BARBEATQ 1.0.5

by zplane.development

February 9, 2018

Contents

1 barbeatQ Documentation

1.1 Introduction

barbeatQ is zplane's automatic cue point SDK. It analyzes the content of a music signal and outputs a sequence of time stamps identifying the most prominent changes in the audio. barbeatQ furthermore compares the regions between the cue points and groups the most similar regions.

barbeatQ can be used in two different modes: Without and with extra beat information. Without beat information there will be much more cue points and corresponding regions with varying timing precision. With beat information the results are on a higher level and always beat and bar synchronized and thus more usedful for musical and Dj application. Zplane's aufTAKT beat tracking SDK seamlessly integrates with barbeatQ, but the algorithm can also be provided with beat information from a different source such as a fixed beat grid.

barbeatQ is an offline process which means it requires the complete audio file as input and outputs the cue point sequence result only after the file has been processed in its entirety.

The project contains the required libraries for the operating system barbeatQ was licensed for with the appropriate header files. An example application illustrates the functionality of this SDK.

This document is structured as follows: The first part contains the API documentation of the barbeatQ SDK. The API documentation contains naming conventions and function descriptions of the C++-API. In the second part, a detailed explanation of the example application is provided.

1.2 API Documentation

The analysis consists of two stages: a pre-processing stage in which the audio is analyzed, and a processing stage that carries out the actual cue point estimation.

The pre-processing stage is based on the push principle: successive blocks of input audio frames are pushed into the BarbeatQIf::PreProcess() function. It finishes by calling BarbeatQIf::FinishPreProcess() after the audio file has been entirely pushed into BarbeatQIf::PreProcess(). The BarbeatQIf::Process() function is called subsequently and results can be obtained by calling BarbeatQIf::GetResult(). In case a beat tracking algorithm is used, this may be run in parallel to the pre processing stage.

1.2.1 Memory Allocation

The barbeatQ SDK does not allocate any buffers handled by the calling application. The input buffer as well as the result objects have to be allocated/created by the calling application.

1.2.2 Naming Conventions

A **frame** denotes the number of audio samples per channel, i.e. 512 stereo frames correspond to 1024 float values (samples).

A **cue point** denotes the point in time in an audio file of significant change. A **region** is the part between two cue points . A **part index** denotes the group that a region belongs to, i.e. all regions with the same part index are similar.

1.2.3 Instance Control Functions

- **static ErrorType BarbeatQIf::CreateInstance (BarbeatQIf*& pCBarbeatQIf, float fSampleRate, int iNumOfChannels)**

  Creates a new instance of barbeatQ. The handle to the new instance is returned in parameter pCBarbeatQIf. fSampleRate and iNumChannels denote the input samplerate and number of channels, respectively.

  The function returns an error code (see BarbeatQIf::ErrorType). A call to this function is mandatory.

- **static ErrorType DestroyInstance (BarbeatQIf*& pCBarbeatQIf)**

  Destroys the barbeatQ instance provided by pCBarbeatQIf. The function returns an error code (see BarbeatQIf::ErrorType). A call to this function is mandatory.

1.2.4 Processing Functions

- **ErrorType BarbeatQIf::PreProcess (float** ppfInputBuffer, int iNumberOfFrames)**

  Pre-processing function. ppfInputBuffer is an array of pointers to the audio data. ppfInputBuffer[0] is a pointer to the data of the first channel, ppfInputBuffer[1] points to the data of the second channel etc. iNumberOfFrames specifies the number of frames, i.e. the number of samples in each channel. This function can repeatedly be called with successive chunks of audio until the entire signal has been pushed into barbeatQ. This function will return an error if it is called after BarbeatQIf::FinishPreProcess() has been called.

- **ErrorType BarbeatQIf::FinishPreProcess()**

  This function has to be called after all audio frames have been pushed into barbeatQ by BarbeatQIf::PreProcess(). It can (and needs to) be called only once and will return an error if called before BarbeatQIf::PreProcess() or after BarbeatQIf::Process().

- **ErrorType BarbeatQIf::Process(const CaufTAKTResultIf* const pBeatResult = nullptr)**

  This function does the actual cue point calculation. It can be called without input arguments, in which case the algorithm will figure out the cue points by itself without any synchronization to bar boundaries - it will therefore also return more cue points then when being synchronized. If a CaufTAKTResultIf beat info object is provided, barbeatQ will use this information to synchronize the

cue points with the provided bar times. This function has to be called after -BarbeatQIf::FinishPreProcess().

1.2.5 Result Retrieving Function

- **ErrorType GetResult (BarbeatQResultIf *pCResult)**

  Returns a list of cue points as a BarbeatQResultIf object. The BarbeatQResult-If object needs to be instantiated first before it is passed to this function. This function can only be called after BarbeatQIf::Process() has been called.

- **int GetDownbeatIndex()**

  Returns the downbeat estimate done by barbeatQ. In addition to the downbeat estimate passed in from the external beat tracking barbeatQ does its own estimate. This is accessible by this method. This function can only be called after BarbeatQIf::Process() has been called.

- **float GetJumpTime (float fSourceTime)**

  Returns the most similar point in time for a given source time. When beat synchronized the result will be quantized to beats. This function can only be called after BarbeatQIf::Process() has been called.

- **float GetJumpTimeForTargetTime(float fSourceTime, float fTargetTime)**

  Returns the most similar point in time for a given source time within the same region as the target time. When beat synchronized the result will be quantized to beats. This function can only be called after BarbeatQIf::Process() has been called.

1.3 Command Line Usage Example

The command line example can be executed by the following command

```
barbeatQCl <inputFile> <chordResultFile>
```

The complete code can be found in the example source file barbeatQClMain.cpp.
In the first step, we declare a BarbeatQIf pointer and a barbeatQResult pointer

```
BarbeatQIf*            pInstanceHandle  = 0;
BarbeatQResultIf*      pCResult         = 0;
```

and create an instance of the BarbeatQIf class:

```
eError = BarbeatQIf::CreateInstance (pInstanceHandle,
                                     pCInputFile->GetSampleRate(),
                                     pCInputFile->GetNumOfChannels());
```

We then read chunks of data from our input file,

```
while (bReadNextFrame)
{
    iNumFramesRead = pCInputFile->Read (ppfFloatData, kBlockSize);

    if(iNumFramesRead < kBlockSize)
    {
        for (int ch = 0; ch < pCInputFile->GetNumOfChannels(); ch++)
            memset(&ppfFloatData[ch][iNumFramesRead], 0, (kBlockSize-
  iNumFramesRead)*sizeof(float));
        bReadNextFrame = false;
    }
```

And push each chunk into our PreProcess() function.

```
    // preprocessing
    eError = pInstanceHandle->PreProcess (ppfFloatData, iNumFramesRead);
```

After the entire file has been read and pushed into barbeatQ, we call FinishPreProcess() once to terminate the preprocessing stage

```
eError = pInstanceHandle->FinishPreProcess();
```

We then call barbeatQ's process function:

```
eError = pInstanceHandle->Process ();
```

If beat information is available we could provide barbeatQ with a CaufTAKTResultIf*

```
//eError = pInstanceHandle->Process (pCBeatResult); // beat results can
   optionally be provided
```

To obtain the resulting chord sequence, we create an instance of BarbeatQResultIf,

```
// create results instance
if (BarbeatQResultIf::CreateInstance (pCResult) == false)
```

and call barbeatQ's GetResult() function

```
// get cue point results
pInstanceHandle->GetResult (pCResult);
```

Finally we destroy the BarbeatQIf instance and eventually the BarbeatQResultIf instance

```
    BarbeatQIf::DestroyInstance (pInstanceHandle);
    BarbeatQResultIf::DestroyInstance(pCResult);
```

The above code snippets demonstrated the basic functionality of the barbeatQ library.

1.4 Support

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

zplane.development GmbH & Co KG

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 BarbeatQIf Class Reference

```
#include <barbeatQIf.h>
```

Public Types

- enum ErrorType { noError, memError, invalidFunctionParamError, notInitializedError, notPreProcessedError, notProcessedError, invalidDataChunk, unknownError, numErrors }
- enum VersionType { major, minor, patch, revision, numVersionInts }

Public Member Functions

- virtual ErrorType PreProcess (float **ppfInputBuffer, int iNumberOfFrames)=0

  Perform the preprocessing.
- virtual ErrorType FinishPreProcess ()=0

  Terminate the preprocessing stage.
- virtual ErrorType Process (CaufTAKTResultIf *pAuftaktResult=0, bool bUseAuftaktDownbeat=false)=0

  Perform the cue-point estimation and segmentation.
- virtual int GetDownbeatIndex ()=0

  get barbeatQ downbeat estimate
- virtual float GetJumpTime (float fSourceTime)=0

  Returns the playposition with the highest similarity to the source time passed in.
- virtual float GetJumpTimeForTargetTime (float fSourceTime, float fTargetTime)=0

  Returns the playposition with the highest similarity to the source time passed in the same cue point area as the target time.
- virtual ErrorType GetResult (BarbeatQResultIf *pCResult)=0

  Returns the playposition with the highest similarity to the source time passed in.
- virtual void GetDataChunk (void *pPreAllocatedDataChunk)=0

  Get internal data after preprocessing.
- virtual size_t GetDataChunkSizeInBytes ()=0

  Get length of memory to allocate for GetDataChunk.
- virtual ErrorType SetDataChunk (void *pDataChunk, int iDataChunkSizeInBytes)=0

  Set saved internal data.

Static Public Member Functions

- static const int GetVersion (const VersionType eVersionIdx)

  Get part of the BarbeatQ version number.
- static const char * GetBuildDate ()

  Get date of this BarbeatQ build.
- static ErrorType CreateInstance (BarbeatQIf *&pCBarbeatQIf, float fSampleRate, int iNumOfChannels)

  Create an instance of BarbeatQ.
- static ErrorType DestroyInstance (BarbeatQIf *&pCBarbeatQIf)

  Destroy the BarbeatQ instance.

Protected Member Functions

- virtual ~BarbeatQIf ()

4.1.1 Member Enumeration Documentation

4.1.1.1 enum BarbeatQIf::ErrorType

error codes

Enumerator:

noError no error occurred

memError memory allocation failed

invalidFunctionParamError one or more function parameters are not valid

notInitializedError instance has not been initialized yet

notPreProcessedError instance has not been initialized yet

notProcessedError instance has not been initialized yet

invalidDataChunk data chunk was invalid

unknownError unknown error occurred

numErrors

4.1.1.2 enum BarbeatQIf::VersionType

version number

Enumerator:

major major version number

minor minor version number

patch patch version number

revision revision number

numVersionInts

4.1.2 Constructor & Destructor Documentation

4.1.2.1 virtual BarbeatQIf::~BarbeatQIf ( ) [inline, protected, virtual]

4.1.3 Member Function Documentation

4.1.3.1 static const int BarbeatQIf::GetVersion (const VersionType *eVersionIdx*) [static]

Returns major version, minor version, patch and build number of this BarbeatQ version

Parameters

eVersionIdx	requested version number part

Returns

version number part

4.1.3.2 static const char* BarbeatQIf::GetBuildDate ( ) [static]

Returns the build date string

Returns

build date string

4.1.3.3 static ErrorType BarbeatQIf::CreateInstance (BarbeatQIf *&* *pCBarbeatQIf,* float *fSampleRate,* int *iNumOfChannels*) [static]

Creates an instance of BarbeatQ

Parameters

pCBarbeat-QIf	reference to BarbeatQ instance pointer
fSampleRate	sample rate of input audio
iNumOf-Channels	number of input audio channels

Returns

an error code, or 0 if no error occurred

4.1.3.4 static ErrorType BarbeatQIf::DestroyInstance (BarbeatQIf *&* *pCBarbeatQIf*) [static]

Destroys an instance of BarbeatQ

Parameters

pCBarbeat-QIf	reference to BarbeatQIf instance pointer

Returns

an error code, or 0 if no error occurred

4.1.3.5 virtual ErrorType BarbeatQIf::PreProcess (float ** *ppfInputBuffer,* int *iNumberOfFrames*) [pure virtual]

This function performs the preprocessing. It can be called multiple times in order to provide successive chunks of the input audio signal.

Parameters

ppfInput-Buffer	pointer to the input data chunk. ppfInputBuffer[i] point to the i-th audio channel.
iNumberOf-Frames	The number of audio samples in each audio channel of the provided input data chunk.

Returns

an error code, or 0 if no error occurred

4.1.3.6 virtual ErrorType BarbeatQIf::FinishPreProcess ( ) [pure virtual]

Function to terminate the preprocessing stage. This function should only be called once after the last input frames have been provided by the PreProcess function. A call to this function is required before proceeding to the Process function.

Returns

an error code, or 0 if no error occurred

4.1.3.7 virtual ErrorType BarbeatQIf::Process (CaufTAKTResultIf * *pAuftaktResult* = 0, bool *bUseAuftaktDownbeat* = false) [pure virtual]

Performs the cue-point estimation and segmentation. This function can only be called when the preprocessing stage has finished or a previously analyzed data chunk has been set.

Parameters

pAuftakt-Result	optional already existing aufTAKT result (if not specified not beat synchronization will be used)
bUse-Auftakt-Downbeat	if set to true barbeatQ will use the downbeat estimate of auftTAKT, otherwise (default) it will generate its own

Returns

an error code, or 0 if no error occurred

4.1.3.8 virtual int BarbeatQIf::GetDownbeatIndex ( ) [pure virtual]

Returns the result of an internal downbeat estimation based on the beat marks passed in

Returns

an downbeat idx

4.1.3.9 virtual float BarbeatQIf::GetJumpTime (float *fSourceTime*) [pure virtual]

Parameters

fSourceTime	

Returns

the playposition with the highest similarity

4.1.3.10 virtual float BarbeatQIf::GetJumpTimeForTargetTime (float *fSourceTime*, float *fTargetTime*) [pure virtual]

Parameters

fSourceTime	
fTargetTime	

Returns

the playposition with the highest similarity

4.1.3.11 virtual ErrorType BarbeatQIf::GetResult (BarbeatQResultIf * *pCResult*) [pure virtual]

Parameters

fSourceTime	

Returns

the playposition with the highest similarity

4.1.3.12 virtual void BarbeatQIf::GetDataChunk (void * *pPreAllocatedDataChunk*) [pure virtual]

Returns internal analysis data after preprocessing in an pre-allocated memory chunk. This can be used to save analysis data for later processing. For allocating memory please refer to GetDataChunkSizeInBytes().

Parameters

pPreAllocatedDataChunk	: pointer to pre-allocated memory chunk

Returns

void : returns

4.1.3.13 virtual size_t BarbeatQIf::GetDataChunkSizeInBytes ( ) [pure virtual]

Returns the length in byte to be pre allocated in order to properly call GetDataChunk.

Returns

size_t : returns length in bytes.

4.1.3.14 virtual ErrorType BarbeatQIf::SetDataChunk (void * *pDataChunk*, int *iDataChunkSizeInBytes*) [pure virtual]

Set a saved data chunk in order to recover a previous pre-analysis state.

Parameters

pDataChunk	: pointer to data chunk
iDataChunkSizeInBytes	: size of the data chunk

Returns

ErrorType : returns noError if data recovery was performed without error, otherwise invalidDataChunk

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

- barbeatQIf.h

4.2 BarbeatQResultIf Class Reference

`#include <barbeatQResultIf.h>`

Classes

- class CueInfo

Public Member Functions

- virtual int GetNumEntries () const =0
- virtual CueInfo GetCuepoint (int iIdx)=0

Static Public Member Functions

- static bool CreateInstance (BarbeatQResultIf ∗&pCbarbeatQResultIf)
- static bool DestroyInstance (BarbeatQResultIf ∗&pCbarbeatQResultIf)

Protected Member Functions

- virtual ∼BarbeatQResultIf ()

4.2.1 Constructor & Destructor Documentation

4.2.1.1 virtual BarbeatQResultIf::∼BarbeatQResultIf ( ) [inline, protected, virtual]

4.2.2 Member Function Documentation

4.2.2.1 static bool BarbeatQResultIf::CreateInstance (BarbeatQResultIf ∗& *pCbarbeatQResultIf*) [static]

4.2.2.2 static bool BarbeatQResultIf::DestroyInstance (BarbeatQResultIf ∗& *pCbarbeatQResultIf*) [static]

4.2.2.3 virtual int BarbeatQResultIf::GetNumEntries ( ) const [pure virtual]

4.2.2.4 virtual CueInfo BarbeatQResultIf::GetCuepoint (int *iIdx*) [pure virtual]

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

- barbeatQResultIf.h

4.3 BarbeatQResultIf::CueInfo Class Reference

`#include <barbeatQResultIf.h>`

Public Attributes

- float fPos
- int iPartIdx

4.3.1 Member Data Documentation

4.3.1.1 float BarbeatQResultIf::CueInfo::fPos

4.3.1.2 int BarbeatQResultIf::CueInfo::iPartIdx

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

- barbeatQResultIf.h

5 File Documentation

5.1 barbeatQIf.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class BarbeatQIf

5.2 barbeatQResultIf.h File Reference

`#include "barbeatQIf.h"` Include dependency graph for barbeatQResultIf.h:

Classes

- class BarbeatQResultIf
- class BarbeatQResultIf::CueInfo

5.3 docugen.txt File Reference

5.3.1 Detailed Description

source documentation main file