Kort 2.1.2

by zplane.development

August 24, 2021

Contents

1 Kort Documentation

1.1 Introduction

Kort is zplane's chord recognition SDK. It analyzes the harmonic content of a music signal and outputs a sequence of chords with corresponding time stamps. Kort assumes that the input signal contains music in Western tonality with 12 pitches per octave (C, C#, D,..., B) and in equal temperament. It furthermore assumes that the music contains simultaneously sounding notes that form musical chords consisting of three to four different pitches.
Each chord in the output chord sequence is specified by a root pitch (e.g. C, C#, D, ...) and a chord type (e.g. major, minor, diminished, 7, ...). The output also provides information about the pitch classes contained in a chord (e.g. "C, E and G" in a C major chord). For each chord in the resulting chord sequence, the user can retrieve a list of alternative chords together with corresponding probabilities.
Kort can be used in three different modes: A basic mode that outputs only major and minor chords, and an extended mode that outputs a larger set of chord types containing between three and four pitches, and a custom mode in which users can define their own set of chords.
Chords can be aligned with beat times. Zplane's [aufTAKT] beat tracking SDK seamlessly integrates with Kort, but the algorithm can also be provided with beat information from a different source such as a fixed beat grid. Likewise, the tuning frequency of the input signal can be specified manually if this information is available.
Kort is an offline process which means it requires the complete audio file as input and outputs the chord sequence result only after the file has been processed in its entirety. It is *not* an online or real-time chord recognition system. In extended mode the algorithm is able to process audio files approx. 60 times faster than realtime on a 64-bit 2.3 GHz Dual-Core processor. The mimimum sample rate is 16 kHz. There is no restriction on the number of input channels.

The project contains the required libraries for the operating system Kort 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 Kort 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.

1.1.1 Naming Conventions

The following naming conventions are used throughout this manual: A **frame** denotes the number of audio samples per channel, i.e. 512 stereo frames correspond to 1024 float values (samples).
A **pitch class** describes the name of a pitch independent of the octave it occurred in. In other words, notes with pitches at *C0, C1, C2, etc.* all belong to pitch class *C*.
The **root pitch** refers to the pitch class of the root of the chord, i.e. the root pitch of a

C major chord is pitch class *C*.
A **chord type** is defined as a set of intervals above a root. The chord type *major*, for example, consists of the intervals: *unison, major third, and fifth*.
Chords consists of a *root pitch* and a chord type, e.g. *C major*.
A **chord dictionary** or simply **dictionary** is a set of chord types. The dictionary Kort↩
::majorMinor for example consists of the chord types *major* and *minor*.

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 chord estimation.
The pre-processing stage is based on the push principle: successive blocks of input audio frames are pushed into the Kort::preProcess() function. It finishes by calling Kort::finishPreProcess() after the audio file has been entirely pushed into Kort::pre↩
Process(). The Kort::process() function is called subsequently and results can be obtained by calling Kort::getResult().

1.2.1 Memory Allocation

The Kort 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 Instance Control Functions

- **ErrorType Kort::initialize (float sampleRate, int fileSizeInFrames, int num↩
OfChannels)**
  Initializes an instance of Kort. sampleRate and numChannels denote the input samplerate and number of channels, respectively. The expected length of the input file in frames is provided by the argument iFileSizeInFrames. This value is used to initialize internal buffers required for the chord estimation. It is possible to push more frames into Kort::preProcess() than initially specified, but this will cause further memory allocations during Kort::preProcess().
  The function returns an error code (see Kort::ErrorType). A call to this function is mandatory.

1.2.3 Parameter Setting Functions

- **ErrorType Kort::setChordDictionaryType (ChordDictionaryType chord↩
Dict)**
  Sets the chord dictionary Kort will use (see Kort::ChordDictionaryType). This function is optional but it needs to be called before Kort::process(). The default dictionary is Kort::ChordDictionaryType::extended.

- **ErrorType setCustomChordDictionary (const std::vector<ChordDictionary↩
Entry>& dictionary)**

Sets a custom chord dictionary. Multiple entries containing the same chord are not allowed.

- **ErrorType Kort::setTuningFrequency (float tuningFrequency)**
  Sets the tuning frequency of the input audio. When called before Kort::pre↩ Process(), this will bypass the internal tuning frequency estimation which will lead to overall shorter processing times.

- **ErrorType Kort::setBassWeight (float bassWeight)**
  Sets the weight of the estimated bass pitches against the estimate of the remaining chord notes. The value must be in the range between 0 and 1. A value of 0 means that the chord estimation is done without bass estimation, a value of 1 means that only the bass (and no other pitches) is taken into account for the chord estimation. The default value is 0.35.

1.2.4 Processing Functions

- **ErrorType Kort::preProcess (float const ∗const ∗const inputBuffer, int number↩ OfFrames)**
  Pre-processing function. ppfInputBuffer is an array of pointers to the audio data. inputBuffer[0] is a pointer to the data of the first channel, ppfInputBuffer[1] points to the data of the second channel etc. numberOfFrames 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 Kort. This function will return an error if it is called after Kort::finishPreProcess() has been called.

- **ErrorType Kort::finishPreProcess()**
  This function has to be called after all audio frames have been pushed into Kort by Kort::preProcess(). It can (and needs to) be called only once and will return an error if called before Kort::preProcess() or after Kort::process().

- **ErrorType Kort::process()**
  This function does the actual chord estimation. It can be called without any input arguments, in which case the algorithm will figure out the chord boundaries by itself. If a vector with a beat grid is provided, Kort will use this information to match the chord boundaries with the provided beat times. This function has to be called after Kort::finishPreProcess().

1.2.5 Result Retrieving Function

- **ErrorType Kort::getResult (std::vector<ChordSequenceElement>& chord↩ Result)**
  Returns the chord sequence as a vector of chord sequence elements. This function can only be called after Kort::process() has been called.

- **ErrorType Kort::getTuningFrequency (float& tuningFrequency)**
  Returns the estimated tuning frequency. This function can only be called after the preprocessing stage.

1.3 Command Line Usage Example

The command line example can be executed by the following command

```
KortCl -i <inputFile> -r <chordResultFile>
```

The complete code can be found in the example source file KortClMain.cpp.
In the first step, we create an instance of the Kort class and initialize:

```
    Kort kortInstance;

    eError = kortInstance.initialize (inputFile.GetSampleRate(),
                                      inputFile.GetFileSize(),
                                      inputFile.GetNumOfChannels());
```

We can select a chord dictionary as follows:

```
    kortInstance.setChordDictionaryType (Kort::extended);
```

We can set the tuning frequency of the input file in case this information is available. It is advisable to do this immediately after instance creation.

```
    // set tuning frequency (optional)
    //kortInstance.setTuningFrequency (tuningFreq);
```

We then read chunks of data from our input file,

```
//        while (readNextFrame)
//        {
//            inputBuffer.clear();
//            numFramesRead = inputFile.Read(inputBuffer, blockSize);
//
//            if (numFramesRead < blockSize)
//            {
//                readNextFrame = false;
//            }
```

And push each chunk into our preProcess() function.

```
    // now we can start the preprocessing!
    cout << "****************************************" << endl;
    cout << "Kort Pre-Processing!" << endl;

    while (readNextFrame)
    {
        inputBuffer.clear();
        // read audio data
        int numFramesRead (inputFile.Read (inputBuffer, blockSize));

        if(numFramesRead < blockSize)
        {
            readNextFrame = false;
        }

        if (verbose)
            CLShowProcessTime(currentFrame++, inputFile.GetSampleRate(), blockSize);


        // preprocessing
        clStartTime = clock();
        eError = kortInstance.preProcess (inputBuffer, numFramesRead);
        }
```

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

```
eError = kortInstance.finishPreProcess();
```

The bass weight can be set by calling setBassWeight()

```
//     set bass weight (optional)
//    kortInstance.setBassWeight (0.73f);
```

We then call Kort's process function:

```
eError = kortInstance.process ();
```

If beat information is available we could provide Kort with with a vector of type std↩ ::vector<KortGlobals::Beat>

```
//eError = kortInstance.process (beatResult); // beat results can optionally be provided
```

To obtain the resulting chord sequence, we call Kort's getResult() function and decide whether to combine adjacent chords that have the same basic triad (bChord↩ Smoothing).

```
// get chord results
bool chordSmoothing = false;
eError = kortInstance.getResult (chordResult, chordSmoothing);
```

We can access the individual chords of the resulting chord sequence (e.g. in order to print them on the command line) as well as a list of alternative chords:

```
cout << "Printing Results!" << endl;

float tuningFreq;
kortInstance.getTuningFrequency (tuningFreq);
//kortInstanceChunked.getTuningFrequency (tuningFreq);
cout << "Tuning frequency: " << tuningFreq << endl;

for (int i = 0; i < chordResult.size(); i++)
{
    const Kort::ChordSequenceElement element = chordResult[i];
    cout << element.chordElement.chord.GetName() << "  ";

    // get alternative chords for the current entry
    std::vector<Kort::ChordElement> alternativeChords = chordResult[i].alternativeChords;
}
cout << endl;
```

The above code snippets demonstrated the basic functionality of the Kort 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 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 KortGlobals Namespace Reference

Global defines and class interfaces.

Classes

- struct Beat
- class Chord

  Class representing a musical chord.
- class Key

  Class representing a musical key.

Enumerations

- enum PitchClass {
  noPitch = -1, C, CSharp, D,
  DSharp, E, F, FSharp,
  G, GSharp, A, ASharp,
  B }
- enum ChordType {
  noChord, major, minor, sus2,
  sus4, diminished, augmented, majorSeventh,
  minorSeventh, seventh, diminishedSeventh, halfDiminishedSeventh,
  majorSixth, minorSixth, numChordTypes }
- enum KeyType { majorKey, minorKey }

Variables

- static const int numPitchClasses = 12

5.1.1 Detailed Description

Global defines and class interfaces.

5.1.2 Enumeration Type Documentation

ChordType `enum KortGlobals::ChordType`

Chord types

Enumerator

noChord	
major	
minor	
sus2	
sus4	
diminished	
augmented	
majorSeventh	
minorSeventh	
seventh	
diminishedSeventh	
halfDiminishedSeventh	
majorSixth	
minorSixth	
numChordTypes	

Definition at line 68 of file KortGlobals.h.

```
69     {
70         noChord,
71         major,
72         minor,
73         sus2,
74         sus4,
75         diminished,
76         augmented,
77         majorSeventh,
78         minorSeventh,
79         seventh,
80         diminishedSeventh,
81         halfDiminishedSeventh,
82         majorSixth,
83         minorSixth,
84 
85         numChordTypes
86     };
```

KeyType `enum KortGlobals::KeyType`

Key types

Enumerator

majorKey	
minorKey	

Definition at line 89 of file KortGlobals.h.

```
90     {
91         majorKey,
92         minorKey
93     };
```

PitchClass `enum KortGlobals::PitchClass`

Pitch classes

Enumerator

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

Definition at line 48 of file KortGlobals.h.

```
49     {
50         noPitch = -1,
51         C,
52         CSharp,
53         D,
54         DSharp,
55         E,
56         F,
57         FSharp,
58         G,
59         GSharp,
60         A,
61         ASharp,
62         B
63     };
```

5.1.3 Variable Documentation

numPitchClasses `const int KortGlobals::numPitchClasses = 12 [static]`

Definition at line 65 of file KortGlobals.h.

5.2 zplane Namespace Reference

Interface class for the Kort SDK.

Classes

- class Kort

5.2.1 Detailed Description

Interface class for the Kort SDK.

6 Class Documentation

6.1 KortGlobals::Beat Struct Reference

`#include <Kort/KortGlobals.h>`

Public Attributes

- float timeInS

  time marker corresponding to this beat in seconds
- unsigned int beatCountInBar

  beat number in the bar, e.g. 1 for a downbeat etc.
- unsigned int numBeatsInBar

  number of beats in the enclosing bar i.e. the meter e.g. 4 for 4/4 but also 7 for 7/8 etc.

6.1.1 Detailed Description

Struct representing instances of beats in a result
Definition at line 96 of file KortGlobals.h.

6.1.2 Member Data Documentation

beatCountInBar `unsigned int KortGlobals::Beat::beatCountInBar`
beat number in the bar, e.g. 1 for a downbeat etc.
Definition at line 98 of file KortGlobals.h.

numBeatsInBar `unsigned int KortGlobals::Beat::numBeatsInBar`
number of beats in the enclosing bar i.e. the meter e.g. 4 for 4/4 but also 7 for 7/8 etc.
Definition at line 99 of file KortGlobals.h.

timeInS `float KortGlobals::Beat::timeInS`

time marker corresponding to this beat in seconds

Definition at line 97 of file KortGlobals.h.

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

- Kort/KortGlobals.h

6.2 KortGlobals::Chord Class Reference

Class representing a musical chord.

`#include <Kort/KortGlobals.h>`

Public Member Functions

- Chord ()
- Chord (PitchClass rootPitch, ChordType chordType)
- Chord (const Chord &other)
- Chord (Impl *pImpl)
- ~Chord ()
- Chord & operator= (const Chord &rhs)
- const bool operator== (const Chord &rhs) const
- const bool operator!= (const Chord &rhs) const
- PitchClass GetRootPitch () const
- ChordType GetChordType () const
- std::vector< PitchClass > GetPitches () const
- std::string GetName () const

6.2.1 Detailed Description

Class representing a musical chord.

Definition at line 103 of file KortGlobals.h.

6.2.2 Constructor & Destructor Documentation

Chord() [1/4] `KortGlobals::Chord::Chord ( )`

Constructs an empty chord object.

Chord() [2/4]
```
KortGlobals::Chord::Chord (
        PitchClass rootPitch,
        ChordType chordType )
```

Constructs a chord with a given rootPitch and chordType.

Chord() [3/4]
```
KortGlobals::Chord::Chord (
        const Chord & other )
```

Constructs a copy of the provided chord.

Chord() [4/4] `KortGlobals::Chord::Chord ( Impl * pImpl ) [explicit]`

Constructs a chord given an existing implementation. This constructor is only for internal use.

~Chord() `KortGlobals::Chord::~Chord ( )`

Destroys a chord object.

6.2.3 Member Function Documentation

GetChordType() `ChordType KortGlobals::Chord::GetChordType ( ) const`

Returns the chord type of the chord.

GetName() `std::string KortGlobals::Chord::GetName ( ) const`

Returns the name string of the chord in the format <rootPitch>:<chordType>.

GetPitches() `std::vector<PitchClass> KortGlobals::Chord::GetPitches ( ) const`

Returns the pitch classes of the chord.

GetRootPitch() `PitchClass KortGlobals::Chord::GetRootPitch ( ) const`

Returns the pitch class of the root of the chord.

operator"!=() `const bool KortGlobals::Chord::operator!= ( const Chord & rhs ) const`

Inequality operator.

operator=() `Chord& KortGlobals::Chord::operator= ( const Chord & rhs )`

Assignment operator.

operator==() `const bool KortGlobals::Chord::operator== ( const Chord & rhs ) const`

Equality operator.

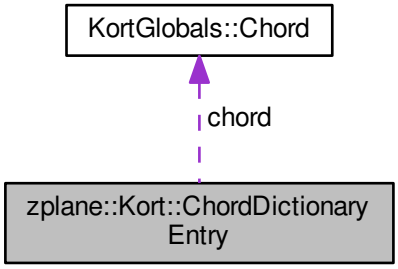
The documentation for this class was generated from the following file:

- Kort/KortGlobals.h

6.3 zplane::Kort::ChordDictionaryEntry Struct Reference

`#include <Kort/Kort.h>`

Collaboration diagram for zplane::Kort::ChordDictionaryEntry:



Public Attributes

- KortGlobals::Chord chord
- float likelihood = -1

6.3.1 Detailed Description

Struct representing instances of chords in a chord dictionary
Definition at line 120 of file Kort.h.

6.3.2 Member Data Documentation

chord `KortGlobals::Chord zplane::Kort::ChordDictionaryEntry::chord`
Definition at line 122 of file Kort.h.

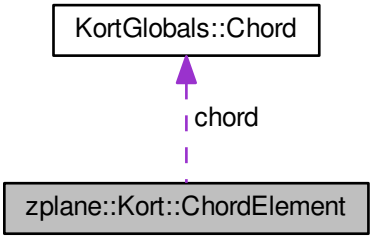
likelihood `float zplane::Kort::ChordDictionaryEntry::likelihood = -1`
Definition at line 123 of file Kort.h.
The documentation for this struct was generated from the following file:

- Kort/Kort.h

6.4 zplane::Kort::ChordElement Struct Reference

`#include <Kort/Kort.h>`

Collaboration diagram for zplane::Kort::ChordElement:



Public Attributes

- KortGlobals::Chord chord
- float probability = -1

  the probability of this sequence element

6.4.1 Detailed Description

Struct representing instances of chords in a result
Definition at line 105 of file Kort.h.

6.4.2 Member Data Documentation

chord `KortGlobals::Chord zplane::Kort::ChordElement::chord`
Definition at line 107 of file Kort.h.

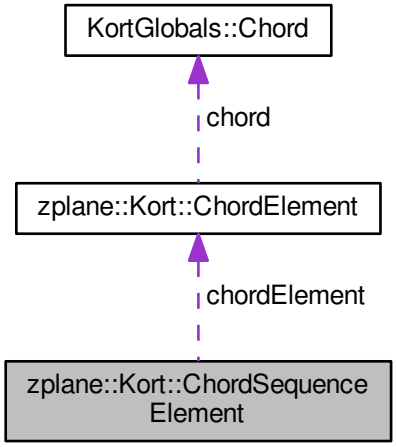
probability `float zplane::Kort::ChordElement::probability = -1`
the probability of this sequence element
Definition at line 108 of file Kort.h.
The documentation for this struct was generated from the following file:

- Kort/Kort.h

6.5 zplane::Kort::ChordSequenceElement Struct Reference

`#include <Kort/Kort.h>`

Collaboration diagram for zplane::Kort::ChordSequenceElement:



Public Attributes

- float startTimeInS
- ChordElement chordElement
- std::vector< ChordElement > alternativeChords

6.5.1 Detailed Description

Struct representing instances of chord elements in a result
Definition at line 112 of file Kort.h.

6.5.2 Member Data Documentation

alternativeChords `std::vector<ChordElement> zplane::Kort::ChordSequence↩ Element::alternativeChords`
Definition at line 116 of file Kort.h.

chordElement `ChordElement zplane::Kort::ChordSequenceElement::chordElement`
Definition at line 115 of file Kort.h.

startTimeInS `float zplane::Kort::ChordSequenceElement::startTimeInS`

Definition at line 114 of file Kort.h.

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

- Kort/Kort.h

6.6 KortGlobals::Key Class Reference

Class representing a musical key.

`#include <Kort/KortGlobals.h>`

Public Member Functions

- Key ()
- Key (PitchClass rootPitch, KeyType keyType)
- Key (const Key &other)
- Key (Impl *pImpl)
- ~Key ()
- Key & operator= (const Key &rhs)
- const bool operator== (const Key &rhs) const
- const bool operator!= (const Key &rhs) const
- PitchClass GetRootPitch () const
- KeyType GetKeyType () const
- std::string GetName () const

6.6.1 Detailed Description

Class representing a musical key.

Definition at line 154 of file KortGlobals.h.

6.6.2 Constructor & Destructor Documentation

Key() [1/4] `KortGlobals::Key::Key ( )`

Constructs an empty key object.

Key() [2/4]
```
KortGlobals::Key::Key (
        PitchClass rootPitch,
        KeyType keyType )
```

Constructs a key with a given rootPitch and keyType.

Key() [3/4]
```
KortGlobals::Key::Key (
        const Key & other )
```

Constructs a copy of the provided key.

Key() [4/4]
```
KortGlobals::Key::Key (
        Impl * pImpl )
```

Constructs a key given an existing implementation. This constructor is only for internal use.

~Key() `KortGlobals::Key::~Key ( )`
Destroys a key object.

6.6.3 Member Function Documentation

GetKeyType() `KeyType KortGlobals::Key::GetKeyType ( ) const`
Returns the key type of the key.

GetName() `std::string KortGlobals::Key::GetName ( ) const`
Returns the name string of the key in the format <rootPitch>:<keyType>.

GetRootPitch() `PitchClass KortGlobals::Key::GetRootPitch ( ) const`
Returns the pitch class of the root of the key.

operator"!=() `const bool KortGlobals::Key::operator!= ( const Key & rhs ) const`
Inequality operator.

operator=() `Key& KortGlobals::Key::operator= ( const Key & rhs )`
Assignment operator.

operator==() `const bool KortGlobals::Key::operator== ( const Key & rhs ) const`
Equality operator.
The documentation for this class was generated from the following file:

- Kort/KortGlobals.h

6.7 zplane::Kort Class Reference

`#include <Kort/Kort.h>`

Classes

- struct ChordDictionaryEntry
- struct ChordElement
- struct ChordSequenceElement

Public Types

- enum ErrorType {
  noError, memError, invalidFunctionParamError, invalidFunctionCallError,
  invalidValueError, invalidChordError, notInitializedError, notPreProcessedError,
  notProcessedError, invalidDataChunk, unknownError, numErrors }
- enum ChordDictionaryType {
  majorMinor, triads, triadsAndSevenths, extended,
  custom }

- enum DataChunkCheckpoint { afterPreProcessing, afterTuningCorrection }

Public Member Functions

- Kort ()
- ~Kort ()
- ErrorType initialize (float sampleRate, int fileSizeInFrames, int numOfChannels)
- bool isInitialized ()
- ErrorType setTuningFrequency (float tuningFrequency)
- ErrorType setKey (KortGlobals::Key key, float strictness=0.038)
- ErrorType setBassWeight (float bassWeight)
- ErrorType setChordDictionaryType (ChordDictionaryType chordDict)
- ChordDictionaryType getChordDictionaryType ()
- ErrorType setCustomChordDictionary (const std::vector< ChordDictionaryEntry > &dictionary)
- ErrorType preProcess (float const *const *const inputBuffer, int numberOf↩ Frames)
- ErrorType finishPreProcess ()
- ErrorType process (const std::vector< KortGlobals::Beat > &beatResult, bool doubleBeatRate, bool highChordChangeSensitivity=false)
- ErrorType process (bool highChordChangeSensitivity=false)
- ErrorType process (const CaufTAKTResultIf *const beatResult, bool high↩ ChordChangeSensitivity=false)
- ErrorType getResult (std::vector< ChordSequenceElement > &chordResult, bool chordSmoothing=false)
- ErrorType getTuningFrequency (float &tuningFrequency)
- ErrorType getDataChunk (void *preAllocatedDataChunk, DataChunkCheckpoint checkpoint=afterTuningCorrection)
- ErrorType getDataChunkSizeInBytes (size_t &dataChunkSizeInBytes, Data↩ ChunkCheckpoint checkpoint=afterTuningCorrection)
- ErrorType setDataChunk (void *dataChunk, int dataChunkSizeInBytes)

Static Public Member Functions

- static const char * getVersion ()
- static const char * getBuildDate ()

6.7.1 Detailed Description

Definition at line 48 of file Kort.h.

6.7.2 Member Enumeration Documentation

ChordDictionaryType `enum zplane::Kort::ChordDictionaryType`

Chord dictionary

Enumerator

majorMinor	only major and minor chords
triads	dictionary containing all triads including suspended and diminished/augmented chords
triadsAndSevenths	same as triads but with seventh chords
extended	dictionary containing triads and tetrads
custom	custom dictionary, return type only when custom dictionary is set

Definition at line 71 of file Kort.h.

```
72         {
73             majorMinor,
74             triads,
75             triadsAndSevenths,
76             extended,
77             custom
78         };
```

DataChunkCheckpoint `enum zplane::Kort::DataChunkCheckpoint`

Chord data chunk checkpoints

Enumerator

afterPreProcessing	save preprocessed audio and tuning frequency
afterTuningCorrection	save computed pre-estimation, not possible to set tuning frequency

Definition at line 81 of file Kort.h.

```
82         {
83             afterPreProcessing,
84             afterTuningCorrection
85         };
```

ErrorType `enum zplane::Kort::ErrorType`

Error codes

Enumerator

noError	no error occurred
memError	memory allocation failed
invalidFunctionParamError	one or more function parameters are not valid
invalidFunctionCallError	function call not allowed at this stage
invalidValueError	value is not in expected range
invalidChordError	chord parameters are invalid

Enumerator

notInitializedError	instance has not been initialized yet
notPreProcessedError	instance has not been preprocessed yet
notProcessedError	instance has not been processed yet
invalidDataChunk	data chunk was invalid
unknownError	unknown error occurred
numErrors	

Definition at line 52 of file Kort.h.

```
53         {
54             noError,
55             memError,
56             invalidFunctionParamError,
57             invalidFunctionCallError,
58             invalidValueError,
59             invalidChordError,
60             notInitializedError,
61             notPreProcessedError,
62             notProcessedError,
63             invalidDataChunk,
64             unknownError,
65
66
67             numErrors
68         };
```

6.7.3 Constructor & Destructor Documentation

Kort() `zplane::Kort::Kort ( )`

~Kort() `zplane::Kort::~Kort ( )`

6.7.4 Member Function Documentation

finishPreProcess() `ErrorType zplane::Kort::finishPreProcess ( )`

Terminates 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.

getBuildDate() `static const char* zplane::Kort::getBuildDate ( ) [static]`

Returns the build date string.

Returns

date string

getChordDictionaryType() `ChordDictionaryType zplane::Kort::getChordDictionaryType ( )`

Returns the current chord dictionary type.

getDataChunk() `ErrorType zplane::Kort::getDataChunk (`
`void * preAllocatedDataChunk,`
`DataChunkCheckpoint checkpoint = afterTuningCorrection )`

Returns internal analysis data after preprocessing in an pre-allocated memory chunk.

This can be used to save analysis data for later processing. The memory handling has to be done by the calling application. The size of pre-allocated memory can be retrieved by getDataChunkSizeInBytes()

getDataChunkSizeInBytes() `ErrorType zplane::Kort::getDataChunkSizeInBytes (`
`size_t & dataChunkSizeInBytes,`
`DataChunkCheckpoint checkpoint = afterTuningCorrection )`

Returns the length in bytes to be pre allocated in order to properly call getDataChunk.

getResult() `ErrorType zplane::Kort::getResult (`
`std::vector< ChordSequenceElement > & chordResult,`
`bool chordSmoothing = false )`

Returns resulting chord sequence.

Parameters

chordSmoothing	whether to combine adjacent chords that have the same basic triad (optional).

getTuningFrequency() `ErrorType zplane::Kort::getTuningFrequency (`
`float & tuningFrequency )`

Returns the tuning frequency.

getVersion() `static const char* zplane::Kort::getVersion ( ) [static]`

Returns major version, minor version, patch and build number of this Kort version.

Returns

version number

initialize() `ErrorType zplane::Kort::initialize (`
`float sampleRate,`
`int fileSizeInFrames,`
`int numOfChannels )`

Initialize an instance of Kort.

Must be called before using any of Kort's functionality. If called on an already initialized instance of Kort, the instance gets re-initialized.

Parameters

sampleRate	sample rate of input audio
fileSizeInFrames	length of input audio in frames
numOfChannels	number of input audio channels

Returns

an error code, or 0 if no error occurred

isInitialized() `bool zplane::Kort::isInitialized ( )`

preProcess()
```
ErrorType zplane::Kort::preProcess (
    float const *const *const inputBuffer,
    int numberOfFrames )
```

Preprocesses a block of audio.

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

Parameters

inputBuffer	pointer to the input data chunk. inputBuffer[i] points to the i-th audio channel.
numberOfFrames	The number of audio samples in each audio channel of the provided input data chunk.

process() [1/3]
```
ErrorType zplane::Kort::process (
    const std::vector< KortGlobals::Beat > & beatResult,
    bool doubleBeatRate,
    bool highChordChangeSensitivity = false )
```

Performs the chord estimation.

Performs the chord estimation. This function can only be called when the preprocessing stage has finished or a previously analyzed data chunk has been set.

Parameters

beatResult	array with beat information.
doubleBeatRate	interpolates the provided beat grid by a factor of 2 (e.g. from 4th to 8th notes).
highChordChangeSensitivity	increases sensitivity for chord changes (optional).

process() [2/3] `ErrorType zplane::Kort::process (`
`bool highChordChangeSensitivity = false )`

process() [3/3] `ErrorType zplane::Kort::process (`
`const CaufTAKTResultIf *const beatResult,`
`bool highChordChangeSensitivity = false )`

Performs the chord estimation with beat results (legacy).

This function can only be called when the preprocessing stage has finished or a previously analyzed data chunk has been set.

Parameters

beatResult	pointer to beat information. Providing a null pointer means that no beat information is available.
highChordChangeSensitivity	increases sensitivity for chord changes (optional).

setBassWeight() `ErrorType zplane::Kort::setBassWeight (`
`float bassWeight )`

Sets the bass weight.

The bass weight controls the influence of the estimated bass pitches against the estimate of the remaining chord notes. The value must be in the range between 0 and 1. A value of 0 means that the chord estimation is done without bass estimation, a value of 1 means that only the bass (and no other pitches) is taken into account for the chord estimation.

This function is optional. The default value for the bass weight is 0.35.

setChordDictionaryType() `ErrorType zplane::Kort::setChordDictionaryType (`
`ChordDictionaryType chordDict )`

Set the chord dictionary type.

The chord dictionary determines the types of chords that Kort can detect. The default value is Kort::ChordDictionaryType::extended. Kort::ChordDictionaryType↩ ::custom cannot be set; a custom dictionary has to be set by Kort::SetCustomChord↩ Dictionary. The chord dictionary type needs to be set before calling Kort::Process().

setCustomChordDictionary() `ErrorType zplane::Kort::setCustomChordDictionary (`
`const std::vector< ChordDictionaryEntry > & dictionary )`

Sets a custom chord dictionary.

The dictionary needs to be set before calling Kort::process(). Chord dictionary does not allow multiple entries of the same chord.

setDataChunk() `ErrorType zplane::Kort::setDataChunk (`
`void * dataChunk,`
`int dataChunkSizeInBytes )`

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

Parameters

dataChunk	pointer to data chunk
dataChunkSizeInBytes	size of the data chunk

Returns

noError if data recovery was performed without error, otherwise invalidData↩ Chunk

setKey() `ErrorType zplane::Kort::setKey (`
`KortGlobals::Key key,`
`float strictness = 0.038 )`

Sets the key.

Whether to prefer chords that are in detected key. This function is optional. The value "strictness" must be in range between 0 and 1.

Parameters

key	detected key of song
strictness	determines how much preference is given to chords that belong to the same key.

setTuningFrequency() `ErrorType zplane::Kort::setTuningFrequency (`
`float tuningFrequency )`

Sets the tuning frequency.

This function is optional. When called before the preprocessing stage it bypasses Kort's internal tuning frequency estimation which can speed up the computation.

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

- Kort/Kort.h

7 File Documentation

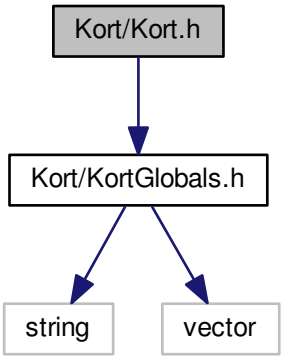
7.1 /work/project/docs/docugen.txt File Reference

7.2 Kort/Kort.h File Reference

interface of the Kort class.

```
#include "Kort/KortGlobals.h"
```

Include dependency graph for Kort.h:



Classes

- class zplane::Kort
- struct zplane::Kort::ChordElement
- struct zplane::Kort::ChordSequenceElement
- struct zplane::Kort::ChordDictionaryEntry

Namespaces

- zplane

  Interface class for the Kort SDK.

7.2.1 Detailed Description

interface of the Kort class.

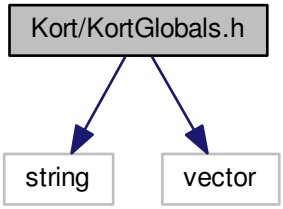
:

7.3 Kort/KortGlobals.h File Reference

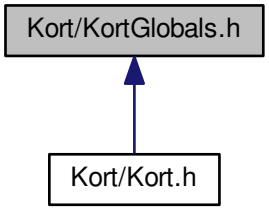
interface of the KortGlobals namespace.

```
#include <string>
#include <vector>
```

Include dependency graph for KortGlobals.h:



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



Classes

- struct KortGlobals::Beat
- class KortGlobals::Chord

  Class representing a musical chord.
- class KortGlobals::Key

  Class representing a musical key.

Namespaces

- KortGlobals

  Global defines and class interfaces.

Enumerations

- enum KortGlobals::PitchClass { KortGlobals::noPitch = -1, KortGlobals::C, KortGlobals::CSharp, KortGlobals::D, KortGlobals::DSharp, KortGlobals::E, KortGlobals::F, KortGlobals::FSharp, KortGlobals::G, KortGlobals::GSharp, KortGlobals::A, KortGlobals::ASharp, KortGlobals::B }
- enum KortGlobals::ChordType { KortGlobals::noChord, KortGlobals::major, KortGlobals::minor, KortGlobals::sus2, KortGlobals::sus4, KortGlobals::diminished, KortGlobals::augmented, KortGlobals::majorSeventh, KortGlobals::minorSeventh, KortGlobals::seventh, KortGlobals::diminishedSeventh, KortGlobals::halfDiminishedSeventh, KortGlobals::majorSixth, KortGlobals::minorSixth, KortGlobals::numChordTypes }
- enum KortGlobals::KeyType { KortGlobals::majorKey, KortGlobals::minorKey }

Variables

- static const int KortGlobals::numPitchClasses = 12

7.3.1 Detailed Description

interface of the KortGlobals namespace.

:

Index