4TUNE 1.0.2

by zplane.development

February 9, 2018

Contents

1 4Tune Documentation

1.1 Introduction

The 4Tune SDK is designed specifically for karaoke apps especially on mobile devices (iOS). It features a low latency pitch tracking using the same technology as used in the elastiqueSOLOIST combined with an easy to use interface.

The concept of the SDK is to provide the developer of a karaoke (or similar) application easy integration of the signal processing part of the application without much knowledge of the signal processing underneath.

In order to operate the 4Tune SDK only the karoake backing track and some knowledge about the melody to be sung is needed. The developer also needs to know how to operate the audio device I/O. Optimally the 4Tune SDK is operated within the audio callback in order to keep input and output in sync. The following figure shows the basic operation of the SDK.

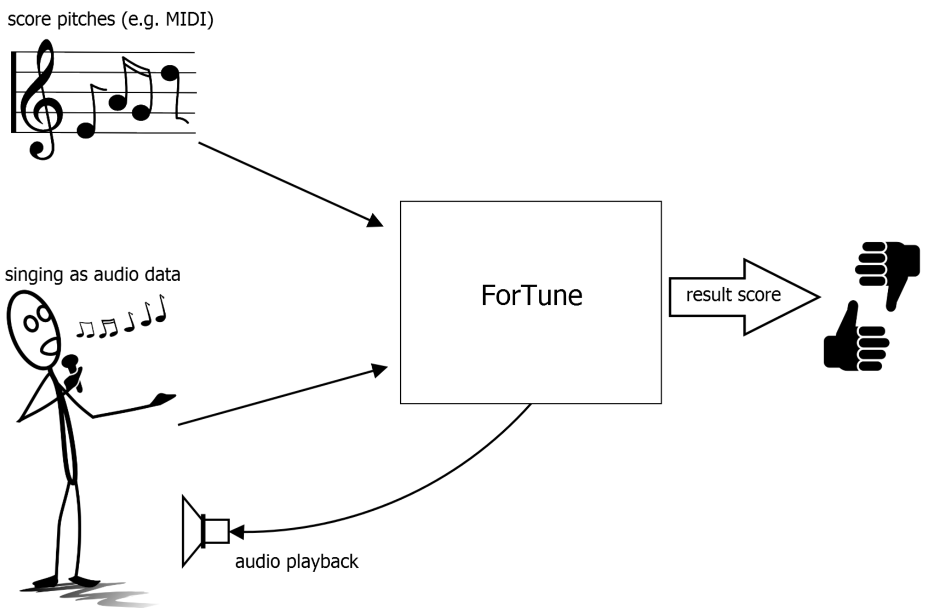

Figure 1: 4Tune operation scheme

While streaming the output (i.d. the karaoke playback) to the speaker one needs to capture the singers input and pass it to the 4Tune API. For each new note sung or each silent part one needs to inform the API about the note or the fact that nothing is sung at that time. In order to avoid octave errors either on the pitch tracking or the karaoke singer side all pitches are mapped to one octave. During and at the end of each note one can query the SDK for the current average deviation of the sung pitch from the desired pitch. This way the application can give feedback about the current performance of the singer for each note. Furthermore there are some service functions for setting the detection tolerances and retrieving the length of the actually sung note with respect to the desired length.

The 4Tune SDK is available for Win32/64, Mac OSX, Linux32/64 and iOS.

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

The structure of this document is as following: First the API of the élastique library is described. The API documentation contains naming conventions and 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 API Documentation

The interface of the SDK is based on the push principle: succeeding blocks of input audio frames are pushed into the process function. Internal memory cannot be accessed from outside, while external memory (e.g. the audio buffer) will not be altered during API function calls.

The SDK is capable of running multiple instances at the same time, but the API is not threadsafe.

1.2.1 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.

1.2.2 Instance Control Functions

The following functions have to be called when using the 4Tune library:

- **C4TuneIf::CreateInstance(.)**

  Creates a new instance of the 4Tune SDK. Besides the reference pointer, the samplerate and the number of channels, optionally a callback class pointer derived from C4TuneCallbackBase may be passed.

- **C4TuneIf::DestroyInstance(.)**

  Destroys an existing instance of the 4Tune SDK.

1.2.3 Parameter Setting Functions

- **C4TuneIf::SetParamValue(.)**

  Use this method to set internal parameters of the SDK. Please refer to C4TuneIf::Param_t for parameters available.

- **C4TuneIf::SetNewNoteOn(.)**

  This method has to be called each time a new note event starts. This is also true for no note events - in this case call the method with the C4TuneIf::kNoNote set.

1.2.4 Process Function

- **C4TuneIf::Process ()**

  This function does the actual processing of the current input audio stream. Please note that this function is not threadsafe - so no other interface methods may be called druing the processing.

1.2.5 Parameter and Result Retrieving Functions

- **C4TuneIf::GetParamValue(.)**

  Use this method to get the value of the internal parameters of the SDK. Please refer to C4TuneIf::Param_t for parameters available.

- **C4TuneIf::GetCurrentAvgPitchOffsetFromLastNoteOn()**

  This returns the the current average offset of the sung note from the desired note set by C4TuneIf::SetNewNoteOn(.). By default there is a tolerance of 50 cent around the pitch, so everything within that range is mapped to the perfect pitch. This may be changed by using C4TuneIf::SetParamValue(.).

- **C4TuneIf::GetPercentageOfSungNoteFromLastNoteOn()**

  This returns the the current percentage of the time of detected pitches versus the time from the last note on set by C4TuneIf::SetNewNoteOn(.).

1.2.5.1 Utilitary Functions

- **C4TuneIf::ResetInstance()**

  Resets the instance to its initial state. May be called instead of destroying the current and creating a new instance.

1.2.6 Usage Example

The complete code can be found in the example source file 4TuneClMain.cpp. For audio file IO, an internal (C++) library is used.

In this example we assume to have an audio file only containing one pitch (in this case that is a C).

In the first step, a pointer to the 4Tune instance has to be declared:

```
C4TuneIf::CreateInstance (pInstanceHandle, pCInputFile->GetSampleRate(),
  pCInputFile->GetNumOfChannels());
```

The number of channels and the samplerate must be known here. Additonally you have the possibility to pass in a pointer to a callback class derived from C4TuneCallback-Base. By overriding the C4TuneCallbackBase::newPitchResultCallback() method the callback will be called each time a new pitch is detected.

After instance creation the expected note has to be set

```
pInstanceHandle->SetNewNoteOn(C4TuneIf::kC);
```

In a real-world-application this should be synchronized to the playback audio stream and a midi file (or similar) containing the expected melody. In that case the prcossing within the following while loop

```
while(bReadNextFrame)
```

will be done in the audio I/O callback.

Now, we do the processing of the inut audio steam:

```
    pInstanceHandle->Process (apfFloatData, iNumFramesRead);
```

The audio data is passed in as a double pointer of which the first pointer represents the channels while the second contains the actual audio samples for each channel: p-Ptr[channels][samples].

After the processing we retrieve the results for the desired note

```
cout << endl<< endl << "Result avg offset:" << pInstanceHandle->
  GetCurrentAvgPitchOffsetFromLastNoteOn() << endl << endl;
```

for the pitch deviation, and

```
cout << endl<< endl << "Result percentage of sung note:" << pInstanceHandle
  ->GetPercentageOfSungNoteFromLastNoteOn() << endl << endl;
```

for the percentage of time that the note was actually sung by the singer.

And finally, when finished, the instance can be destroyed:

```
C4TuneIf::DestroyInstance (pInstanceHandle);
```

The above code snippets demonstrated the basic functionality of the 4Tune library. - Most of the additional functions can be used similar to the given code examples. The exact functionality of the functions is described above.

1.2.7 Error Codes

where are the errors defined etc.. Below is a short description of possible error codes:

- **C4TuneIf::kNoError**
  no error occurred
- **C4TuneIf::kMemError**
  some internal memory allocation failed
- **C4TuneIf::kInvalidFunctionParamError**
  one or more function parameters are not valid
- **C4TuneIf::kNotInitializedError**
  instance has not been initialized yet
- **C4TuneIf::kUnknownError**
  unknown error occurred

1.3 SDK Content

1.3.1 Folder 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 MS VisualC++-Solution (.sln) and all single Projectfiles (.vcproj) 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

If include files are project-intern, they are in the source directory **./src** of the project itself. If include files are to be included by other projects, they can be found in **./src/incl**. The main interface header of the SDK can be found in **./inc**.

1.3.1.5 Resource Files

The resource files, if present, 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 (as well as the distributable Dynamic Link Libraries if contained in the project) can be found in the directory **./bin/release**. In debug-builds, the binary files are in the subfolder **./bin/Debug**.

1.3.2 Project Structure

The project structure is as following:

- **lib4Tune**: The actual 4Tune-library. The project output is a Static Library (Lib).
- **4TuneCl**: Application using the SDK. Consists of the following files:
  - 4TuneClMain.cpp: example code showing how to integrate the SDK.

  The project output is an executable binary (EXE).
- **libzplAudioFile**: internal library for audio IO. The project output is a Static Library (Lib).

1.4 Third Party Libraries

Add information about the libraries under different licenses, add information about these licenses, add information about patent licenses (fees?).

1.5 Support

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

`zplane.development`

grunewaldstr.83

d-10823 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: `info@zplane.de`

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Class Documentation

4.1 C4TuneCallbackBase Class Reference

```
#include <4TuneAPI.h>
```

Public Member Functions

- C4TuneCallbackBase ()
- virtual ~C4TuneCallbackBase ()
- virtual void newPitchResultCallback (float fNewPitchInSemitones, double dTimeStampFromStartOfProcessingInSec, double dTimeStampFromStartofCurrentNoteInSec)=0

  method that has to be overloaded in a derived class which implements the desired behavior

4.1.1 Detailed Description

pure virtual callback class for getting each current pitch result. This class has to be derived and passed to the API. This is optional.

Definition at line 49 of file 4TuneAPI.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 C4TuneCallbackBase::C4TuneCallbackBase ( )

4.1.2.2 virtual C4TuneCallbackBase::~C4TuneCallbackBase ( ) [virtual]

4.1.3 Member Function Documentation

4.1.3.1 virtual void C4TuneCallbackBase::newPitchResultCallback (float *fNewPitchInSemitones*, double *dTimeStampFromStartOfProcessingInSec*, double *dTimeStampFromStartofCurrentNoteInSec*) `[pure virtual]`

method that has to be overloaded in a derived class which implements the desired behavior

Parameters

fNewPitchInSemitones	the current detected pitch
dTimeStampFromStartOfProcessingInSec	the current timestamp in seconds calculated from the beginning of processing or after a reset
dTimeStampFromStartofCurrentNoteInSec	the current timestamp in seconds calculated from the beginning of the current note onset

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

- 4TuneAPI.h

4.2 C4TuneIf Class Reference

```
#include <4TuneAPI.h>
```

Public Types

- enum Notes_t { kNoNote = -1, kC, kCsharp, kD, kDsharp, kE, kF, kFsharp, kG, kGsharp, kA, kAsharp, kB, kNumNotes }

  < all possible notes of the scale. Dflat is equal to Csharp
- enum Param_t { kTuningFreqInHz = 0, kToleranceInSemitones, kIgnoreTimeAfterNoteOnInSec, kNumParams }

  < set of user adjustable parameters
- enum Error_t { kNoError, kMemError, kInvalidFunctionParamError, kNotInitializedError, kUnknownError, kNumErrors }
- enum Version_t { kMajor, kMinor, kPatch, kBuild, kNumVersionInts }

Public Member Functions

- virtual Error_t ResetInstance ()=0

resets the instance, should be called when a new song is to be processed

- virtual Error_t GetParamValue (Param_t iParamIdx, float &fValue)=0

  returns the current value of a specific parameter (see CSOLOIST4TuneIf::Param_t)
- virtual Error_t SetParamValue (Param_t iParamIdx, float fValue)=0

  sets the value for a specific parameter (see CSOLOIST4TuneIf::Param_t)
- virtual Error_t SetNewNoteOn (Notes_t iNoteIdx)=0

  set a new note on for each note and pause (kNoNote) - this re-triggers averaging of the current offset result needs to be called before the next Process call after a note change
- virtual float GetCurrentAvgPitchOffsetFromLastNoteOn ()=0

  returns the current averaged difference between the audio input and the desired note (SetNewNoteOn) - averaging starts with each SetNewNoteOn()
- virtual float GetPercentageOfSungNoteFromLastNoteOn ()=0

  returns the percentage of the actually sung note (the IgnoreTimeAfterNoteOnInSec is also ignored). This means if the singer sings only half of the note this will return 0.5 == 50%
- virtual Error_t Process (float ∗∗ppfInputBufferSeparate, int iNumberOfFrames)=0

  does the actual audio processing aka pitch tracking

Static Public Member Functions

- static const int GetVersion (const Version_t eVersionIdx)

  returns current version of the SDK
- static const char ∗ GetBuildDate ()

  returns the build date as a char string
- static Error_t CreateInstance (C4TuneIf ∗&pCSOLOIST4Tune, float fSampleRate, int iNumberOfChannels, int iMaxInputBufferSize=8192, C4TuneCallbackBase ∗pCallbackClassPtr=NULL)

  creates an instance of the CSOLOIST4TuneIf SDK
- static Error_t DestroyInstance (C4TuneIf ∗&pCSOLOIST4Tune)

  destroys an instance of CSOLOIST4TuneIf SDK

4.2.1 Detailed Description

Definition at line 68 of file 4TuneAPI.h.

4.2.2 Member Enumeration Documentation

4.2.2.1 enum C4TuneIf::Error_t

Enumerator:

kNoError	no error occurred
kMemError	memory allocation failed

kInvalidFunctionParamError one or more function parameters are not valid

kNotInitializedError instance has not been initialized yet

kUnknownError unknown error occurred

kNumErrors

Definition at line 102 of file 4TuneAPI.h.

4.2.2.2 enum C4TuneIf::Notes_t

< all possible notes of the scale. Dflat is equal to Csharp

Enumerator:

kNoNote

kC

kCsharp

kD

kDsharp

kE

kF

kFsharp

kG

kGsharp

kA

kAsharp

kB

kNumNotes

Definition at line 72 of file 4TuneAPI.h.

4.2.2.3 enum C4TuneIf::Param_t

< set of user adjustable parameters

Enumerator:

kTuningFreqInHz optionally set the tuning frequency in Hz (default = 440 Hz)

kToleranceInSemitones optionally set the averaging +/- tolerance in semitones (default = 0.5 semitones = 50 cent), all pitches detected within this range around the target pitch will be mapped to the target pitch

kIgnoreTimeAfterNoteOnInSec optionally set the ignore time after a note onset in seconds (default = 0.010 sec) < all pitches detected within that time span after a new note-on are ignored

kNumParams

Definition at line 91 of file 4TuneAPI.h.

4.2.2.4 enum C4TuneIf::Version_t

Enumerator:

- ***kMajor***
- ***kMinor***
- ***kPatch***
- ***kBuild***
- ***kNumVersionInts***

Definition at line 113 of file 4TuneAPI.h.

4.2.3 Member Function Documentation

4.2.3.1 static Error_t C4TuneIf::CreateInstance (C4TuneIf *& *pCSOLOIST4Tune*, float *fSampleRate*, int *iNumberOfChannels*, int *iMaxInputBufferSize* = `8192`, C4TuneCallbackBase * *pCallbackClassPtr* = `NULL`) `[static]`

creates an instance of the CSOLOIST4TuneIf SDK

Parameters

pCSOLOIST4Tune	the handle to the newly created instance
fSampleRate	the desired samplerate
iNumberOfChannels	the number of audio channels to be processed
iMaxInputBufferSize	defines the maximum input buffersize used in a process call
pCallbackClassPtr	optional instance pointer to derived class instance of CSOLOIST4TuneCallbackBase

Returns

an error code of type CSOLOIST4TuneIf::Error_t

4.2.3.2 static Error_t C4TuneIf::DestroyInstance (C4TuneIf *& *pCSOLOIST4Tune*) `[static]`

destroys an instance of CSOLOIST4TuneIf SDK

Parameters

pCSOLOIST4Tune	a pointer to a previously created instance of CSOLOIST4TuneIf

Returns

an error code of type CSOLOIST4TuneIf::Error_t

4.2.3.3 static const char∗ C4TuneIf::GetBuildDate ( ) [static]

returns the build date as a char string

Returns

build date as a char string

4.2.3.4 virtual float C4TuneIf::GetCurrentAvgPitchOffsetFromLastNoteOn ( ) [pure virtual]

returns the current averaged difference between the audio input and the desired note (SetNewNoteOn) - averaging starts with each SetNewNoteOn()

Returns

current averaged difference

4.2.3.5 virtual Error_t C4TuneIf::GetParamValue (Param_t *iParamIdx*, float & *fValue*) [pure virtual]

returns the current value of a specific parameter (see CSOLOIST4TuneIf::Param_t)

Parameters

iParamIdx	index of the desired parameter
fValue	returns the current value

Returns

4.2.3.6 virtual float C4TuneIf::GetPercentageOfSungNoteFromLastNoteOn ( ) [pure virtual]

returns the percentage of the actually sung note (the IgnoreTimeAfterNoteOnInSec is also ignored). This means if the singer sings only half of the note this will return 0.5 == 50%

Returns

the percentage

4.2.3.7 static const int C4TuneIf::GetVersion (const Version_t *eVersionIdx*) [static]

returns current version of the SDK

Parameters

eVersionIdx	depending on the CSOLOIST4TuneIf::Version_t indes this defines which part of the version number is returned

Returns

version number

4.2.3.8 virtual Error_t C4TuneIf::Process (float ** *ppfInputBufferSeparate,* int *iNumberOfFrames*) `[pure virtual]`

does the actual audio processing aka pitch tracking

Parameters

ppfInput-Buffer-Separate	a double pointer to the input audio data usually coming from the audio callback. It has the form [channels][sampleframes]
iNumberOf-Frames	the number of sample frames passed in

Returns

an error code of type CSOLOIST4TuneIf::Error_t

4.2.3.9 virtual Error_t C4TuneIf::ResetInstance ( ) `[pure virtual]`

resets the instance, should be called when a new song is to be processed

Returns

an error code of type CSOLOIST4TuneIf::Error_t

4.2.3.10 virtual Error_t C4TuneIf::SetNewNoteOn (Notes_t *iNoteIdx*) `[pure virtual]`

set a new note on for each note and pause (kNoNote) - this re-triggers averaging of the current offset result needs to be called before the next Process call after a note change

Parameters

iNoteIdx	the index of the desired note

Returns

4.2.3.11 virtual Error_t C4TuneIf::SetParamValue (Param_t *iParamIdx*, float *fValue*) [pure virtual]

sets the value for a specific parameter (see CSOLOIST4TuneIf::Param_t)

Parameters

iParamIdx	the index of the desired parameter
fValue	the desired value for the parameter

Returns

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

- 4TuneAPI.h

5 File Documentation

5.1 4TuneAPI.h File Reference

Classes

- class C4TuneCallbackBase
- class C4TuneIf

5.2 docugen.txt File Reference

5.2.1 Detailed Description

source documentation main file

Definition in file docugen.txt.