



elastique efficient time stretching SDK

by zplane.development

(c) 2017 zplane.development GmbH & Co. KG

May 23, 2017

# Contents

<b>1</b>	<b>elastique Efficient 3.x Documentation</b>	<b>2</b>
1.1	Introduction	2
1.2	What's new in V3	2
1.3	Functions explained	2
1.3.1	infiniStretch and Hold explained	2
1.3.2	Pitch synchronization explained	4
1.4	API Documentation	6
1.4.1	Memory Allocation	6
1.4.2	Naming Conventions	6
1.4.3	Stereo Processing	6
1.4.4	C++ API description	6
1.5	Command Line Usage Example	11
1.6	Support	11
<b>2</b>	<b>Class Index</b>	<b>11</b>
2.1	Class List	12
<b>3</b>	<b>File Index</b>	<b>12</b>
3.1	File List	12
<b>4</b>	<b>Class Documentation</b>	<b>12</b>
4.1	CElastiqueV3If 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
<b>5</b>	<b>File Documentation</b>	<b>19</b>
5.1	docugen.txt File Reference	19
5.2	elastiqueV3API.h File Reference	19

# 1 elastique Efficient 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 project contains the required libraries for operating system elastique was licensed for with the appropriate header files.

The structure of this document is as following: First the API of the elastique 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 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

## 1.3 Functions explained

The following sections explain some of the features of elastique V3 that are not obvious at first sight.

### 1.3.1 infiniStretch and Hold explained

infiniStretch and Hold use the same base technology of frequency frame extrapolation. In order to achieve high stretch ratios up to infinity (which equals the Hold function) an already stretched frame is extrapolated. In elastique V3 stretch factors larger than 6 are split into a smaller integer number of smaller stretch ratios. For example a factor of 7 is divided into  $2 \times 3.5$ . Then one block is stretched by 3.5 and the next extrapolated by the same factor based on the results of the previous stretch. The following figure illustrates this behaviour:

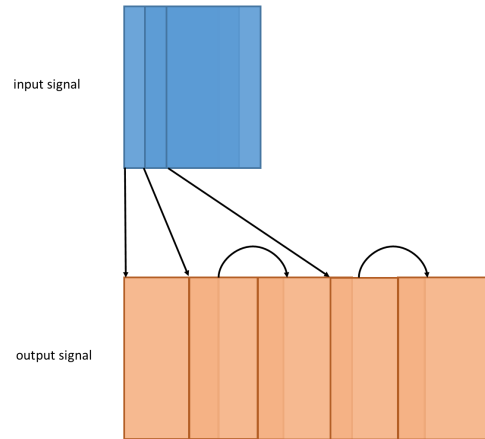


Figure 1: infiniStretch illustrated

The Hold basically uses the same infiniStretch technology with the difference that the last frame is extrapolated (held) endlessly until the Hold function is set to false. There are two different submodes to be differentiated: Hold without time synchronization and with time synchronization. Without time synchronization the input state is frozen and as soon as the Hold function is turned off the input audio continues from the point in time where the Hold function was applied. The following figure illustrates that behavior:

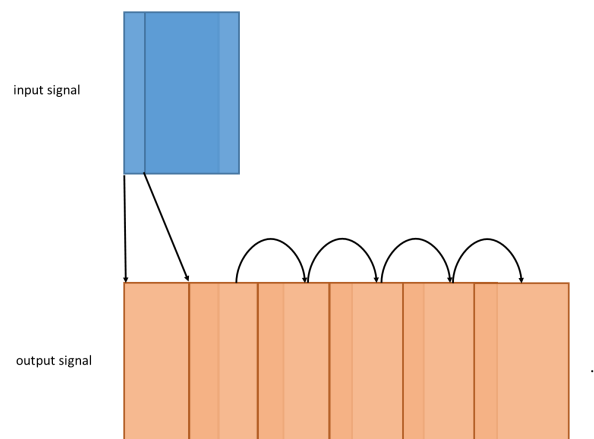


Figure 2: Hold function without time sync illustrated

With time synchronization elastic keeps on requesting input audio in order to simulate a continuing input stream with the current stretch factor. Those input audio blocks are internally ignored and the output is the result of the held audio. As soon as the Hold function is turned off the input audio continues from the point in time where it would if no hold had been applied. The following figure illustrates that behavior:

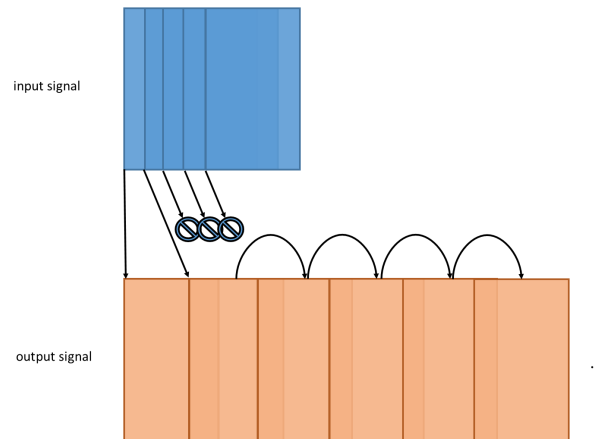


Figure 3: Hold function with time sync illustrated

### 1.3.2 Pitch synchronization explained

Pitchshifting in elastic (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.

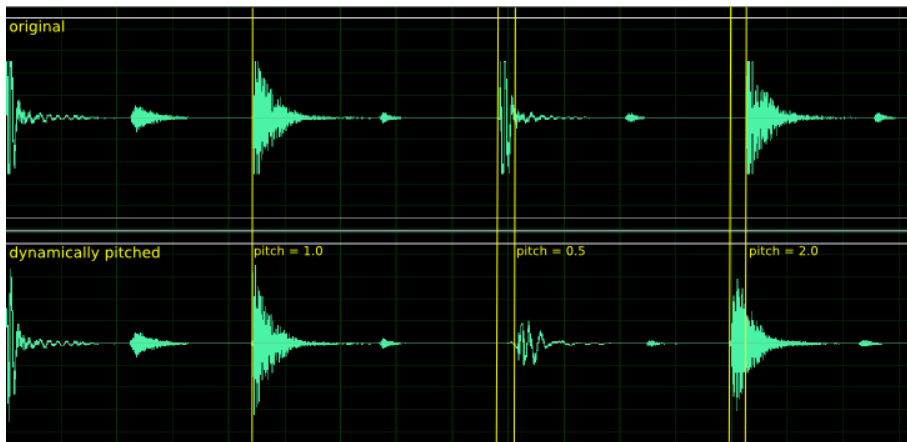


Figure 4: Original and pitched audio without sync

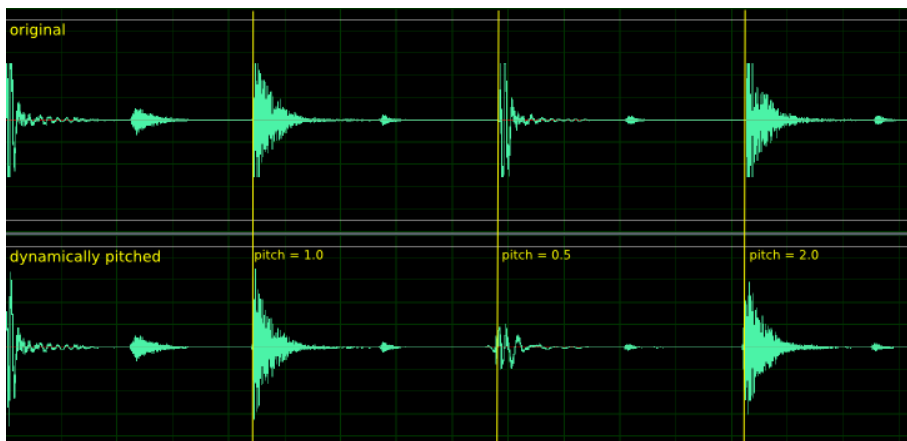


Figure 5: Original and pitched audio with sync

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 syn-

chronization. Otherwise when not set the default is "no-sync" maintaining the behavior of previous versions of the API.

## 1.4 API Documentation

elastique offers two different APIs. The normal and easy to use API can be accessed via the file [elastiqueV3API.h](#), where the class [CElastiqueV3If](#) provides the interface for elastique.

Since version 1.3 there is an alternative API called [elastiqueV3DirectAPI](#). 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 [CElastiqueV3If::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.

- [CElastiqueV3If::CreateInstance](#)(.)

description see below

- **CElastiqueV3If::ProcessData(.)**

description see below

- **CElastiqueV3If::DestroyInstance(.)**

description see below

#### 1.4.4.2 Complete Function Description

##### 1.4.4.2.1 Instance Handling Functions

- **int CElastiqueV3If::CreateInstance** (CElastiqueV3If\* & cCElastique, int iMaxOutputBufferSize, int iNumOfChannels, float fSampleRate, ElastiqueMode\_t eElastiqueMode = kV3, float fMinCombinedFactor = 0.1f);

Creates a new instance of the elastique time stretcher. The handle to the new instance is returned in parameter \*cCElastique. The maximum requested size of the output buffer is given in frames in parameter iMaxOutputBufferSize. This will also be the default output buffer size after instance creation. The output buffer size must not exceed the value of 1024 frames. The parameter iNumOfChannels describes the number of channels with which elastique is used. fSampleRate denotes the input samplerate and eElastiqueMode chooses the processing mode. 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 kV3mobile modes is algorithmically the same as elastique Efficient V2 but features the new infiniStretch and hold functions.

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

The use of this function is required.

- **int CElastiqueV3If::DestroyInstance** (CElastiqueV3If\* cCElastique)

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

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 CElastiqueV3If::SetStretchQPitchFactor**(float& fStretchFactor, float fPitchFactor, bool bUsePitchSync= false);

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



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

Due to the algorithmic properties of the elastique 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 elastique for the time alignment of very long signals. Two different workarounds are applicable to circumvent these issues:

- use the function [CElastiqueV3If::SetStretchPitchQFactor](#) to quantize the pitch factor instead of the stretch factor. This will lead to a minor not noticeable pitch shift in the signal and is therefore the recommended method
- 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.

Setting `bUsePitchSync` to true enables the pitch synchronization as described in [Pitch synchronization explained](#)

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

- **int [CElastiqueV3If::SetStretchPitchQFactor](#) (float fStretchFactor, float& fPitchFactor, bool bUsePitchSync= false)**

This function will do exactly the same as [CElastiqueV3If::SetStretchQPitchFactor](#) (.), but quantize the pitch factor instead of the stretch factor. For a more detailed description, see the description of [CElastiqueV3If::SetStretchQPitchFactor](#) (.).

- **int [CElastiqueV3If::SetCutOffFreq](#) (float fFreq)**

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

- **int [CElastiqueV3If::GetFramesNeeded](#) ()**

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

The method may be called with a new output buffer size as parameter. This will change the output buffer size until it is called with another buffer size again. The new output buffer size must not exceed the the max output buffer size defined upon instance creation.

- **int [CElastiqueV3If::ProcessData](#) (float\*\* ppInSampleData, int iNumOfInFrames, float\*\* ppOutSampleData)**

Processes the input data and returns the stretched output data. The input as well as the output data is given as an array of pointers to the data. This means that, for example, `ppInSampleData[0]` is a pointer to the float input data buffer of the first channel while `ppInSampleData[1]` is the pointer to the float input data buffer of the second input channel. The range of the audio input data is  $\pm 1.0F$ .

The parameter `iNumOfInFrames` describes the number of input frames. Please make sure that this parameter `iNumOfInFrames` equals the value returned by the function `CElastiqueV3If::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.

- **void `CElastiqueV3If::FlushBuffer` (float\*\* `ppfOutSampleData`)**

Gets all the remaining internal frames when no more input data is available and writes them into the buffer `ppfOutSampleData`. Returns the number of written samples.

The use of this function is optional.

- **void `CElastiqueV3If::Reset` ()**

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

The use of this function is optional.

#### 1.4.4.3 C++ Usage example

The complete code can be found in the example source file `elastiqueEfficientTestCL.cpp`.

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

```
CElastiqueV3If *pcElastiqueHandle = 0;
```

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

```
iError = CElastiqueV3If::CreateInstance( pcElastiqueHandle ,
    _OUTPUT_BUFFER_SIZE,
    iNumOfChannels,
    (float)iSampleRate,
    (CElastiqueV3If::ElastiqueMode_t) iMode
);

if (iError)
{
    cout << "Instance Create Error!" << endl;
    delete pCInputFile;
    delete pCOutputFile;
    return -1;
}
```

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

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

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

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

In the next step, the processing loop can be executed

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

In the first step, we have to ask elastique 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 or AIFF audio file using the `zpl-AudioFile` library provided with the SDK.

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

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

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

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

```
pCOutputFile->Write(apfProcessOutputData, (_OUTPUT_BUFFER_SIZE));
```

and are at the end of the processing loop:

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

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

```
while ((iNumOfFrames = pcElastiqueHandle->FlushBuffer(
apfProcessOutputData) > 0)
{
    iTotalFrames += iNumOfFrames;
    pCOutputFile->Write(apfProcessOutputData, (iNumOfFrames));
```

After the successful processing, the instance of elastique can be destroyed

```
        delete [] apfProcessInputData[iChIdx];
        apfProcessInputData[iChIdx] = 0;

        if (apfProcessOutputData[iChIdx])
            delete [] apfProcessOutputData[iChIdx];
        apfProcessOutputData[iChIdx] = 0;
    }

    // instance destroy
    delete pCInputFile;
    delete pCOutputFile;
    // delete instance
    CElastiqueV3If::DestroyInstance(pcElastiqueHandle);
```

## 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:

```
elastiqueEfficientTestCL -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):

```
elastiqueEfficientTestCL -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](http://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](mailto:flohrer@zplane.de)

## 2 Class Index

## 2.1 Class List

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

[CElastiqueV3If](#) 12

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

[elastiqueV3API.h](#) 19

## 4 Class Documentation

### 4.1 CElastiqueV3If Class Reference

```
#include <elastiqueV3API.h>
```

#### Public Types

- enum [ElastiqueStereoInputMode\\_t](#) { [kPlainStereoMode](#) = 0, [kMSMode](#) }
- enum [ElastiqueMode\\_t](#) { [kV3](#) = 0, [kV3mobile](#) }
- enum [Version\\_t](#) { [kMajor](#), [kMinor](#), [kPatch](#), [kRevision](#), [kNumVersionInts](#) }

#### Public Member Functions

- virtual [~CElastiqueV3If](#) ()
- virtual int [ProcessData](#) (float \*\*ppInSampleData, int iNumOfInFrames, float \*\*ppOutSampleData)=0
- virtual int [GetFramesNeeded](#) (int iOutBufferSize)=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 int [FlushBuffer](#) (float \*\*ppfOutSampleData)=0
- virtual int [GetFramesBuffered](#) ()=0
- virtual int [SetStereoInputMode](#) ([ElastiqueStereoInputMode\\_t](#) eStereoInputMode)=0
- virtual int [SetCutOffFreq](#) (float fFreq)=0
- virtual long long [GetCurrentTimePos](#) ()=0
- virtual int [SetHold](#) (bool bSetHold, bool bKeepTime=false)=0

### Static Public Member Functions

- static const int [GetVersion](#) (const [Version\\_t](#) eVersionIdx)
- static const char \* [GetBuildDate](#) ()
- static int [CreateInstance](#) ([CElastiqueV3If](#) \*&cCElastique, int iMaxOutputBufferSize, int iNumOfChannels, float fSampleRate, [ElastiqueMode\\_t](#) eElastiqueMode=[kV3](#), float fMinCombinedFactor=0.1f)
- static int [DestroyInstance](#) ([CElastiqueV3If](#) \*cCElastique)

#### 4.1.1 Detailed Description

##### CLASS

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

##### USAGE

Definition at line 59 of file [elastiqueV3API.h](#).

#### 4.1.2 Member Enumeration Documentation

##### 4.1.2.1 enum [CElastiqueV3If::ElastiqueMode\\_t](#)

Enumerator:

- kV3***
- kV3mobile***

Definition at line 71 of file [elastiqueV3API.h](#).

```
{
    kV3 = 0,
    kV3mobile,
};
```

##### 4.1.2.2 enum [CElastiqueV3If::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 65 of file [elastiqueV3API.h](#).

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

## 4.1.2.3 enum CElastiqueV3If::Version\_t

Enumerator:

*kMajor*  
*kMinor*  
*kPatch*  
*kRevision*  
*kNumVersionInts*

Definition at line 77 of file elasticV3API.h.

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

    kNumVersionInts
};
```

## 4.1.3 Constructor &amp; Destructor Documentation

## 4.1.3.1 virtual CElastiqueV3If::~CElastiqueV3If( ) [inline, virtual]

Definition at line 63 of file elasticV3API.h.

```
{};
```

## 4.1.4 Member Function Documentation

4.1.4.1 static int CElastiqueV3If::CreateInstance ( CElastiqueV3If \**cCElastique*, int *iMaxOutputBufferSize*, int *iNumOfChannels*, float *fSampleRate*, ElasticqueMode\_t *eElastiqueMode* = kV3, float *fMinCombinedFactor* = 0.1f ) [static]

creates an instance of zplane's élastique class

## Parameters

<i>cCElastique</i>	: returns a pointer to the class instance
<i>iOutput- BufferSize</i>	: desired max number of frames at the output, if not changed by Get-FramesNeeded(.) this one is the default output block size, maximum is 1024 sample frames
<i>iNumOf- Channels</i>	:
<i>fSampleRate</i>	: input samplerate
<i>fMin- Combined- Factor,:</i>	minimum combined factor of stretch * pitch to be used

**Returns**

static int : returns some error code otherwise NULL

**4.1.4.2 static int CElastiqueV3If::DestroyInstance ( CElastiqueV3If \*  
 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 CElastiqueV3If::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 NULL

**4.1.4.4 static const char\* CElastiqueV3If::GetBuildDate ( ) [static]**
**4.1.4.5 virtual long long CElastiqueV3If::GetCurrentTimePos ( ) [pure  
 virtual]**

gets the current input time position in sample frames

**Returns**

long long: returns the time position

**4.1.4.6 virtual int CElastiqueV3If::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.



**Returns**

int : returns the number of frames buffered

**4.1.4.7 virtual int CElastiqueV3If::GetFramesNeeded ( int *iOutBufferSize* )**  
 [pure virtual]

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

**Parameters**

<i>iOutBufferSize</i>	: required output buffer size
-----------------------	-------------------------------

**Returns**

virtual int : returns the number of frames required or -1 if OutBufferSize exceeds the maxbuffersize defined upon createinstance

**4.1.4.8 virtual int CElastiqueV3If::GetFramesNeeded ( )** [pure virtual]

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

**Returns**

virtual int : returns the number of frames required

**4.1.4.9 virtual int CElastiqueV3If::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

**4.1.4.10 static const int CElastiqueV3If::GetVersion ( const Version\_t *eVersionIdx* )**  
 [static]

**4.1.4.11 virtual int CElastiqueV3If::ProcessData ( float \*\* *ppInSampleData*, int *iNumOfInFrames*, float \*\* *ppOutSampleData* )** [pure virtual]

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

**Parameters**

<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]

**Returns**

virtual int : returns the error code, otherwise 0

#### 4.1.4.12 virtual void CElastiqueV3If::Reset ( ) [pure virtual]

resets the internal state of the élastique algorithm

**Returns**

int : returns some error code otherwise NULL

#### 4.1.4.13 virtual int CElastiqueV3If::SetCutOffFreq ( float *fFreq* ) [pure virtual]

sets a cutoff frequency in order to save processing time

**Parameters**

<i>fFreq</i>	: cutoff freq in Hz
--------------	---------------------

**Returns**

int : returns some error code otherwise NULL

#### 4.1.4.14 virtual int CElastiqueV3If::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

**Parameters**

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

**4.1.4.15** `virtual int CElastiqueV3If::SetStereoInputMode ( ElastiqueStereoInputMode_t eStereoInputMode )` [pure virtual]

sets the stereo input mode

#### Parameters

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

#### Returns

int : returns some error code otherwise NULL

**4.1.4.16** `virtual int CElastiqueV3If::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)

#### Returns

int : returns the error code, otherwise 0

**4.1.4.17** `virtual int CElastiqueV3If::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>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)

**Returns**

int : returns the error code, otherwise 0

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

- [elastiqueV3API.h](#)

## **5 File Documentation**

### **5.1 docugen.txt File Reference**

### **5.2 elastiqueV3API.h File Reference**

**Classes**

- class [CElastiqueV3If](#)