float **ppfOutSampleData)=0
- virtual int [SetStereoInputMode](#) ([CElastiqueProV3DirectIf::ElastiqueStereoInputMode_t](#) eStereoInputMode)=0
- virtual int [SetEnvelopeFactor](#) (float fShiftFactor)=0
- virtual int [SetEnvelopeOrder](#) (int iOrder)=0
- virtual int [SetHold](#) (bool bSetHold, bool bKeepTime=false)=0
- virtual int [SetNextSyncPoint](#) (int iInputTimeInSamples, int iOutputTimeInSamples, float &fPitchFactor, bool bAssumeTransientAtSyncPoint=true)=0
- virtual bool [IsReadyForNextSyncPoint](#) () const =0

Static Public Member Functions

- static const int [GetVersion](#) (const [Version_t](#) eVersionIdx)
- static const char * [GetBuildDate](#) ()
- static int [CreateInstance](#) ([CElastiqueProV3DirectIf](#) *&cCElastique, int iNumOfChannels, float fSampleRate, [ElastiqueVersion_t](#) eElastiqueVersion=[kV3Pro](#), float fMinCombinedFactor=0.1f)
- static int [DestroyInstance](#) ([CElastiqueProV3DirectIf](#) *cCElastique)

4.1.1 Detailed Description

Definition at line 60 of file [elastiqueProV3DirectAPI.h](#).

4.1.2 Member Enumeration Documentation

4.1.2.1 enum [CElastiqueProV3DirectIf::ElastiqueStereoInputMode_t](#)

Enumerator:

kPlainStereoMode normal LR stereo mode

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

Definition at line 66 of file [elastiqueProV3DirectAPI.h](#).

```
{
    kPlainStereoMode = 0,
    kMSMode
};
```

4.1.2.2 enum [CElastiqueProV3DirectIf::ElastiqueVersion_t](#)

Enumerator:

kV3Pro

kV3Eff

kV3mobile

Definition at line 72 of file [elastiqueProV3DirectAPI.h](#).

```
{
    kV3Pro = 0,
    kV3Eff,
    kV3mobile
};
```

4.1.2.3 enum CElastiqueProV3DirectIf::Version_t

Enumerator:

kMajor
kMinor
kPatch
kRevision
kNumVersionInts

Definition at line 79 of file `elastiqueProV3DirectAPI.h`.

```
{
    kMajor,
    kMinor,
    kPatch,
    kRevision,

    kNumVersionInts
};
```

4.1.3 Constructor & Destructor Documentation

4.1.3.1 virtual CElastiqueProV3DirectIf::~CElastiqueProV3DirectIf ()
[inline, virtual]

Definition at line 64 of file `elastiqueProV3DirectAPI.h`.

```
{};
```

4.1.4 Member Function Documentation

4.1.4.1 static int CElastiqueProV3DirectIf::CreateInstance (CElastiqueProV3DirectIf *& cElastique, int iNumOfChannels, float fSampleRate, ElastiqueVersion_t eElastiqueVersion = kV3Pro, float fMinCombinedFactor = 0.1f) [static]

creates an instance of zplane's `élastique` class

Parameters

<i>cElastique</i>	: returns a pointer to the class instance
<i>iNumOfChannels</i>	: number of channels
<i>fSampleRate</i>	: input samplerate
<i>fMinCombinedFactor</i> :	minimum combined factor to be used: the combined minimum is <code>minstretch*minpitch</code>

Returns

static int : returns some error code otherwise NULL

4.1.4.2 static int CElastiqueProV3DirectIf::DestroyInstance (CElastiqueProV3DirectIf * *cCElastique*) [static]

destroys an instance of the zplane's élastique class

Parameters

<i>cCElastique</i>	: pointer to the instance to be destroyed
--------------------	---

Returns

static int : returns some error code otherwise NULL

4.1.4.3 virtual int CElastiqueProV3DirectIf::FlushBuffer (float ** *ppfOutSampleData*) [pure virtual]

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

Parameters

<i>ppfOut-Sample-Data,:</i>	double pointer to the output buffer of samples [channels][samples]
-----------------------------	--

Returns

int : returns some error code otherwise number of frames returned

4.1.4.4 static const char* CElastiqueProV3DirectIf::GetBuildDate () [static]

4.1.4.5 virtual long long CElastiqueProV3DirectIf::GetCurrentTimePos () [pure virtual]

returns the current input time position

Returns

returns the current input time position

4.1.4.6 virtual int CElastiqueProV3DirectIf::GetFramesNeeded () [pure virtual]

returns the number of frames needed for the next processing step, this function should be always called directly before CElastiqueProDirectIf::ProcessData(..) when using -PreFillData this function always returns the same value in Pro Mode, in Efficient mode

the value may change with the stretch factor but only in terms of powers of 2. The only exception is when using `infiniStretch` then it may return 0. when using `PreProcessData` the first call will be different but after that it will behave like with `PreFillData`

Returns

int : returns the number of frames required

4.1.4.7 virtual int CElastiqueProV3DirectIf::GetFramesProcessed () [pure virtual]

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

Returns

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

4.1.4.8 virtual int CElastiqueProV3DirectIf::GetMaxFramesNeeded () [pure virtual]

returns the maximum number of frames needed based on the minimum combined factor passed on `CreateInstance`

Returns

virtual int : returns number of frames

4.1.4.9 virtual int CElastiqueProV3DirectIf::GetNumOfInitialUnusedFrames () [pure virtual]

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

Returns

int : returns the number of frames to be omitted

4.1.4.10 virtual int CElastiqueProV3DirectIf::GetNumOfProcessCalls () [pure virtual]

returns the number of process calls needed

Returns

int : returns the number of process calls required

4.1.4.11 virtual int CElastiqueProV3DirectIf::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(..)`

Returns

int : returns the number of frames required

4.1.4.12 virtual int CElastiqueProV3DirectIf::GetProcessedData (float **
ppOutSampleData) [pure virtual]

after processing call that function to retrieve the processed data

Parameters

<i>ppOut- SampleData</i>	:	
------------------------------	---	--

Returns

int : returns the number of frames returned

4.1.4.13 static const int CElastiqueProV3DirectIf::GetVersion (const Version_t
eVersionIdx) [static]

4.1.4.14 virtual bool CElastiqueProV3DirectIf::IsReadyForNextSyncPoint ()
const [pure virtual]

Returns true if the last set sync point has been reached and been processed. It is advised to wait for this method to be true in order to set the next sync point. Otherwise the whole process may be broken.

Returns

bool : returns true is the last sync point has been reached and processed, otherwise false

4.1.4.15 virtual int CElastiqueProV3DirectIf::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

<i>ppInSample- Data</i>	:	double pointer to the input buffer of samples [channels][samples]
<i>iNumOfIn- Frames</i>	:	the number of input frames
<i>ppOut- SampleData</i>	:	double pointer to the output buffer of samples [channels][samples]

Returns

int : returns number of processed frames

4.1.4.16 virtual int CElastiqueProV3DirectIf::PreProcessData (float ** ppInSampleData, int iNumOfInFrames, float ** ppOutSampleData, bool bDisableStartupPerfOptimization = false) [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

<i>ppInSampleData</i>	: double pointer to the input buffer of samples [channels][samples]
<i>iNumOfInFrames</i>	: the number of input frames
<i>ppOutSampleData</i>	: double pointer to the output buffer of samples [channels][samples]
<i>bDisableStartupPerfOptimization</i> , :	if set to true this disables the startup optimization which will lead to a worse performance at the beginning after reset or instance creation but may improve potential transient problems at the beginning.

Returns

int : returns number of processed frames

4.1.4.17 virtual int CElastiqueProV3DirectIf::ProcessData (float ** ppInSampleData, int iNumOfInFrames) [pure virtual]

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

Parameters

<i>ppInSampleData</i>	: double pointer to the input buffer of samples [channels][samples]
<i>iNumOfInFrames</i>	: the number of input frames

Returns

int : returns the error code, otherwise 0

4.1.4.18 virtual int CElastiqueProV3DirectIf::ProcessData () [pure virtual]

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

Returns

int : returns the error code, otherwise 0

4.1.4.19 virtual void CElastiqueProV3DirectIf::Reset () [pure virtual]

resets the internal state of the élastique algorithm

Returns

int : returns some error code otherwise NULL

4.1.4.20 virtual int CElastiqueProV3DirectIf::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

<i>fShiftFactor</i> ;	The envelope shift factor.
-----------------------	----------------------------

@return int: returns some error code otherwise NULL

4.1.4.21 virtual int CElastiqueProV3DirectIf::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

<i>iOrder</i> ;	Sets the order of the spectral envelope estimation
-----------------	--

Returns

int: returns some error code otherwise NULL

4.1.4.22 `virtual int CElastiqueProV3DirectIf::SetHold (bool bSetHold, bool bKeepTime = false) [pure virtual]`

if set to true holds the current audio endlessly until set to false may either resume from the point when hold was enabled or bypass the current audio and resume in time as if the input audio had been playing this may return an error if the previously chosen stretch ratio is too large, reduce to a factor smaller than 6

Parameters

<i>bSetHold</i>	: enable or disable hold
<i>bKeepTime</i> ,:	if set to true the audio stream will continue but will be bypassed

Returns

int : returns some error code otherwise NULL (see above)

4.1.4.23 `virtual int CElastiqueProV3DirectIf::SetNextSyncPoint (int iInputTimeInSamples, int iOutputTimeInSamples, float & fPitchFactor, bool bAssumeTransientAtSyncPoint = true) [pure virtual]`

This method is an alternative way of stretching. Instead setting an explicit stretch factor time stamps for input and output are set and the stretch factor is determined internally in order to achieve the highest timing precision possible for that sync point. Time stamps are always relative to the last play start position (input time resp. output time after Reset) and must be more than 4096 samples apart. Additionally a pitch factor can be set. Using the `SetStretchPitchQFactor(.)` or `SetStretchQPitchFactor(.)` will break the process, therefore it is advised to stick on method or the other.

Parameters

<i>iInputTimeInSamples</i>	: the next input time stamp
<i>iOutputTimeInSamples</i>	: the corresponding output time stamp
<i>fPitchFactor</i>	: the pitch factor for that part to be stretched
<i>bAssumeTransientAtSyncPoint</i>	: prefers transient at sync point if set to true otherwise the normal transient detection is used

Returns

int : returns some error code if input or output time stamps are invalid otherwise NULL

4.1.4.24 `virtual int CElastiqueProV3DirectIf::SetStereoInputMode (CElastiqueProV3DirectIf::ElastiqueStereoInputMode_t eStereoInputMode) [pure virtual]`

sets the stereo input mode

Parameters

<i>eStereo-InputMode</i>	: sets the mode according to <code>_ElastiqueStereoInputMode_</code>
--------------------------	--

Returns

static int : returns some error code otherwise NULL

4.1.4.25 `virtual int CElastiqueProV3DirectIf::SetStretchPitchQFactor (float fStretchFactor, float & fPitchFactor, bool bUsePitchSync = false) [pure virtual]`

sets the internal stretch & pitch factor. The pitch factor is quantized. The product of the stretch factor and the pitch factor must be larger than the `fMinCombinedFactor` defined upon instance creation.

Parameters

<i>fStretch-Factor</i>	: stretch factor
<i>fPitch-Factor,;</i>	pitch factor
<i>bUsePitch-Sync,;</i>	synchronizes timestretch and pitchshifting (default is set to <code>_FALSE</code> in order to preserve old behavior, see documentation), this is ignored in SOLO modes

Returns

int : returns the error code, otherwise 0

4.1.4.26 `virtual int CElastiqueProV3DirectIf::SetStretchQPitchFactor (float & fStretchFactor, float fPitchFactor, bool bUsePitchSync = false) [pure virtual]`

sets the internal stretch & pitch factor. The stretch factor is quantized. The product of the stretch factor and the pitch factor must be larger than the `fMinCombinedFactor` defined upon instance creation.

Parameters

<i>fStretch-Factor</i>	: stretch factor
------------------------	------------------

<i>fPitchFactor</i> ;	pitch factor
<i>bUsePitchSync</i> ;	synchronizes timestretch and pitchshifting (default is set to <code>_FALSE</code> in order to preserve old behavior, see documentation), 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:

- [elastiqueProV3DirectAPI.h](#)

5 File Documentation

5.1 docugen_ProDirect.txt File Reference

5.2 elastiqueProV3DirectAPI.h File Reference

Classes

- class [CElastiqueProV3DirectIf](#)