



elastique Tune time and pitch modification 1.2.0

by zplane.development

(c) 2016 zplane.development GmbH & Co. KG

July 15, 2016

Contents

1	élastique Tune 1.1 SDK Documentation	2
1.1	Introduction	2
1.2	API Documentation	3
1.2.1	Memory Allocation	3
1.2.2	Naming Conventions	3
1.2.3	Stereo/Multichannel Processing	3
1.2.4	C++ API description	3
1.2.5	C++ Usage example	13
1.3	Delivered Files (example project)	18
1.3.1	File Structure	18
1.4	Coding Style minimal overview	19
1.5	Command Line Usage Example	19
1.6	Support	19
2	Class Index	20
2.1	Class List	20
3	File Index	20
3.1	File List	20
4	Class Documentation	21
4.1	_stPitch_ Struct Reference	21
4.1.1	Detailed Description	21
4.1.2	Member Data Documentation	21
4.2	_stPitchSegmentationResult_ Struct Reference	22
4.2.1	Detailed Description	22
4.2.2	Member Data Documentation	22
4.3	_stSOLOISTPitchObjectData_ Struct Reference	23
4.3.1	Detailed Description	25
4.3.2	Member Data Documentation	25
4.4	CAudioReader Class Reference	27
4.4.1	Detailed Description	28
4.4.2	Constructor & Destructor Documentation	28
4.4.3	Member Function Documentation	28
4.5	CSOLOISTPitchAnalysisIf Class Reference	30
4.5.1	Detailed Description	32
4.5.2	Constructor & Destructor Documentation	32
4.5.3	Member Function Documentation	32
4.6	CSOLOISTPitchSynthesizerIf Class Reference	39
4.6.1	Detailed Description	41
4.6.2	Member Enumeration Documentation	41
4.6.3	Member Function Documentation	44
5	File Documentation	51
5.1	DoxyfileTune.txt File Reference	51
5.2	PSOLAAPI.h File Reference	51
5.2.1	Detailed Description	51
5.3	SOLOISTPitchAnalysisAPI.h File Reference	51

5.3.1	Typedef Documentation	52
5.4	SOLOISTPitchSynthesizerAPI.h File Reference	52
5.4.1	Typedef Documentation	53
5.4.2	Function Documentation	53

1 élastique Tune 1.1 SDK Documentation

1.1 Introduction

élastique Tune is a time and pitch correction technology. It is based on the élastique SOLOIST engine and thus works only with monophonic input signals. While SOLOIST provides a low level approach to monophonic time stretching and pitch shifting, Tune allows high level and easy access to musical parameters. Based on a simple and easy to use interface it offers numerous editing options based on pitch objects which correspond closely to the musical concept of notes. The Tune SDK provides the functionality for Audio-to-MIDI conversion as well as a synthesis class to control the playback while applying all parameters for time and pitch manipulation in real-time. The interface offers two parameter groups, one controlling overall melody properties and the other controlling parameters of each individual note. The following figure shows an example of how such a representation could look like.

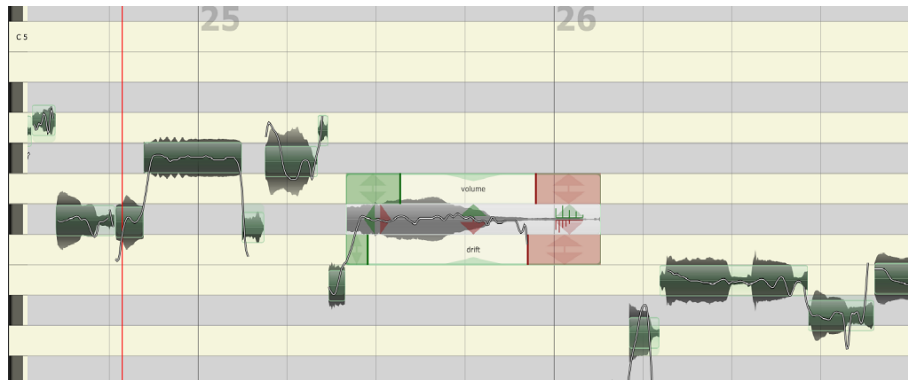


Figure 1: Example of visualization of Tune analysis output with editing options

The SDK consists of an analysis and an synthesis library. The latter uses a customizable audio access class in order to ease integration in existing systems.

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

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

For detailed information about how PSOLA synthesis works please refer to the élastique SOLOIST manual.

1.2 API Documentation

The C++-API can be accessed via the files [SOLOISTPitchAnalysisAPI.h](#) and [SOLOISTPitchSynthesizerAPI.h](#). All variable types needed are either defined in file [elastique-API.h](#) or standard C++-types. Error codes are defined as enum in the corresponding class.

1.2.1 Memory Allocation

The Tune SDK does not allocate buffers handled by the calling application except for the buffers containing the results of the analysis. The input buffer as well as the output buffer has to be allocated by the calling application. The exact size of the input and output buffer is defined by the user depending on how many sample frames are passed in or are requested. The only place where audio buffer sizes are determined by the SDK is in the [CAudioReader](#) class which has to be overridden in order to match the users requirements. Buffers are allocated as double arrays of [channels][SamplesPerChannel].

*

1.2.2 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.

1.2.3 Stereo/Multichannel Processing

When processing stereo or multichannel input, it is strongly recommended to use the SDK with a stereo/multichannels instance, not with two or more mono instances. - 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 processing is linked between all channels. The performance will be better since many analysis steps can be combined for all channels.

1.2.4 C++ API description

1.2.4.1 Required Functions

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

- **CPitchMarksIf::CreateInstance(.)**
description see below
- **CPitchMarksIf::GenerateInitialBuffers(.)**
description see below

- **CPitchMarksIf::DestroyInstance(.)**
description see below
- **CSOLOISTPitchAnalysisIf::CreateInstance(.)**
description see below
- **CSOLOISTPitchAnalysisIf::ProcessData(.)**
description see below
- **CSOLOISTPitchAnalysisIf::DestroyInstance(.)**
description see below
- **CSOLOISTPitchSynthesizerIf::CreateInstance(.)**
description see below
- **CSOLOISTPitchSynthesizerIf::processData(.)**
description see below
- **CSOLOISTPitchSynthesizerIf::DestroyInstance(.)**
description see below
- **An audio reader class must be derived from CAudioReader**
description see below

1.2.4.2 Complete Function Description

1.2.4.2.1 Instance Handling Functions

- **int CPitchMarksIf::CreateInstance(CPitchMarksIf* & pCPitchMarks)**
Creates a new instance of the élastique SOLOIST pitchmarks. The handle to the new instance is returned in parameter *pCPitchMarks. If the function fails, the return value is not 0.
- **int CSOLOISTPitchAnalysisIf::CreateInstance(CSOLOISTPitchAnalysisIf* & pCInstance, CPitchMarksIf* pcPitchMarks, float fSampleRate, int iNumOfChannels)**
Creates a new instance of the analysis class. The handle to the new instance is returned in parameter *pCInstance. The handle to the pitchmarks to be calculated is passed via pcPitchMarks. The parameters fSampleRate and iNumOfChannels describe the sample rate as well as the number of channels of the audio to be analysed.
If the function fails, the return value is not 0.
- **int CSOLOISTPitchSynthesizerIf::CreateInstance(CSOLOISTPitchSynthesizerIf* & pCInstance, CAudioReader* const pCAudioReader, CPitchMarksIf* const pcPitchMarks, CSOLOISTPitchAnalysisIf* const pCAnalysisIf)**
Creates a new instance of the synthesis class. The handle to the new instance is returned in parameter *pCInstance. A deriec instance of the custom audio

reader class is passed via `pCAudioReader`. `pCPitchMarks` receives the handle to the previously calculated pitchmarks and a pointer to the corresponding analysis class has to be provided.

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

The use of this function is required.

- **int CPitchMarksIf::DestroyInstance(CPitchMarksIf* pCPitchMarks)**
Destroys the instance of the pitchmarks given in parameter `*pCPitchMarks`.
If the function fails, the return value is not 0.
- **int CSOLOISTPitchAnalysisIf::DestroyInstance(CSOLOISTPitchAnalysisIf* & pCInstance)**
Destroys the instance of the analysis given in parameter `*pCInstance`.
If the function fails, the return value is not 0.
- **int CSOLOISTPitchSynthesizerIf::DestroyInstance(CSOLOISTPitchSynthesizerIf* & pCInstance)**
Destroys the instance of the synthesis given in parameter `*pCInstance`.
If the function fails, the return value is not 0.

1.2.4.2.2 Pitchmark Functions

- **int CPitchMarksIf::Reset()**
Resets pitchmarks to its initial state.
If the function fails, the return value is not 0.
- **int CPitchMarksIf::FlushPitchMarks(int iIdx2FlushTo)**
This function flushes pitchmarks until index specified in `iIdx2FlushTo`. it is not recommended to use this function.
If the function fails, the return value is not 0.
- **int CPitchMarksIf::GenerateInitialBuffers(int iInitialSize)**
Pre-allocates a set of pitchmarks. Should be called after instance creation to avoid unnecessary reallocation. If more pitchmarks are needed these are reallocated automatically.
If the function fails, the return value is not 0.
- **int CPitchMarksIf::GetPitchMarkBuffer(BSampleInfoEntry* & psPitchMarkBuffer)**
Returns a pointer to the internal list of marks. The return value is the number of list entries. One should use this funktion to save the result of the onset detection or beat tracking process.
If the function fails, the return value is not 0.
- **int CPitchMarksIf::GetNumOfMarks()**
Returns the current number of marks in the list.

- **int CPitchMarksIf::PutBuffers(BSampleInfoEntry* psPitchMarkBuffer, int iSize)**

Copies a previously saved pitchmark list (see CPitchMarksIF::GetPitchMarkBuffer()) into an instance of the mark module.

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

1.2.4.2.3 Analysis Functions

- **int CSOLOISTPitchAnalysisIf::Reset()**

Resets the analysis to its initial state.

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

- **int CSOLOISTPitchAnalysisIf::ProcessData(float** ppSampleData, int iNumOfFrames)**

Does the pitch analysis processing. ppSampleData contains the input audio data. iNumOfFrames contains the number of samples per channel.

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

- **int CSOLOISTPitchAnalysisIf::SetEOF()**

Declares the end of audio samples, invokes last processing stages and flushes internal buffers. Should be called after last block of audio has been pushed into the CPSOLAAnalysisEnhIf::ProcessData() function.

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

- **int CSOLOISTPitchAnalysisIf::DoSegmentation()**

Does the pitch segmentation after the pitch analysis. Must not be called before ProcessData() has been finished by calling SetEOF(). This method can be called alternatively with an integer parameter (0..8) in order to split the analysis in smaller parts. This is helpfull when displaying a progress bar for example.

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

- **int CSOLOISTPitchAnalysisIf::GetSegmentationResult(stPitchSegmentationResult* & pstSegmentationResults)**

Returns a pointer to a buffer containing the results of the segmentation analysis.

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

1.2.4.2.4 Synthesis Functions

- **int CSOLOISTPitchSynthesizerIf::processData(float** ppfAudioData, int iNumOfFramesToRead)**

Does the synthesis processing at the current playback position. It returns iNumOfFramesToRead sample frames. The buffers for ppfAudioData must be allocated by the calling application.

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

- **void CSOLOISTPitchSynthesizerIf::setTimePositionInSec(double newReadPosition)**
Sets a new read position. The read position is always relative to the beginning of the audio file. If the read position is out of bounds zero buffers are returned.
- **double CSOLOISTPitchSynthesizerIf::getTimePositionInSec()**
Returns the current read position.
- **double CSOLOISTPitchSynthesizerIf::getLengthInSec()**
Returns the current playback length. It takes all time modification into account.
- **float CSOLOISTPitchSynthesizerIf::getPitchForTimeInSec(double dTime)**
Returns the pitch at the time position specified. This is useful for drawing a pitch curve. If no pitch is available it returns -1.
- **float CSOLOISTPitchSynthesizerIf::getEnvelopeForTimeInSec(double dTime)**
Returns the volume at the time position specified. This is useful for drawing a volume envelope.
- **int CSOLOISTPitchSynthesizerIf::getNumOfPitchObjects()**
Returns the number of pitch objects currently in use by the synthesizer.
- **_errorCode CSOLOISTPitchSynthesizerIf::removePitchObject(int iPitchObjectIndex)**
Removes a pitch object by index.
If the function fails, the return value is not 0.
- **void CSOLOISTPitchSynthesizerIf::removeAllPitchObjects()**
Removes all pitch objects.
- **void CSOLOISTPitchSynthesizerIf::addPitchObject(stSOLOISTPitchObjectData& stInitialPitchObjectData)**
Adds a pitch objects at the correct time position. All the data in the structure stInitialPitchObjectData must be valid.
If the function fails, the return value is not 0.
- **_errorCode CSOLOISTPitchSynthesizerIf::splitPitchObject(int iPitchObjectIndex, double dRelativeCutPositionFromStartOfPitchObject)**
Splits a pitch object at the position specified. The split position is relative in seconds to the beginning of the pitch object.
If the function fails, the return value is not 0.
- **_errorCode CSOLOISTPitchSynthesizerIf::joinPitchObjects(int iPitchObjectIndex1, int iPitchObjectIndex2)**
Joins two adjacent note objects. The object indices don't need to be in order.
If the function fails, the return value is not 0.

- **`_errorCode CSOLOISTPitchSynthesizerIf::getPitchObjectData(int iPitch-ObjectIndex, stSOLOISTPitchObjectData& ObjData)`**
Returns a complete data structure for the given pitch object.
If the function fails, the return value is not 0.
- **`_errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectData(int iPitch-ObjectIndex, stSOLOISTPitchObjectData& ObjData, bool bUpdateSurrounding-Objects)`**
Sets the complete data structure for the given pitch object. Modifying the data structure should only be done with the proper knowledge.
If the function fails, the return value is not 0.
- **`_errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectBounds(int iPitch-ObjectIndex, double& dNewStartPos, double& dNewEndPos, bool bUpdate-SurroundingObjects)`**
Changes the start/end position of a pitch object affecting the stretch factor of the note if start and end position change differently. If `bUpdateSurroundingObjects` is set true the surrounding objects are also adapted which may affect the stretch factor of these objects, too.
If the function fails, the return value is not 0.
- **`_errorCode CSOLOISTPitchSynthesizerIf::getPitchObjectBounds(int iPitch-ObjectIndex, double& dStartPos, double& dEndPos)`**
Returns the current start/end position of a pitch object.
If the function fails, the return value is not 0.
- **`_errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectParam(int iPitch-ObjectIndex, _ePitchObjectParam eParam, float fValue)`**
Sets a pitch object parameter. Parameters are described in [Synthesis Object - Parameters](#)
If the function fails, the return value is not 0.
- **`float CSOLOISTPitchSynthesizerIf::getPitchObjectParam(int iPitchObject-Index, _ePitchObjectParam eParam) const`**
Returns a pitch object parameter. Parameters are described in [Synthesis Object Parameters](#)
If index is out of bounds it returns 0
- **`_errorCode CSOLOISTPitchSynthesizerIf::setGlobalParam(_eGlobalParam eParam, float fValue)`**
Set a global parameter. Parameters are described in [Synthesis Global Parameters](#)
If the function fails, the return value is not 0.
- **`float CSOLOISTPitchSynthesizerIf::getGlobalParam(_eGlobalParam eParam) const`**
Returns a global parameter. Parameters are described in [Synthesis Global - Parameters](#)

1.2.4.2.5 Synthesis Object Parameters

- **CSOLOISTPitchSynthesizerIf::kPitchObjectPitch**

This sets directly the pitch of the whole pitch object as seen in the following figure. Unit is semitones.

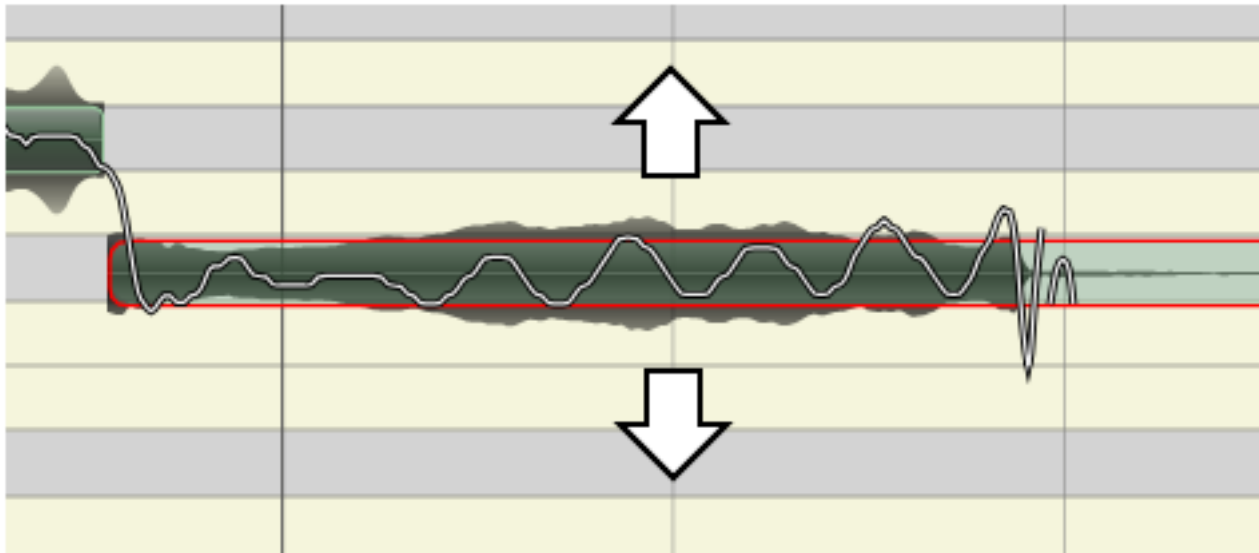


Figure 2: changing the pitch

- **CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothInTimeInPercent**
CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothOutTimeInPercent

These parameters define the transition range that is automatically adapted when changing the pitch of a pitch object. This is to keep smooth transitions between two pitch objects. Unit is in percent divided by 100 relative to the pitch object length. In-time is measured from the beginning, out-time from the end of the object.

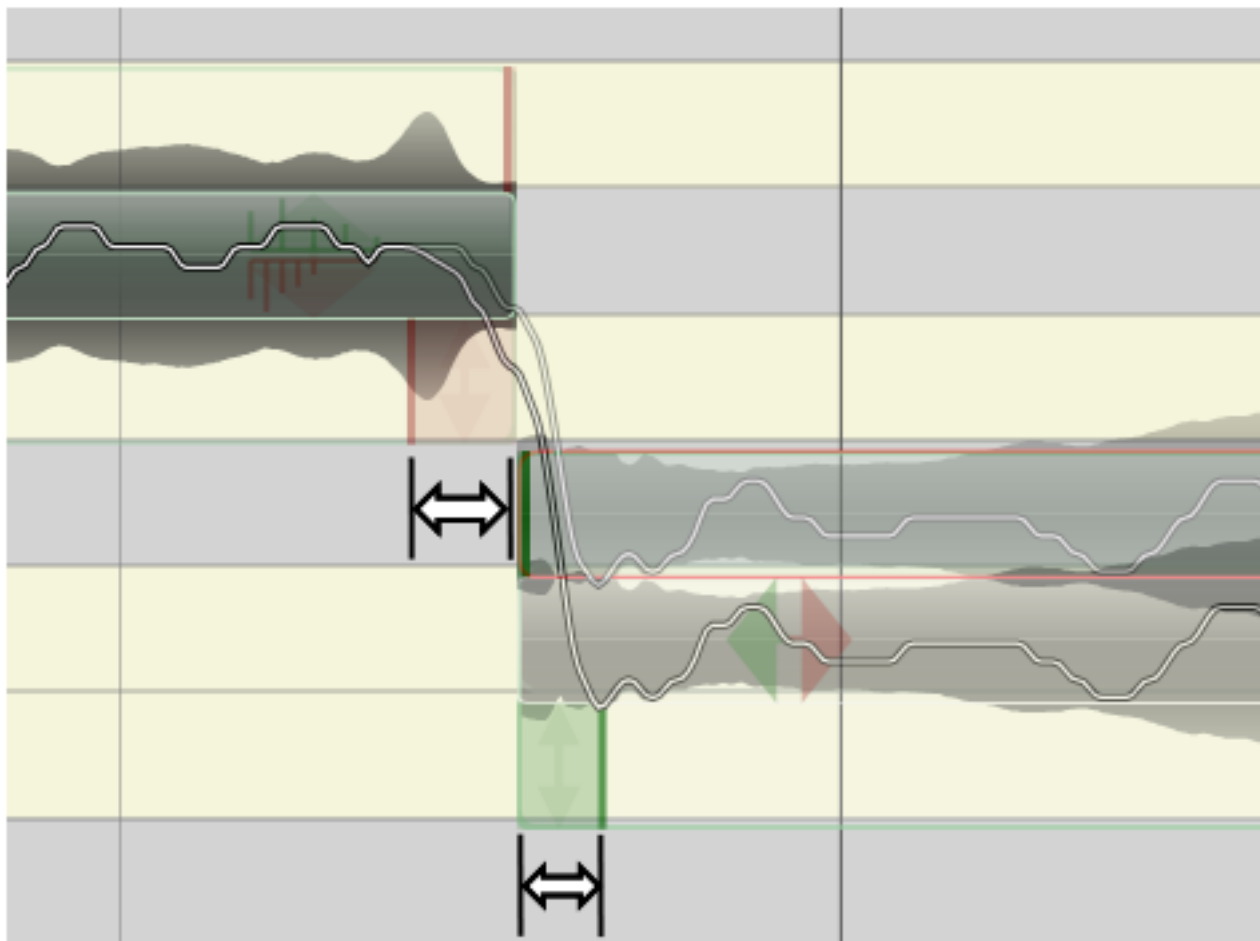


Figure 3: smooth pitch in and out time

- **CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothInPitch**
CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothOutPitch
These parameters can be used to adapt the automatic transition smoothing (see above) by ramping the transition up or down. Unit is semitones.
- **CSOLOISTPitchSynthesizerIf::kPitchObjectVolumeFadeInTimeInPercent**
CSOLOISTPitchSynthesizerIf::kPitchObjectVolumeFadeOutTimeInPercent
These parameters define the volume fade range. Unit is in percent divided by 100 relative to the pitch object length. In-time is measured from the beginning, out-time from the end of the object.

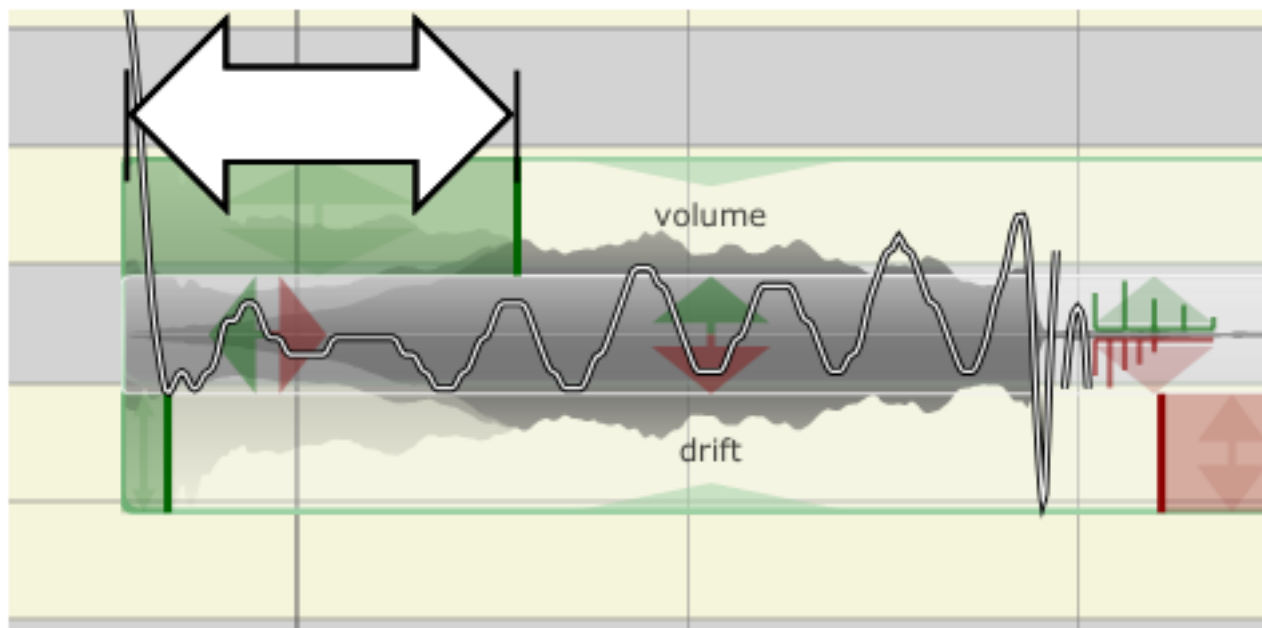


Figure 4: smooth pitch in and out time

- **`CSOLOISTPitchSynthesizerIf::kPitchObjectVolumeFadeInGain`**
`CSOLOISTPitchSynthesizerIf::kPitchObjectVolumeFadeOutGain`
These parameters determine the start fade-in gain or end fade-out gain (see above). Unit is in percent divided by 100 relative to the pitch object length.
- **`CSOLOISTPitchSynthesizerIf::kPitchObjectFormantShift`**
This affects the formants of the pitch object. Unit is semitones. The default is 0 semitones.
- **`CSOLOISTPitchSynthesizerIf::kPitchObjectDrift`**
The drift parameter defines the deviation of the local pitch from the average pitch of the pitch object as seen in the following figure. Unit is percent divided by 100. A value of 1.0 is the original deviation.

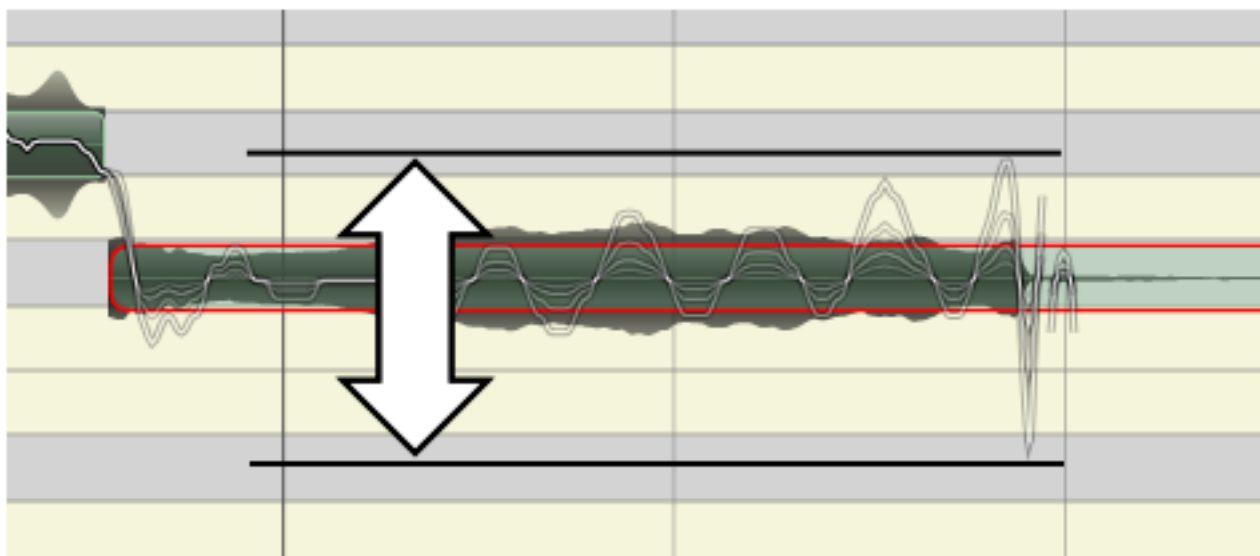


Figure 5: changing the drift

- **CSOLOISTPitchSynthesizerIf::kPitchObjectVolume**

Defines the volume of a pitch object. Unit is percent divided by 100. A value of 1.0 is the original deviation.

1.2.4.2.6 Synthesis Global Parameters

- **CSOLOISTPitchSynthesizerIf::kGlobalStretch**

Defines the global stretch factor for whole audio file. Unit is percent divided by 100.

- **CSOLOISTPitchSynthesizerIf::kGlobalPitchShift**

Defines the global pitch factor for the whole audio file. Unit is semitones.

- **CSOLOISTPitchSynthesizerIf::kGlobalFormantShift**

Defines the global formant shift factor for the whole audio file. Unit is semitones.

- **CSOLOISTPitchSynthesizerIf::kGlobalTune**

Defines the global tuning of the whole audio file. Unit is percent divided by 100. Default is 0 leaving everything as it is. With a tuning of 1.0 the average pitch of each pitch object is moved to it's integer pitch. E.g.: a pitch object has an average pitch of 69.32, with a tuning of 1.0 it's moved to 69.0.

- **CSOLOISTPitchSynthesizerIf::kGlobalDrift**

Affects the drift of all pitch objects at once (see above). This is independent of the local drift settings of the pitch object.

- **CSOLOISTPitchSynthesizerIf::kGlobalTransitionFactor**

The transition factor determines how the drift factor affects the smooth in/out area. This is useful to keep smooth transitions also with a drift of 0. Unit is percent divided by 100 and default is 1.0 which means that the transitions are kept smooth. With a factor of 0 and a drift of 0 transitions are abrupt.

1.2.5 C++ Usage example

The complete code can be found in the example source file `elastiqueTuneCMain.cpp`. This example works on raw 16bit PCM audio data using `stdio` file functions.

In the first step we need to define our own [CAudioReader](#) class for the synthesis later on:

```
class CMyAudioReader : public CAudioReader
{
private:
    CzplAudioFile    *m_pFFileHandle;
    int              m_iNumOfChannels,
                   m_iSampleRate,
                   m_iLengthInFrames;

public:
    CMyAudioReader(CzplAudioFile *pFInputFile, int iMaxBufferSize) :
        m_pFFileHandle(pFInputFile)

    {
        m_iNumOfChannels = m_pFFileHandle->GetNumOfChannels();
        m_iSampleRate    = m_pFFileHandle->GetSampleRate();

        m_iLengthInFrames = m_pFFileHandle->GetFileSize();
    };

    virtual ~CMyAudioReader()
    {
    };

    int      seekPos(int iPositionInFrames)
    {
        m_pFFileHandle->SetFilePos(iPositionInFrames);
        return 0;
    };

    int      readAudioData(float** ppfAudioData, int iNumOfFramestoRead)
    {
        m_pFFileHandle->Read(ppfAudioData, iNumOfFramestoRead);

        return 0;
    }

    int      getLengthInFrames()
    {
        return m_iLengthInFrames;
    }

    int      getNumOfChannels()
    {
        return m_iNumOfChannels;
    }
};
```

```

};

int      getSampleRate()
{
    return m_iSampleRate;
}

};

```

Before synthesis we need to either do the analysis or reload the analysis results. Here we do the analysis. Before that an instance of the pitchmark class has to be created. After that it is recommended to pre-allocate some pitchmarks in order to speed up processing.

```

CPitchMarksIf::CreateInstance(pcPitchMarksHandle);

// init the pitchmark container with a whole lot of empty pitchmarks
pcPitchMarksHandle->GenerateInitialBuffers(8192);

```

Now we create an instance of the analysis class passing a handle to the previously created pitchmarks and setting the other parameters according to our audio input.

```

CSOLOISTPitchAnalysisIf::CreateInstance(pcAnalysisHandle,
    pcPitchMarksHandle, static_cast<float>(iSampleRate), iNumOfChannels);

```

Now we run the analysis loop until the audio data is finished. The audio samples are simply pushed into the [CSOLOISTPitchAnalysisIf::ProcessData\(\)](#) function. The number of frames per call should not exceed 4096.

```

while (bReadNextFrame)
{
    // read the samples from the PCM input file
    iNumSamplesRead = pFInputFile->Read(apfProcessInputData,
        _ANALYSIS_BUFFER_SIZE);

    // if the number of read frames unequals the required number of frames,
    // the end of the file is reached
    // then, fill the remaining buffer values with zeros
    // and set flag to end loop
    if(iNumSamplesRead < (_ANALYSIS_BUFFER_SIZE))
    {
        bReadNextFrame = false;
    }

    // start a simple time measurement
    clStartTime = clock();

    //#if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined (WIN32))
    //    _controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW |
    //        _EM_UNDERFLOW | _EM_DENORMAL), _MCW_EM) ;
    //#endif // #ifndef WITHOUT_EXCEPTIONS

```



```

// do the analysis processing
iError = pcAnalysisHandle->ProcessData( apfProcessInputData,
                                       iNumSamplesRead);

if (iError)
{
    fprintf(stderr, "élastique Analysis Error No.:%d\n", iError);
    break;
}

// update time measurement
clTotalTime += clock() - clStartTime;

}

```

After having finished the loop `CPSOLAAnalysisEnhIf` has to be told that we finished.

```
pcAnalysisHandle->SetEOF();
```

Now all pitchmarks are set. These pitchmarks may be saved for later use.

Now we should do the segmentation:

```
pcAnalysisHandle->DoSegmentation();
```

This gives you these results:

```

iNumOfPitches   = pcAnalysisHandle->GetPitchResult(pPitchResult);
iNumOfSegments  = pcAnalysisHandle->GetSegmentationResult(pSegmentResult);

```

Finally, the synthesis may be invoked. First we need an instance of our [CAudioReader](#) class.

```
pcAudioReader = new CMyAudioReader(pFInputFile, _INPUT_BUFFER_SIZE);
```

and use this to create the synthesis class.

```
CSOLOISTPitchSynthesizerIf::CreateInstance( pcSynthesisHandle,
                                           pCAudioReader, pcPitchMarksHandle, pcAnalysisHandle);
```

we set some globale parameters

```

/*
pcSynthesisHandle->setGlobalParam(CSOLOISTPitchSynthesizerIf::kGlobalPitchShift, fPitchRatio);

pcSynthesisHandle->setGlobalParam(CSOLOISTPitchSynthesizerIf::kGlobalStretch
, fStretchRatio);

```

and then do some direct manipulation upon a single pitch object

```
// pcSynthesisHandle->setPitchObjectParam(ii,
    CSOLOISTPitchSynthesizerIf::kPitchObjectPitch, 67.0);

{
    //typedef CSOLOISTPitchSynthesizerIf::PitchClass PitchClass;
    //std::vector <PitchClass> myScale = { PitchClass::C, PitchClass::D,
    PitchClass::E, PitchClass::F, PitchClass::G, PitchClass::A, PitchClass::B };

    std::vector <CSOLOISTPitchSynthesizerIf::PitchClass> myScale =
    generateScaleBasedOnScaleIdx (pcAnalysisHandle->GetKey());

    pcSynthesisHandle->setGlobalParam (
    CSOLOISTPitchSynthesizerIf::kGlobalDrift, 0.0);

    pcSynthesisHandle->setGlobalParam (
    CSOLOISTPitchSynthesizerIf::kGlobalTune, 1.0);

    pcSynthesisHandle->setGlobalParam (
    CSOLOISTPitchSynthesizerIf::kGlobalTransitionFactor, 0.0);

    pcSynthesisHandle->quantizePitchObjectsToScale (myScale);
}

//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectDrift, 0.0);
//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectFormantShift, 0.0);
//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothInTimeInPercent, 0.5);
//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothInPitch, -5);
//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothOutTimeInPercent, 0.5);
//pcSynthesisHandle->setPitchObjectParam(2,
    CSOLOISTPitchSynthesizerIf::kPitchObjectSmoothOutPitch, 8);
```

alternatively we can get the data structure of pitch objects and do some external manipulation

```
//stSOLOISTPitchObjectData stTmp1, stTmp2;
//double length;

//pcSynthesisHandle->getPitchObjectData(4, stTmp1);

//length = stTmp1.dOrigEndTimeInSec - stTmp1.dOrigStartTimeInSec;

//pcSynthesisHandle->splitPitchObject(4, length*0.5);

//pcSynthesisHandle->getPitchObjectData(4, stTmp1);

//pcSynthesisHandle->getPitchObjectData(5, stTmp2);

//length = stTmp1.dOrigEndTimeInSec - stTmp1.dOrigStartTimeInSec;

//pcSynthesisHandle->removeAllPitchObjects(); // remove all notes
```

```

//stTmp1.dCurEndTimeInSec -= stTmp1.dCurStartTimeInSec; //move note
//to the beginning
//stTmp1.dCurStartTimeInSec -= stTmp1.dCurStartTimeInSec;

//stTmp2.dCurEndTimeInSec += (stTmp1.dCurEndTimeInSec -
//stTmp2.dCurStartTimeInSec); // move 2. note to the end of 1. note
//stTmp2.dCurStartTimeInSec += (stTmp1.dCurEndTimeInSec -
//stTmp2.dCurStartTimeInSec);

//stTmp1.fCurPitch -= 1.0;
//pitch shift by a -1 semitones
//stTmp2.fCurPitch -= 1.0;
//pitch shift by a -1 semitones
//pcSynthesisHandle->addPitchObject(stTmp1);
//pcSynthesisHandle->addPitchObject(stTmp2);
//
//length = stTmp2.dCurEndTimeInSec - stTmp2.dCurStartTimeInSec;

//stTmp2.dCurStartTimeInSec = stTmp2.dCurEndTimeInSec;
//copy the 2. note and reposition it to the end of itself

//stTmp2.dCurEndTimeInSec = stTmp2.dCurStartTimeInSec + 3.5*length;
//stretch it by a factor of 2.5
//stTmp2.fCurPitch -= 1.0;
//pitch shift by a -1 semitones

//pcSynthesisHandle->addPitchObject(stTmp2);

```

For the start the direct manipulation recommended, the external manipulation would only be used if really necessary.

Now we set our playback engine to the start position

```
pcSynthesisHandle->setTimePositionInSec(0);
```

and do the synthesis loop

```

while(((float)iCurrentFrame * _OUTPUT_BUFFER_SIZE)/(float)pCAudioReader->
getSampleRate()) < pcSynthesisHandle->getLengthInSec())
{

// start a simple time measurement
clStartTime = clock();

#if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined(WIN32))
_controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW |
_EM_UNDERFLOW | _EM_DENORMAL), _MCW_EM);
#endif // #ifndef WITHOUT_EXCEPTIONS

// do the processing
pcSynthesisHandle->processData( apfProcessOutputData,
_OUTPUT_BUFFER_SIZE);

if (iError)
{
fprintf(stderr, "élastique Process Error No.:%d\n", iError);
}
}

```

```
        break;
    }

    // update time measurement
    clTotalTime += clock() - clStartTime;

    // Write stretched data to output file buffer
    pfOutputFile->Write(apfProcessOutputData, _OUTPUT_BUFFER_SIZE);

    // update the command line processing info
    CLShowProcessTime(iCurrentFrame++, iSampleRate, iNumOfInFrames);
} // while (bReadNextFrame)
```

After the successful processing, the instances can be destroyed.

```
CPitchMarksIf::DestroyInstance(pcPitchMarksHandle);
CSOLOISTPitchAnalysisIf::DestroyInstance(pcAnalysisHandle);
CSOLOISTPitchSynthesizerIf::DestroyInstance(pcSynthesisHandle);
```

1.3 Delivered Files (example project)

1.3.1 File Structure

1.3.1.1 Documentation

This documentation and all other documentation can be found in the directory **./doc**.

1.3.1.2 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.

1.3.1.3 Source Files

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

1.3.1.4 Include Files

Include files can be found in **./incl**.

1.3.1.5 Resource Files

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

1.3.1.6 Library Files

The directory **./lib** is for used and built libraries.

1.3.1.7 Binary Files

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

1.3.1.8 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`.

1.4 Coding Style minimal overview

Variable names have a preceding letter indicating their types:

unsigned:	u
pointer:	p
array:	a
class:	C
bool:	b
char:	c
short (int16):	s
int (int32):	i
__int64:	l
float (float32):	f
double (float64):	d
class/struct:	c

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

1.5 Command Line Usage Example

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

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

```
soloistTestCL 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):

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

1.6 Support

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

[zplane.development](#)

tim flohrer

katzbachstr. 21

D-10965 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:

stPitch	Structure containing the pitch information for the single pitches	21
stPitchSegmentationResult	Structure containing the result of the segmentation	22
stSOLOISTPitchObjectData	Structure that keeps the properties of each pitch object aka note object	23
CAudioReader	Virtual base class for audio file handling, needs to be inherited in order to give the synthesis access to the audio data	27
CSOLOISTPitchAnalysisIf	This class does the PSOLA analysis and pitch segmentation	30
CSOLOISTPitchSynthesizerIf	Interface for the Pitch synthesizer	39

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

PSOLAAPI.h	51
SOLOISTPitchAnalysisAPI.h	51

[SOLOISTPitchSynthesizerAPI.h](#)

52

4 Class Documentation

4.1 `_stPitch_` Struct Reference

Structure containing the pitch information for the single pitches.

```
#include <SOLOISTPitchAnalysisAPI.h>
```

Public Attributes

- float `fPitch`
The pitch of the corresponding pitchmark.
- float `fVelocity`
The velocity of the corresponding pitchmark. The max. velocity is normalized to 1.0.
- int `bValidPitch`
Give information if the corresponding pitchmark is a pitch or not.
- int `iSamplePos`
The position of the pitch in sample frames.

4.1.1 Detailed Description

Structure containing the pitch information for the single pitches.

Pitches are spaced in time according to the pitch. The indexes are equivalent to the indexes of the pitchmarks.

Definition at line 101 of file SOLOISTPitchAnalysisAPI.h.

4.1.2 Member Data Documentation

4.1.2.1 `int _stPitch_::bValidPitch`

Give information if the corresponding pitchmark is a pitch or not.

Definition at line 117 of file SOLOISTPitchAnalysisAPI.h.

4.1.2.2 `float _stPitch_::fPitch`

The pitch of the corresponding pitchmark.

Definition at line 107 of file SOLOISTPitchAnalysisAPI.h.

4.1.2.3 `float _stPitch_::fVelocity`

The velocity of the corresponding pitchmark. The max. velocity is normalized to 1.0.

Definition at line 112 of file SOLOISTPitchAnalysisAPI.h.

4.1.2.4 `int _stPitch_::iSamplePos`

The position of the pitch in sample frames.

Definition at line 122 of file `SOLOISTPitchAnalysisAPI.h`.

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

- [SOLOISTPitchAnalysisAPI.h](#)

4.2 `_stPitchSegmentationResult_ Struct Reference`

Structure containing the result of the segmentation.

```
#include <SOLOISTPitchAnalysisAPI.h>
```

Public Attributes

- float `fPitch`
The estimated pitch of the segment.
- float `fVelocity`
The estimated velocity of the segment.
- int `iLength`
The length of the segment in sample frames.
- int `iStartSamplePos`
The start position of the segment in sample frames.
- int `iEndSamplePos`
The end position of the segment in sample frames.
- int `iStartIdx`
The index of the pitchmark corresponding to the start position of the segment.
- int `iStopIdx`
The index of the pitchmark corresponding to the end position of the segment.

4.2.1 Detailed Description

Structure containing the result of the segmentation.

It contains the estimated pitch and velocity, as well as the beginning and end of the segment in terms of sample position and in terms of referenced pitchmark indexes.

Definition at line 50 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2 Member Data Documentation

4.2.2.1 `float _stPitchSegmentationResult_::fPitch`

The estimated pitch of the segment.

Definition at line 56 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.2 `float _stPitchSegmentationResult_::fVelocity`

The estimated velocity of the segment.

Definition at line 61 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.3 `int _stPitchSegmentationResult_::iEndSamplePos`

The end position of the segment in sample frames.

Definition at line 77 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.4 `int _stPitchSegmentationResult_::iLength`

The length of the segment in sample frames.

Definition at line 67 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.5 `int _stPitchSegmentationResult_::iStartIdx`

The index of the pitchmark corresponding to the start position of the segment.

Definition at line 82 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.6 `int _stPitchSegmentationResult_::iStartSamplePos`

The start position of the segment in sample frames.

Definition at line 72 of file `SOLOISTPitchAnalysisAPI.h`.

4.2.2.7 `int _stPitchSegmentationResult_::iStopIdx`

The index of the pitchmark corresponding to the end position of the segment.

Definition at line 87 of file `SOLOISTPitchAnalysisAPI.h`.

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

- [SOLOISTPitchAnalysisAPI.h](#)

4.3 `_stSOLOISTPitchObjectData_` Struct Reference

structure that keeps the properties of each pitch object aka note object

```
#include <SOLOISTPitchSynthesizerAPI.h>
```

Public Attributes

- float `fDrift`
determines the variation around the average pitch per note (1.0 == 100% == original) default = 1.0
- float `fFormantFactor`
shifts the formants for one note as factor (semitones to factor: $\text{pow}(2.0F, \text{semitone_value}/12.0F)$) default = 0.0

- float `fVolumeFactor`
determines the volume of the note (1.0 == 100% == original) default = 1.0
- float `fPitchSmoothInTimeInPercent`
time value in percent (seen from start of note and relative to the note length) of the pitch smooth in time. This is used for smoothing of the transition between notes when the pitch is changed. default = set automatically
- float `fPitchSmoothOutTimeInPercent`
time value in percent (seen from end of the note and relative to the note length) of the pitch smooth out time. This is used for smoothing of the transition between notes when the pitch is changed. default = set automatically
- float `fPitchSmoothIn`
additional smooth in shift in semitones default = 0.0
- float `fPitchSmoothOut`
additional smooth in shift in semitones default = 0.0
- float `fVolumeFadeInTimeInPercent`
initial time value in percent (seen from start of note and relative to the note length) of the volume fade-in time. default = 0.0
- float `fVolumeFadeOutTimeInPercent`
initial time value in percent (seen from end of the note and relative to the note length) of the volume fade-out time. default = 0.0
- float `fVolumeFadeInGain`
gain of the fade-in in percent (1.0 == 100% == original) default = 1.0
- float `fVolumeFadeOutGain`
gain of the fade-out in percent (1.0 == 100% == original) default = 1.0
- float `fCurPitch`
current pitch in semitones default = set automatically
- double `dCurStartTimeInSec`
current note onset time in seconds. This affects the stretch of a note. default = set automatically
- double `dCurEndTimeInSec`
current note offset time in seconds. This affects the stretch of a note. default = set automatically
- double `dCurValidNoteEndTimeInSec`
a note can consist of a valid (voiced) and invalid (unvoiced/silence) part. This value is just for information and is set automatically depending on `dOrigValidNoteEndTimeInSec`. default = set automatically
- float `fOrigPitch`
original pitch in semitones (don't edit that) default = set automatically
- float `fOrigVelocity`
original velocity in percent (1.0 == 100% == fullscale) default = set automatically
- double `dOrigStartTimeInSec`
original note onset time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically
- double `dOrigEndTimeInSec`
original note offset time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically

- double `dOrigValidNoteEndTimeInSec`
original valid note end time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically

4.3.1 Detailed Description

structure that keeps the properties of each pitch object aka note object

Definition at line 10 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2 Member Data Documentation

4.3.2.1 double `_stSOLOISTPitchObjectData_::dCurEndTimeInSec`

current note offset time in seconds. This affects the stretch of a note. default = set automatically

Definition at line 104 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2.2 double `_stSOLOISTPitchObjectData_::dCurStartTimeInSec`

current note onset time in seconds. This affects the stretch of a note. default = set automatically

Definition at line 104 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2.3 double `_stSOLOISTPitchObjectData_::dCurValidNoteEndTimeInSec`

a note can consist of a valid (voiced) and invalid (unvoiced/silence) part. This value is just for information and is set automatically depending on `dOrigValidNoteEndTimeInSec`. default = set automatically

Definition at line 104 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2.4 double `_stSOLOISTPitchObjectData_::dOrigEndTimeInSec`

original note offset time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically

Definition at line 141 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2.5 double `_stSOLOISTPitchObjectData_::dOrigStartTimeInSec`

original note onset time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically

Definition at line 141 of file `SOLOISTPitchSynthesizerAPI.h`.

4.3.2.6 double `_stSOLOISTPitchObjectData_::dOrigValidNoteEndTimeInSec`

original valid note end time in seconds. Don't edit that unless you want to change the original segmentation and you know what you're doing default = set automatically

Definition at line 141 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.7 `float _stSOLOISTPitchObjectData_::fCurPitch`

current pitch in semitones default = set automatically

Definition at line 97 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.8 `float _stSOLOISTPitchObjectData_::fDrift`

determines the variation around the average pitch per note (1.0 == 100% == original) default = 1.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.9 `float _stSOLOISTPitchObjectData_::fFormantFactor`

shifts the formants for one note as factor (semitones to factor: $\text{pow}(2.0F, \text{semitone_value}/12.0F)$) default = 0.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.10 `float _stSOLOISTPitchObjectData_::fOrigPitch`

original pitch in semitones (don't edit that) default = set automatically

Definition at line 128 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.11 `float _stSOLOISTPitchObjectData_::fOrigVelocity`

original velocity in percent (1.0 == 100% == fullscale) default = set automatically

Definition at line 128 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.12 `float _stSOLOISTPitchObjectData_::fPitchSmoothIn`

additional smooth in shift in semitones default = 0.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.13 `float _stSOLOISTPitchObjectData_::fPitchSmoothInTimeInPercent`

time value in percent (seen from start of note and relative to the note length) of the pitch smooth in time. This is used for smoothing of the transition between notes when the pitch is changed. default = set automatically

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.14 `float _stSOLOISTPitchObjectData_::fPitchSmoothOut`

additional smooth in shift in semitones default = 0.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.15 float _stSOLOISTPitchObjectData_::fPitchSmoothOutTimeInPercent

time value in percent (seen from end of the note and relative to the note length) of the pitch smooth out time. This is used for smoothing of the transition between notes when the pitch is changed. default = set automatically

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.16 float _stSOLOISTPitchObjectData_::fVolumeFactor

determines the volume of the note (1.0 == 100% == original) default = 1.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.17 float _stSOLOISTPitchObjectData_::fVolumeFadeInGain

gain of the fade-in in percent (1.0 == 100% == original) default = 1.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.18 float _stSOLOISTPitchObjectData_::fVolumeFadeInTimeInPercent

initial time value in percent (seen from start of note and relative to the note length) of the volume fade-in time. default = 0.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.19 float _stSOLOISTPitchObjectData_::fVolumeFadeOutGain

gain of the fade-out in percent (1.0 == 100% == original) default = 1.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

4.3.2.20 float _stSOLOISTPitchObjectData_::fVolumeFadeOutTimeInPercent

initial time value in percent (seen from end of the note and relative to the note length) of the volume fade-out time. default = 0.0

Definition at line 19 of file SOLOISTPitchSynthesizerAPI.h.

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

- [SOLOISTPitchSynthesizerAPI.h](#)

4.4 CAudioReader Class Reference

virtual base class for audio file handling, needs to be inherited in order to give the synthesis access to the audio data

```
#include <SOLOISTPitchSynthesizerAPI.h>
```

Public Member Functions

- [CAudioReader \(\)](#)

constructor

- virtual `~CAudioReader()`

destructor

- virtual int `seekPos(int iPositionInFrames)=0`
set a new file read position in sample frames
- virtual int `readAudioData(float **ppfAudioData, int iNumOfFramestoRead)=0`

*reads the actual audio data into a n-dimensional array of channels (usually 1 or 2).
If the the number of frames requested exceeds the number of frames available the
remaining frames must be set to zero*

- virtual int `getLengthInFrames()`=0
method to retrieve the length of the audio data in sample frames
- virtual int `getNumOfChannels()`=0
method to retrieve the number of channels
- virtual int `getSampleRate()`=0
method to retrieve the sample rate

4.4.1 Detailed Description

virtual base class for audio file handling, needs to be inherited in order to give the synthesis access to the audio data

Definition at line 164 of file SOLOISTPitchSynthesizerAPI.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 CAudioReader::CAudioReader() [inline]

constructor

Definition at line 172 of file SOLOISTPitchSynthesizerAPI.h.

```
{};
```

4.4.2.2 virtual CAudioReader::~~CAudioReader() [inline, virtual]

destructor

Definition at line 177 of file SOLOISTPitchSynthesizerAPI.h.

```
{};
```

4.4.3 Member Function Documentation

4.4.3.1 virtual int CAudioReader::getLengthInFrames() [pure virtual]

method to retrieve the length of the audio data in sample frames

Returns

length in sample frames

4.4.3.2 virtual int CAudioReader::getNumOfChannels () [pure virtual]

method to retrieve the number of channels

Returns

the number of channels

4.4.3.3 virtual int CAudioReader::getSampleRate () [pure virtual]

method to retrieve the sample rate

Returns

the samplerate

4.4.3.4 virtual int CAudioReader::readAudioData (float ** *ppfAudioData*, int *iNumOfFramestoRead*) [pure virtual]

reads the actual audio data into a n-dimensional array of channels (usually 1 or 2). If the the number of frames requested exceeds the number of frames available the remaining frames must be set to zero

Parameters

<i>ppfAudio-Data</i>	double pointer containing the audio data. The audio buffer is allocated as <code>ppfAudioData[channels][sampleframes]</code>
<i>iNumOf-Framesto-Read</i>	the number of same frames to be read

Returns

NULL if operation was successful

4.4.3.5 virtual int CAudioReader::seekPos (int *iPositionInFrames*) [pure virtual]

set a new file read position in sample frames

Parameters

<i>iPositionIn-Frames</i>	the desired position in sampleframes
---------------------------	--------------------------------------

Returns

NULL if operation was successful

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

- [SOLOISTPitchSynthesizerAPI.h](#)

4.5 CSOLOISTPitchAnalysisIf Class Reference

This class does the PSOLA analysis and pitch segmentation.

```
#include <SOLOISTPitchAnalysisAPI.h>
```

Public Member Functions

- virtual `~CSOLOISTPitchAnalysisIf ()`
- virtual int `ProcessData (float **ppSampleData, int iNumOfFrames)=0`
Does the analysis processing, is basically the same as the SOLOIST Analysis.
- virtual int `SetEOF ()=0`
Tells the analysis that the audio files has ended and processes the remaining internal buffers.
- virtual int `Reset ()=0`
Resets analysis to original state.
- virtual int `DoSegmentation ()=0`
Does the segmentation processing, may be called multiple times, e.g. if the a section has been reanalysed. May only be called after `ProcessData()` and `SetEOF()` are finished.
- virtual int `DoSegmentation (int iStep)=0`
Same as `DoSegmentation`, but splits it into 9 (0..8) steps, the last step (8) is the clean up of detected pitches and may be omitted.
- virtual int `GetSegmentationResult (stPitchSegmentationResult *&pstSegmentationResults)=0`
Retrieves a list of segments, generated by `DoSegmentation()`
- virtual int `GetPitchForSection (float &fPitch, float &fVelocity, int iStartIdx, int iEndIdx)=0`
If the pitch for a certain section is to be recalculated, call this function.
- virtual int `ReAnalyseSection (float **ppSampleData, int iNumOfFrames, int iStartIdx, float fDesiredPitch)=0`
May be used to reanalyse a section of already analyzed data with a given pitch estimate.
- virtual int `GetPitchResult (stPitch *&pstPitch)=0`
Retrieves a list of pitches generated. may only be called after calling `DoSegmentation`.
- virtual float `GetAvgPitchDeviation ()=0`
Returns the average pitch deviation of the optimal pitch. May be used in order to generate MIDI output. The Result should be added to the pitch values in segment list before rounding to MIDI pitches.

- virtual int **GetKey** (int iKeyOffset=0)=0
returns the estimated key idx ranging from 0-23 where 0-11 are all major key starting from c and 12-23 are all minor keys
- virtual void **PutPitches** (stPitch *pstPitch, int uiNumOfPitches)=0
*pitches retrieved by *GetPitchResult()* may be put back that way*
- virtual void **PutPitchSegments** (stPitchSegmentationResult *pstSegmentationResults, int uiNumOfSegments)=0
*pitch segments retrieved by *GetSegmentationResult()* may be put back that way*
- virtual int **ConvertPitchMarks2PitchArrays** ()=0
*must be called before *GetPitchResult**
- virtual void **SetDistancePitchJoinMaxLimit** (float fVal)=0
sets the max pitch distance between two pitches that allows them to be joined (default is 0.75)
- virtual float **GetDistancePitchJoinMaxLimit** ()=0
get the current maximum pitch distance value
- virtual void **SetLengthNoteMinLimit** (float fVal)=0
sets the min note length in sec, notes shorter than that value are discarded (default is 0.042sec)
- virtual float **GetLengthNoteMinLimit** ()=0
get the current min note length
- virtual void **SetDistanceNoteJoinMaxLimit** (float fVal)=0
sets the max time distance between two notes that allows them to be joined (default is 0.092sec)
- virtual float **GetDistanceNoteJoinMaxLimit** ()=0
get the current max time distance to join
- virtual void **SetLengthNoteJoinMaxLimit** (float fVal)=0
sets the max note length of a pitch that allows it to be joined (default is 0.184sec)
- virtual float **GetLengthNoteJoinMaxLimit** ()=0
get the current the max note length to join
- virtual int **NormalizeEnergy** ()=0
normalizes the pitch energies
- virtual float **GetAvgPitchForPitches** (_stPitch_ *pstPitches, int iStartIdx, int i-EndIdx)=0
returns the average pitch for a set of pitches passed
- virtual void **SetTonalityThreshold** (float fThreshold)=0
sets the threshold for tonality detection, default is 5 this value is on some kind of log scale.
- virtual float **GetTonalityThreshold** ()=0
returns the current threshold
- virtual int **GetDataChunkSizeinBytes** ()=0
- virtual void **GetDataChunk** (void *pPreAllocatedDataChunk)=0
- virtual bool **SetDataChunk** (void *pDataChunk, int iDataChunkSizeInBytes)=0

Static Public Member Functions

- static int [CreateInstance](#) ([CSOLOISTPitchAnalysisIf](#) *pCInstance, [CPitchMarksIf](#) *pcPitchMarks, float fSampleRate, int iNumOfChannels)
Creates an instance of the CSOLOISTOffLineAnalysis class.
- static int [DestroyInstance](#) ([CSOLOISTPitchAnalysisIf](#) *pCInstance)
Destroys an instance of the class.

4.5.1 Detailed Description

This class does the PSOLA analysis and pitch segmentation.

Generally this class should be used as follows:

- generate [PitchMark](#) class and pass it to the instance of the class when creating it with [CreateInstance](#). One may also pass a previously save [PitchMark](#) class, please refer to the SOLOIST SDK documentation for this. In that case, please skip the [ProcessData\(\)](#) and [SetEOF\(\)](#) methods.
- call [ProcessData\(\)](#) until your input audio is finished
- call [SetEOF\(\)](#) in order to tell the class that audio has ended
- call [DoSegmentation\(\)](#) in order to kick off the segmentation processing
- call [GetSegmentationResult](#) and [GetPitchResult\(\)](#) in order to retrieve the results
In case of wrong analysis results, use [ReAnalyseSection\(\)](#) in order to let the user correct the result For separating or combining segments, use [GetPitchForSection\(\)](#) to get the overall pitch of the new section. Those new segments must be handled by the application, the internal segment list is not updated.

Definition at line 142 of file [SOLOISTPitchAnalysisAPI.h](#).

4.5.2 Constructor & Destructor Documentation

4.5.2.1 `virtual CSOLOISTPitchAnalysisIf::~CSOLOISTPitchAnalysisIf ()`
[inline, virtual]

Definition at line 145 of file [SOLOISTPitchAnalysisAPI.h](#).

```
{};
```

4.5.3 Member Function Documentation

4.5.3.1 `virtual int CSOLOISTPitchAnalysisIf::ConvertPitchMarks2PitchArrays ()`
[pure virtual]

must be called before [GetPitchResult](#)

4.5.3.2 `static int CSOLOISTPitchAnalysisIf::CreateInstance (CSOLOISTPitchAnalysisIf *& pCInstance, CPitchMarkslf * pcPitchMarks, float fSampleRate, int iNumOfChannels) [static]`

Creates an instance of the CSOLOISTOffLineAnalysis class.

Parameters

<i>pCInstance</i>	Pointer to the new instance of the class.
<i>pcPitch-Marks</i>	Pointer to a previously created instance of the CPitchMark class.
<i>fSampleRate</i>	The samplerate of the audio to be analyzed.
<i>iNumOf-Channels</i>	The number of channels of the audio to be analyzed.

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.3 `static int CSOLOISTPitchAnalysisIf::DestroyInstance (CSOLOISTPitchAnalysisIf *& pCInstance) [static]`

Destroys an instance of the class.

Parameters

<i>pCInstance</i>	Pointer to the instance to be destroyed
-------------------	---

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.4 `virtual int CSOLOISTPitchAnalysisIf::DoSegmentation () [pure virtual]`

Does the segmentation processing, may be called multiple times, e.g. if the a section has been reanalysed. May only be called after `ProcessData()` and `SetEOF()` are finished.

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.5 `virtual int CSOLOISTPitchAnalysisIf::DoSegmentation (int iStep) [pure virtual]`

Same as `DoSegmentation`, but splits it into 9 (0..8) steps, the last step (8) is the clean up of detected pitches and may be omitted.

Parameters

<i>iStep</i>	the current segmentation step, may not be called out of order
--------------	---

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.6 `virtual float CSOLOISTPitchAnalysisIf::GetAvgPitchDeviation ()`
`[pure virtual]`

Returns the average pitch deviation of the optimal pitch. May be used in order to generate MIDI output. The Result should be added to the pitch values in segment list before rounding to MIDI pitches.

Returns

The average pitch deviation

4.5.3.7 `virtual float CSOLOISTPitchAnalysisIf::GetAvgPitchForPitches (`
`_stPitch_ * pstPitches, int iStartIdx, int iEndIdx)` `[pure virtual]`

returns the average pitch for a set of pitches passed

Parameters

<i>pstPitches</i>	an array of pitches to be analyzed
<i>iStartIdx</i>	the start offset into the array, usually set to 0
<i>iEndIdx</i>	the end index of the array, usually set to the length of the array

Returns

the average pitch in terms of MIDI notes

4.5.3.8 `virtual void CSOLOISTPitchAnalysisIf::GetDataChunk (void *`
`pPreAllocatedDataChunk)` `[pure virtual]`

4.5.3.9 `virtual int CSOLOISTPitchAnalysisIf::GetDataChunkSizeinBytes ()`
`[pure virtual]`

4.5.3.10 `virtual float CSOLOISTPitchAnalysisIf::GetDistanceNoteJoinMaxLimit`
`()` `[pure virtual]`

get the current max time distance to join

4.5.3.11 `virtual float CSOLOISTPitchAnalysisIf::GetDistancePitchJoinMaxLimit`
`()` `[pure virtual]`

get the current maximum pitch distance value

4.5.3.12 `virtual int CSOLOISTPitchAnalysisIf::GetKey (int iKeyOffset = 0)`
 [pure virtual]

returns the estimated key idx ranging from 0-23 where 0-11 are all major key starting from c and 12-23 are all minor keys

Parameters

<i>iKeyOffset</i>	if idx == 0 or idx == 12 is not supposed to be C, one may define an offset here
-------------------	---

4.5.3.13 `virtual float CSOLOISTPitchAnalysisIf::GetLengthNoteJoinMaxLimit ()`
 [pure virtual]

get the current the max note length to join

4.5.3.14 `virtual float CSOLOISTPitchAnalysisIf::GetLengthNoteMinLimit ()`
 [pure virtual]

get the current min note length

4.5.3.15 `virtual int CSOLOISTPitchAnalysisIf::GetPitchForSection (float & fPitch, float & fVelocity, int iStartIdx, int iEndIdx)` [pure virtual]

If the pitch for a certain section is to be recalculated, call this function.

Parameters

<i>fPitch</i>	returns the pitch estimated for the section.
<i>fVelocity</i>	returns the velocity for the section
<i>iStartIdx</i>	index of the pitchmark where the section starts
<i>iEndIdx</i>	index of the pitchmark where the section ends

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.16 `virtual int CSOLOISTPitchAnalysisIf::GetPitchResult (stPitch *& pstPitch)` [pure virtual]

Retrieves a list of pitches generated. may only be called after calling `DoSegmentation`.

Parameters

<i>pstPitch</i>	Pointer to a list of pitches
-----------------	------------------------------

Returns

The number of pitches in the pitch list

4.5.3.17 `virtual int CSOLOISTPitchAnalysisIf::GetSegmentationResult (stPitchSegmentationResult *& pstSegmentationResults)` [pure virtual]

Retrieves a list of segments, generated by [DoSegmentation\(\)](#)

Parameters

<i>pst-Segmentation-Results</i>	The pointer to the segment list
---------------------------------	---------------------------------

Returns

The number of segments in the segment list

4.5.3.18 `virtual float CSOLOISTPitchAnalysisIf::GetTonalityThreshold ()` [pure virtual]

returns the current threshold

4.5.3.19 `virtual int CSOLOISTPitchAnalysisIf::NormalizeEnergy ()` [pure virtual]

normalizes the pitch energies

4.5.3.20 `virtual int CSOLOISTPitchAnalysisIf::ProcessData (float ** ppSampleData, int iNumOfFrames)` [pure virtual]

Does the analysis processing, is basically the same as the SOLOIST Analysis.

Parameters

<i>ppSample-Data</i>	Double pointer to input audio sample data. Audio data is arranged as in VST
<i>iNumOf-Frames</i>	Number of frames passed to the process function

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.21 `virtual void CSOLOISTPitchAnalysisIf::PutPitches (stPitch * pstPitch, int uiNumOfPitches) [pure virtual]`

pitches retrieved by [GetPitchResult\(\)](#) may be put back that way

4.5.3.22 `virtual void CSOLOISTPitchAnalysisIf::PutPitchSegments (stPitchSegmentationResult * pstSegmentationResults, int uiNumOfSegments) [pure virtual]`

pitch segments retrieved by [GetSegmentationResult\(\)](#) may be put back that way

4.5.3.23 `virtual int CSOLOISTPitchAnalysisIf::ReAnalyseSection (float ** ppSampleData, int iNumOfFrames, int iStartIdx, float fDesiredPitch) [pure virtual]`

May be used to reanalyse a section of already analyzed data with a given pitch estimate.

Parameters

<i>ppSample-Data</i>	Double pointer to the audiodata starting at the sample position of the pitchmark at the index where the section starts
<i>iNumOf-Frames</i>	The number of frames to reanalyzed
<i>iStartIdx</i>	The index of the pitchmark where the section to reanalyzed starts
<i>fDesired-Pitch</i>	The user defined pitch estimate for the reanalysis section

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.24 `virtual int CSOLOISTPitchAnalysisIf::Reset () [pure virtual]`

Resets analysis to original state.

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.25 `virtual bool CSOLOISTPitchAnalysisIf::SetDataChunk (void * pDataChunk, int iDataChunkSizeInBytes) [pure virtual]`

4.5.3.26 `virtual void CSOLOISTPitchAnalysisIf::SetDistanceNote.JoinMaxLimit (float fVal) [pure virtual]`

sets the max time distance between two notes that allows them to be joined (default is 0.092sec)

Parameters

<i>fVal</i>	the time distance in sec
-------------	--------------------------

Returns

Write description of return value here.

4.5.3.27 virtual void CSOLOISTPitchAnalysisIf::SetDistancePitchJoinMaxLimit (float *fVal*) [pure virtual]

sets the max pitch distance between two pitches that allows them to be joined (default is 0.75)

Parameters

<i>fVal</i>	the max distance in midi pitch (1.0 = semitone)
-------------	---

Returns

Write description of return value here.

4.5.3.28 virtual int CSOLOISTPitchAnalysisIf::SetEOF () [pure virtual]

Tells the analysis that the audio files has ended and processes the remaining internal buffers.

Returns

`_NO_ERROR == 0` if no error occurred otherwise refer to `zErrors.h`

4.5.3.29 virtual void CSOLOISTPitchAnalysisIf::SetLengthNoteJoinMaxLimit (float *fVal*) [pure virtual]

sets the max note length of a pitch that allows it to be joined (default is 0.184sec)

Parameters

<i>fVal</i>	the max distance in midi pitch (1.0 = semitone)
-------------	---

Returns

Write description of return value here.

4.5.3.30 virtual void CSOLOISTPitchAnalysisIf::SetLengthNoteMinLimit (float *fVal*) [pure virtual]

sets the min note length in sec, notes shorter than that value are discarded (default is 0.042sec)

Parameters

<i>fVal</i>	the min note length in sec
-------------	----------------------------

Returns

Write description of return value here.

4.5.3.31 virtual void CSOLOISTPitchAnalysisIf::SetTonalityThreshold (float *fThreshold*) [pure virtual]

sets the threhold for tonality detection, default is 5 this value is on some kind of log scale.

Parameters

<i>fThreshold</i>	the threshold (useful ranges are from 3 to 9)
-------------------	---

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

- [SOLOISTPitchAnalysisAPI.h](#)

4.6 CSOLOISTPitchSynthesizerIf Class Reference

interface for the Pitch synthesizer

```
#include <SOLOISTPitchSynthesizerAPI.h>
```

Public Types

- enum `_errorCode` { `kNoError`, `kInvalidParameter`, `kIndexOutOfBounds`, `numOfErrorCodes` }
- enum `PitchClass` { `C`, `CSharp`, `D`, `DSharp`, `E`, `F`, `FSharp`, `G`, `GSharp`, `A`, `ASharp`, `B`, `kNumOfPitchClasses` }
- enum `_ePitchObjectParam` { `kPitchObjectPitch`, `kPitchObjectSmoothInTimeInPercent`, `kPitchObjectSmoothOutTimeInPercent`, `kPitchObjectSmoothInPitch`, `kPitchObjectSmoothOutPitch`, `kPitchObjectVolumeFadeInTimeInPercent`, `kPitchObjectVolumeFadeOutTimeInPercent`, `kPitchObjectVolumeFadeInGain`, `kPitchObjectVolumeFadeOutGain`, `kPitchObjectFormantShift`, `kPitchObjectDrift`, `kPitchObjectVolume`, `numOfPitchObjectParams` }
- enum `_eGlobalParam` { `kGlobalStretch`, `kGlobalPitchShift`, `kGlobalFormantShift`, `kGlobalTune`, `kGlobalDrift`, `kGlobalTransitionFactor`, `numOfGlobalParams` }

Public Member Functions

- virtual void `processData` (float **ppfAudioData, int iNumOfFramestoRead)=0
generates the audio output at the current position

- virtual void `setTimePositionInSec` (double newReadPosition)=0
set next play position in seconds
- virtual double `getTimePositionInSec` () const =0
retrieve the current play position
- virtual double `getLengthInSec` () const =0
retrieve the current playback length of the synthesizer object. This takes all editing (e.g. stretching) into account.
- virtual float `getPitchForTimeInSec` (double dTime)=0
retrieves the pitch value of the pitch curve a a given point in time. Useful for drawing a pitch curve.
- virtual float `getEnvelopeForTimeInSec` (double dTime)=0
retrieves the volume value of the audio at a given point in time. Useful for drawing the volume envelope.
- virtual int `getNumOfPitchObjects` ()=0
retrieves the number of pitch (note) objects
- virtual `_errorCode removePitchObject` (int iPitchObjectIndex)=0
removes a pitch object
- virtual void `removeAllPitchObjects` ()=0
removes all pitch objects
- virtual void `addPitchObject` (stSOLOISTPitchObjectData &stInitialPitchObjectData)=0
adds a pitch object, the object is put into the right order automatically. Overlapping objects are not allowed but not checked. So, it is in the users responsibility to avoid overlapping.
- virtual `_errorCode splitPitchObject` (int iPitchObjectIndex, double dRelativeCutPositionFromStartOfPitchObject)=0
splits a pitch object into two
- virtual `_errorCode joinPitchObjects` (int iPitchObjectIndex1, int iPitchObjectIndex2)=0
joins two adjacent pitch objects
- virtual `_errorCode getPitchObjectData` (int iPitchObjectIndex, stSOLOISTPitchObjectData &ObjData)=0
get the complete property structure of a given pitch object
- virtual `_errorCode setPitchObjectData` (int iPitchObjectIndex, stSOLOISTPitchObjectData &ObjData, bool bUpdateSurroundingObjects=true)=0
set a complete property structure of a given pitch object
- virtual `_errorCode setPitchObjectBounds` (int iPitchObjectIndex, double &dNewStartPos, double &dNewEndPos, bool bUpdateSurroundingObjects=true)=0
change the onset and offset of a given pitch object. This affects the stretch factor of the pitch object. Onset and offset of the given pitch object must not exceed the corresponding onset and offset of the surrounding objects.
- virtual `_errorCode quantizePitchObjectsToScale` (std::vector< PitchClass > &Scale)=0
quantize all pitch objects to the notes given in the Scale vector
- virtual `_errorCode getPitchObjectBounds` (int iPitchObjectIndex, double &dStartPos, double &dEndPos)=0

- retrieve the onset and offset position in seconds of a given pitch object.*

 - virtual `_errorCode setPitchObjectParam` (int iPitchObjectIndex, `_ePitchObjectParam` eParam, float fValue)=0
 - set a parameter for a give pitch object directly.*
 - virtual float `getPitchObjectParam` (int iPitchObjectIndex, `_ePitchObjectParam` eParam) const =0
 - retrieve a parameter for a give pitch object directly.*
 - virtual `_errorCode setGlobalParam` (`_eGlobalParam` eParam, float fValue)=0
 - set a global parameter for the whole audio file.*
 - virtual float `getGlobalParam` (`_eGlobalParam` eParam) const =0
 - retrieve a global parameter for the whole audio file.*

Static Public Member Functions

- static int `CreateInstance` (`CSOLOISTPitchSynthesizerIf` *&pCInstance, `CAudioReader` *const pCAudioReader, `CPitchMarksIf` *const pCPitchMarks, `CSOLOISTPitchAnalysisIf` *const pCAnalysisIf)
 - creates an instance of the Pitch synthesizer class*
- static int `DestroyInstance` (`CSOLOISTPitchSynthesizerIf` *&pCInstance)
 - destroys an instance of the Pitch synthesizer class*

4.6.1 Detailed Description

interface for the Pitch synthesizer

Definition at line 242 of file SOLOISTPitchSynthesizerAPI.h.

4.6.2 Member Enumeration Documentation

4.6.2.1 enum CSOLOISTPitchSynthesizerIf::_eGlobalParam

Enumerator:

- kGlobalStretch*** set the global stretch factor
- kGlobalPitchShift*** set the global pitch factor in semitones
- kGlobalFormantShift*** set the global formant factor in semitones
- kGlobalTune*** set the global tuning 1.0 == 100% tuning
- kGlobalDrift*** set the global drift
- kGlobalTransitionFactor*** set the global transition factor, this determines how much the drift factor affects the pitch smooth in/out regions of each pitch object
- numOfGlobalParams*** the number of global parameters

Definition at line 596 of file SOLOISTPitchSynthesizerAPI.h.

```

{
    kGlobalStretch,
    kGlobalPitchShift,
    kGlobalFormantShift,
    kGlobalTune,
    kGlobalDrift,
    kGlobalTransitionFactor,

    numOfGlobalParams
};

```

4.6.2.2 enum CSOLOISTPitchSynthesizerIf::_ePitchObjectParam

Enumerator:

kPitchObjectPitch see [_stSOLOISTPitchObjectData_::fCurPitch](#)

kPitchObjectSmoothInTimeInPercent see [_stSOLOISTPitchObjectData_::fPitchSmoothInTimeInPercent](#)

kPitchObjectSmoothOutTimeInPercent see [_stSOLOISTPitchObjectData_::fPitchSmoothOutTimeInPercent](#)

kPitchObjectSmoothInPitch see [_stSOLOISTPitchObjectData_::fPitchSmoothIn](#)

kPitchObjectSmoothOutPitch see [_stSOLOISTPitchObjectData_::fPitchSmoothOut](#)

kPitchObjectVolumeFadeInTimeInPercent see [_stSOLOISTPitchObjectData_::fVolumeFadeInTimeInPercent](#)

kPitchObjectVolumeFadeOutTimeInPercent see [_stSOLOISTPitchObjectData_::fVolumeFadeOutTimeInPercent](#)

kPitchObjectVolumeFadeInGain see [_stSOLOISTPitchObjectData_::fVolumeFadeInGain](#)

kPitchObjectVolumeFadeOutGain see [_stSOLOISTPitchObjectData_::fVolumeFadeOutGain](#)

kPitchObjectFormantShift see [_stSOLOISTPitchObjectData_::fFormantFactor](#), here not as factor but in semitones

kPitchObjectDrift see [_stSOLOISTPitchObjectData_::fDrift](#)

kPitchObjectVolume see [_stSOLOISTPitchObjectData_::fVolumeFactor](#)

numOfPitchObjectParams number of parameters

Definition at line 536 of file SOLOISTPitchSynthesizerAPI.h.

```

{
    kPitchObjectPitch,

    kPitchObjectSmoothInTimeInPercent,
    kPitchObjectSmoothOutTimeInPercent,

    kPitchObjectSmoothInPitch,
    kPitchObjectSmoothOutPitch,

    kPitchObjectVolumeFadeInTimeInPercent,

```

```
kPitchObjectVolumeFadeOutTimeInPercent,  
  
kPitchObjectVolumeFadeInGain,  
kPitchObjectVolumeFadeOutGain,  
  
kPitchObjectFormantShift,  
  
kPitchObjectDrift,  
  
kPitchObjectVolume,  
  
numOfPitchObjectParams  
};
```

4.6.2.3 enum CSOLOISTPitchSynthesizerIf::_errorCode

Enumerator:

kNoError
kInvalidParameter
kIndexOutOfBounds
numOfErrorCodes

Definition at line 283 of file SOLOISTPitchSynthesizerAPI.h.

```
{  
    kNoError,  
    kInvalidParameter,  
    kIndexOutOfBounds,  
  
    numOfErrorCodes  
};
```

4.6.2.4 enum CSOLOISTPitchSynthesizerIf::PitchClass

Enumerator:

C
CSharp
D
DSharp
E
F
FSharp
G
GSharp
A
ASharp
B

kNumOfPitchClasses

Definition at line 488 of file SOLOISTPitchSynthesizerAPI.h.

```
{
    C,
    CSharp,
    D,
    DSharp,
    E,
    F,
    FSharp,
    G,
    GSharp,
    A,
    ASharp,
    B,
    kNumOfPitchClasses
};
```

4.6.3 Member Function Documentation

4.6.3.1 virtual void CSOLOISTPitchSynthesizerIf::addPitchObject (stSOLOISTPitchObjectData & stInitialPitchObjectData) [pure virtual]

adds a pitch object, the object is put into the right order automatically. Overlapping objects are not allowed but not checked. So, it is in the users responsibility to avoid overlapping.

Parameters

<i>stInitialPitchObjectData</i>	
---------------------------------	--

4.6.3.2 static int CSOLOISTPitchSynthesizerIf::CreateInstance (CSOLOISTPitchSynthesizerIf *& pCInstance, CAudioReader *const pCAudioReader, CPitchMarksIf *const pCPitchMarks, CSOLOISTPitchAnalysisIf *const pCAnalysisIf) [static]

creates an instance of the Pitch synthesizer class

Parameters

<i>pCInstance</i>	the pointer to the created class
<i>pCAudioReader</i>	pointer to an inherited AudioReader class that performs the audio data IO
<i>pCPitchMarks</i>	pointer to the pitchmark class created by the analysis
<i>pCAnalysisIf</i>	pointer to the analysis class

Returns

`_errorCode`

4.6.3.3 `static int CSOLOISTPitchSynthesizerIf::DestroyInstance (CSOLOISTPitchSynthesizerIf *& pCInstance)` [static]

destroys an instance of the Pitch synthesizer class

Parameters

<code><i>pCInstance</i></code>	pointer to the instance to be destroyed
--------------------------------	---

Returns

an `_errorCode`

4.6.3.4 `virtual float CSOLOISTPitchSynthesizerIf::getEnvelopeForTimeInSec (double dTime)` [pure virtual]

retrieves the volume value of the audio at a given point in time. Useful for drawing the volume envelope.

Parameters

<code><i>dTime</i></code>	time in seconds
---------------------------	-----------------

Returns

the volume at the given time. 1 == fullscale

4.6.3.5 `virtual float CSOLOISTPitchSynthesizerIf::getGlobalParam (_eGlobalParam eParam) const` [pure virtual]

retrieve a global parameter for the whole audio file.

Parameters

<code><i>eParam</i></code>	the parameter to be retrieved (refer to CSOLOISTPitchSynthesizerIf::_eGlobalParam)
----------------------------	---

Returns

the value of the parameter

4.6.3.6 virtual double CSOLOISTPitchSynthesizerIf::getLengthInSec () const
 [pure virtual]

retrieve the current playback length of the synthesizer object. This takes all editing (e.g. stretching) into account.

Returns

playback length in seconds

4.6.3.7 virtual int CSOLOISTPitchSynthesizerIf::getNumOfPitchObjects ()
 [pure virtual]

retrieves the number of pitch (note) objects

Returns

number of pitch objects

4.6.3.8 virtual float CSOLOISTPitchSynthesizerIf::getPitchForTimeInSec (
double *dTime*) [pure virtual]

retrieves the pitch value of the pitch curve a a given point in time. Useful for drawing a pitch curve.

Parameters

<i>dTime</i>	time in seconds
--------------	-----------------

Returns

the pitch in semitones at the given time. Returns -1 if there is no pitch detected.

4.6.3.9 virtual _errorCode CSOLOISTPitchSynthesizerIf::getPitchObject-
Bounds (int *iPitchObjectIndex*, double & *dStartPos*, double & *dEndPos*)
 [pure virtual]

retrieve the onset and offset position in seconds of a given pitch object.

Parameters

<i>iPitch-ObjectIndex</i>	the index of the pitch object
<i>dNewStart-Pos</i>	current onset position of the pitch object
<i>dNewEnd-Pos</i>	current offset position of the pitch object

Returns`_errorCode`

4.6.3.10 `virtual _errorCode CSOLOISTPitchSynthesizerIf::getPitchObjectData (int iPitchObjectIndex, stSOLOISTPitchObjectData & ObjData)` [pure virtual]

get the complete property structure of a given pitch object

Parameters

<i>iPitch-ObjectIndex</i>	the index of the pitch object
<i>ObjData</i>	the structure for the data

Returns`_errorCode`

4.6.3.11 `virtual float CSOLOISTPitchSynthesizerIf::getPitchObjectParam (int iPitchObjectIndex, _ePitchObjectParam eParam) const` [pure virtual]

retrieve a parameter for a give pitch object directly.

Parameters

<i>iPitch-ObjectIndex</i>	the index of the pitch object
<i>eParam</i>	the parameter to be changed (refer to CSOLOISTPitchSynthesizerIf::_ePitchObjectParam)

Returns

the value of the parameter

4.6.3.12 `virtual double CSOLOISTPitchSynthesizerIf::getTimePositionInSec () const` [pure virtual]

retrieve the current play position

Returns

the current play position

4.6.3.13 `virtual _errorCode CSOLOISTPitchSynthesizerIf::joinPitchObjects (int iPitchObjectIndex1, int iPitchObjectIndex2)` [pure virtual]

joins two adjacent pitch objects

Parameters

<i>iPitch-Object-Index1</i>	index of one pitch object to join
<i>iPitch-Object-Index2</i>	index of the other pitch object to join

Returns

`_errorCode`

4.6.3.14 `virtual void CSOLOISTPitchSynthesizerIf::processData (float ** ppfAudioData, int iNumOfFramestoRead)` [pure virtual]

generates the audio output at the current position

Parameters

<i>ppfAudio-Data</i>	double pointer to the audio data to be generated. The audio buffer is allocated as <code>ppfAudioData[channels][sampleframes]</code> .
<i>iNumOf-Framesto-Read</i>	the number of frames requested

4.6.3.15 `virtual _errorCode CSOLOISTPitchSynthesizerIf::quantizePitch-ObjectsToScale (std::vector< PitchClass > & Scale)` [pure virtual]

quantize all pitch objects to the notes given in the Scale vector

Parameters

<i>Scale</i>	a list of PitchClass notes allowed
--------------	------------------------------------

Returns

`_errorCode`

4.6.3.16 `virtual void CSOLOISTPitchSynthesizerIf::removeAllPitchObjects ()` [pure virtual]

removes all pitch objects

Returns

`_errorCode`

4.6.3.17 `virtual _errorCode CSOLOISTPitchSynthesizerIf::removePitchObject (int iPitchObjectIndex) [pure virtual]`

removes a pitch object

Parameters

<i>iPitchObjectIndex</i>	index of the object to be removed
--------------------------	-----------------------------------

Returns

`_errorCode`

4.6.3.18 `virtual _errorCode CSOLOISTPitchSynthesizerIf::setGlobalParam (_eGlobalParam eParam, float fValue) [pure virtual]`

set a global parameter for the whole audio file.

Parameters

<i>eParam</i>	the parameter to be changed (refer to CSOLOISTPitchSynthesizerIf::_eGlobalParam)
<i>fValue</i>	the new value of the parameter

Returns

4.6.3.19 `virtual _errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectBounds (int iPitchObjectIndex, double & dNewStartPos, double & dNewEndPos, bool bUpdateSurroundingObjects = true) [pure virtual]`

change the onset and offset of a given pitch object. This affects the stretch factor of the pitch object. Onset and offset of the given pitch object must not exceed the corresponding onset and offset of the surrounding objects.

Parameters

<i>iPitchObjectIndex</i>	the index of the pitch object
<i>dNewStartPos</i>	new onset position in seconds of the pitch object
<i>dNewEndPos</i>	new offset position in seconds of the pitch object
<i>bUpdateSurroundingObjects</i>	if true the surrounding pitch objects are adapted in terms of note onset and offset. Onset and offset of the given pitch object must not exceed the corresponding onset and offset of the surrounding objects. The stretch factor of the surrounding object may be affected.

Returns`_errorCode`

4.6.3.20 `virtual _errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectData (int iPitchObjectIndex, stSOLOISTPitchObjectData & ObjData, bool bUpdateSurroundingObjects = true)` [pure virtual]

set a complete property structure of a given pitch object

Parameters

<i>iPitch-ObjectIndex</i>	the index of the pitch object
<i>ObjData</i>	the structure for the data
<i>bUpdate-Surrounding-Objects</i>	if true the surrounding pitch objects are adapted in terms of note onset and offset. Onset and offset of the given pitch object must not exceed the corresponding onset and offset of the surrounding objects. The stretch factor of the surrounding object may be affected.

Returns`_errorCode`

4.6.3.21 `virtual _errorCode CSOLOISTPitchSynthesizerIf::setPitchObjectParam (int iPitchObjectIndex, _ePitchObjectParam eParam, float fValue)` [pure virtual]

set a parameter for a give pitch object directly.

Parameters

<i>iPitch-ObjectIndex</i>	the index of the pitch object
<i>eParam</i>	the parameter to be changed (refer to CSOLOISTPitchSynthesizerIf::_ePitchObjectParam)
<i>fValue</i>	the new value of the parameter

Returns

4.6.3.22 `virtual void CSOLOISTPitchSynthesizerIf::setTimePositionInSec (double newReadPosition)` [pure virtual]

set next play position in seconds

Parameters

<i>newRead-Position</i>	new play position in seconds
-------------------------	------------------------------

4.6.3.23 `virtual _errorCode CSOLOISTPitchSynthesizerIf::splitPitchObject (int iPitchObjectIndex, double dRelativeCutPositionFromStartOfPitchObject)`
 [pure virtual]

splits a pitch object into two

Parameters

<i>iPitchObjectIndex</i>	index of the pitch object to be split
<i>dRelativeCutPositionFromStartOfPitchObject</i>	cut position in seconds relative to the start of the pitch object

Returns

`_errorCode`

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

- [SOLOISTPitchSynthesizerAPI.h](#)

5 File Documentation

5.1 DoxyfileTune.txt File Reference

5.2 PSOLAAPI.h File Reference

`#include "SampleInfo.h"` Include dependency graph for PSOLAAPI.h: This graph shows which files directly or indirectly include this file:

5.2.1 Detailed Description

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

5.3 SOLOISTPitchAnalysisAPI.h File Reference

`#include "PSOLAAPI.h"` Include dependency graph for SOLOISTPitchAnalysisAPI.h:

Classes

- struct [_stPitchSegmentationResult_](#)
Structure containing the result of the segmentation.
- struct [_stPitch_](#)
Structure containing the pitch information for the single pitches.
- class [CSOLOISTPitchAnalysisIf](#)
This class does the PSOLA analysis and pitch segmentation.

Typedefs

- typedef struct [_stPitchSegmentationResult_ stPitchSegmentationResult](#)
Structure containing the result of the segmentation.
- typedef struct [_stPitch_ stPitch](#)
Structure containing the pitch information for the single pitches.

5.3.1 Typedef Documentation

5.3.1.1 typedef struct [_stPitch_ stPitch](#)

Structure containing the pitch information for the single pitches.

Pitches are spaced in time according to the pitch. The indexes are equivalent to the indexes of the pitchmarks.

5.3.1.2 typedef struct [_stPitchSegmentationResult_ stPitchSegmentationResult](#)

Structure containing the result of the segmentation.

It contains the estimated pitch and velocity, as well as the beginning and end of the segment in terms of sample position and in terms of referenced pitchmark indexes.

5.4 SOLOISTPitchSynthesizerAPI.h File Reference

Classes

- struct [_stSOLOISTPitchObjectData_](#)
structure that keeps the properties of each pitch object aka note object
- class [CAudioReader](#)
virtual base class for audio file handling, needs to be inherited in order to give the synthesis access to the audio data
- class [CSOLOISTPitchSynthesizerIf](#)
interface for the Pitch synthesizer

Typedefs

- typedef struct `_stSOLOISTPitchObjectData_` `stSOLOISTPitchObjectData`
structure that keeps the properties of each pitch object aka note object

Functions

- `std::vector < CSOLOISTPitchSynthesizerIf::PitchClass > generateScaleBasedOnScaleIdx` (int `iScaleIdx`)

5.4.1 Typedef Documentation

5.4.1.1 typedef struct `_stSOLOISTPitchObjectData_` `stSOLOISTPitchObjectData`

structure that keeps the properties of each pitch object aka note object

5.4.2 Function Documentation

5.4.2.1 `std::vector<CSOLOISTPitchSynthesizerIf::PitchClass>` `generateScaleBasedOnScaleIdx` (int *`iScaleIdx`*)