



auftakt 4.2.0

by zplane.development

(c) 2023 zplane.development GmbH & Co. KG

March 1, 2023

Contents

| | |
|---|-----------|
| 1 AUFTAKT Documentation | 2 |
| 1.1 Introduction | 2 |
| 1.2 What's new in AUFTAKT v4 | 3 |
| 1.3 API Documentation | 3 |
| 1.3.1 Memory Allocation | 3 |
| 1.3.2 Naming Conventions | 3 |
| 1.3.3 Required Functions | 3 |
| 1.3.4 C++ Usage example | 4 |
| 1.4 Command Line Usage Example | 5 |
| 1.5 Support | 5 |
| 2 Namespace Index | 6 |
| 2.1 Namespace List | 6 |
| 3 Class Index | 6 |
| 3.1 Class List | 6 |
| 4 File Index | 6 |
| 4.1 File List | 6 |
| 5 Namespace Documentation | 6 |
| 5.1 zplane Namespace Reference | 6 |
| 6 Class Documentation | 7 |
| 6.1 zplane::aufTAKT Class Reference | 7 |
| 6.1.1 Detailed Description | 8 |
| 6.1.2 Member Enumeration Documentation | 8 |
| 6.1.3 Constructor & Destructor Documentation | 9 |
| 6.1.4 Member Function Documentation | 9 |
| 6.2 zplane::aufTAKT::Beat Struct Reference | 14 |
| 6.2.1 Detailed Description | 14 |
| 6.2.2 Member Data Documentation | 14 |
| 6.3 zplane::aufTAKT::KnownDownbeat Struct Reference | 15 |
| 6.3.1 Detailed Description | 15 |
| 6.3.2 Constructor & Destructor Documentation | 16 |
| 6.3.3 Member Data Documentation | 16 |
| 7 File Documentation | 17 |
| 7.1 /work/project/docs/docugen.txt File Reference | 17 |
| 7.2 aufTAKT/aufTAKT.h File Reference | 17 |
| 7.2.1 Detailed Description | 17 |
| Index | 18 |

1 AUFTAKT Documentation

1.1 Introduction

AUFTAKT is a beat tracking algorithm that is able to perform beat estimation even in drumless material and short audio snippets such as loops. The fourth version of the AUFTAKT SDK is able to not only follow tempo changes but also to cope with meter changes and rhythmically complex material. Beside the new capabilities, AUFTAKT v4 provides an improvement in downbeat tracking of roughly 25%, when compared with the previous version of AUFTAKT.

Based on a novel approach, AUFTAKT utilizes a deep neural network to perform beat tracking. The neural network estimates a probability for each point in the musical piece to be either a beat or a downbeat. From these beat probabilities, AUFTAKT then predicts the most likely sequence of beats in the musical piece using a statistical model. An overview of the algorithm is shown below.

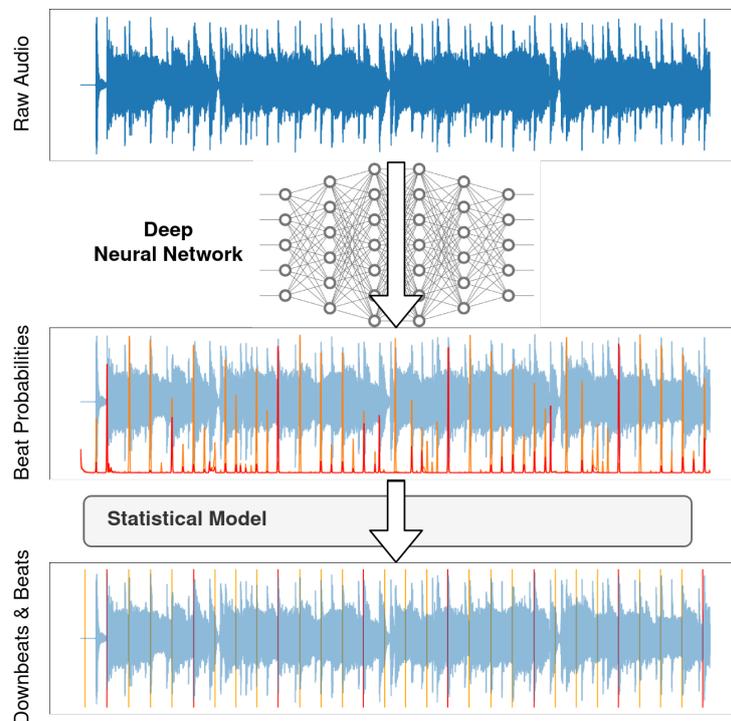


Figure 1: Overview of the beat tracking algorithm

This document contains all information you need to get started and use AUFTAKT to the best of its abilities. The documentation structured as follows: After introducing what is new in this version of AUFTAKT, the API is described. At the core of the API description is a usage example taken from the `aufTAKTCIMain.cpp` file, which is included in this package to get you started as quickly as possible. Afterwards, a brief description of the pre-compiled command line application delivered with the SDK is given. This can be used to quickly and easy get results from AUFTAKT. What follows is

a detailed documentation of the AUFTAKT interface and all of the methods and structs contained within.

1.2 What's new in AUFTAKT v4

Apart from its aforementioned completely new designed algorithm AUFTAKT also introduces a new interface structure, when compared to previous zplane SDKs. The restructuring made the interface more clear and easier to use. Instead of using virtual methods and having to deal with pointers to instances, we now use the pImpl idiom. Meaning you can now directly instantiate the aufTAKT class. After the instantiation it just needs one call to the **aufTAKT::initialize()** method and the instance is ready to be put to use. We were also able to reduce the interface to just one header (leaving out a dedicated interface for results), which makes obtaining beat-tracking results from the AUFTAKT class simpler and more intuitive than in previous versions of the SDK.

1.3 API Documentation

1.3.1 Memory Allocation

The AUFTAKT SDK does not allocate buffers handled by the calling application. The input buffers have to be allocated by the calling application. The input buffer size of the OnsetDetection module shouldnt exceed a value of 16384 frames. Audio buffers are allocated as double arrays of [channels][SamplesPerChannel].

1.3.2 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.

1.3.3 Required Functions

The following methods have to be used (in the order they appear here) to obtain beat tracking results from AUFTAKT. For details see the documentation of the methods and the example below.

- **aufTAKT::initialize (float sampleRate, int numberOfChannels, int max↔ BlockSize, float minTempoInBPM = 60.f, float maxTempoInBPM = 240.f, std::vector<int> meters = {3, 4});**
 - Initalizes an AUFTAKT instance. For parameters s. below. Defaults should work well for most applications
- **aufTAKT::preProcess (float const* const* const inputBuffer, int number↔ OffFrames);**
 - Feed audio data (samples) to AUFTAKT. For parameters s. below
- **aufTAKT::finishPreProcessing(bool flushBuffer = true)**
 - Finish the pre processing stage and prepare AUFTAKT for beat estimation. Set flushBuffer to false if you plan on calling preProcess again after retrieving an intermediate result

- **aufTAKT::process()**
 - Estimate beats and downbeats
- **aufTAKT::getResult (std::vector<aufTAKT::Beat>& beatResult)**
 - Fill a pre-allocated vector of **aufTAKT::Beat** structs with the result

1.3.4 C++ Usage example

The complete code referenced to here can be found in the example source file `aufTAKTCIMain.cpp`. In the first step, an instance of `aufTAKT` has to be created. Note that this is different to how you would create an instance of previous `zplane` SDKs. After instantiation, we use the **`aufTAKT::initialize()`** method to prepare `AUFTAKT` to be used.

```
aufTAKT aufTaktInstance;
aufTaktInstance.initialize (inputFile.GetSampleRate(),
                           inputFile.GetNumOfChannels(),
                           blockSize);
```

In this case, we initialize the instance of `AUFTAKT` simply with its default parameters. Other parameters than the ones shown here are the minimum and maximum tempo `AUFTAKT` operates at, as well as the meters it considers when estimating downbeats. For details on these parameters see the documentation of **`aufTAKT::initialize()`** or **`aufTAKT::setBeatTrackingParams()`**. After initialization we call the **`aufTAKT::preProcess()`** method feeding `AUFTAKT` blocks of audio data. This function could for example be called several times from e.g. a while-loop that runs as long as there are still samples available for `AUFTAKT`.

```
const aufTAKT::Error_t preProcessingError =
aufTaktInstance.preProcess (inputBuffer, numFramesRead);
```

Once all available blocks have been fed to `AUFTAKT` we can now call the **`aufTAKT::finishPreProcessing()`** method to prepare `AUFTAKT` for the beat and downbeat estimation.

```
const aufTAKT::Error_t errorFromFinishPreProcessing =
aufTaktInstance.finishPreProcessing();
```

This method contains roughly half of `AUFTAKT`'s computational complexity. The other half of the computation happens once the process method is called subsequently:

```
const aufTAKT::Error_t errorFromProcess = aufTaktInstance.process();
```

After **`aufTAKT::process()`** has been called the results are now available for you to retrieve. To get the beat results one first has to allocate a vector of **`aufTAKT::Beat`** structs and subsequently pass it on to the **`aufTAKT::getResult()`** method. Similarly one can retrieve the estimated tempo via **`aufTAKT::getOverallTempo()`**.

```
std::vector<aufTAKT::Beat> beats;
aufTaktInstance.getResult (beats);

float tempoInBPM = -1.f;
aufTaktInstance.getOverallTempo (tempoInBPM);
```

Each instance of `aufTAKT::Beat` in the returned vector contains the members `timeInS`, `beatCountInBar` as well as `numBeatsInBar` providing all needed information about the beats i.e. the exact point in time, the location within a bar and the current meter. The tempo is computed simply as a mean over the duration of the audio `AUF` `TAKT` was given. Therefore, if there are a lot of tempo changes present in the piece of music, this estimate does not represent the true tempo.

If you are not satisfied with the results `AUF` `TAKT` delivered initially, you have two possibilities to refine its estimation. The first and easier one is to use the `aufTAKT::setBeatTrackingParams()` method to narrow down or extend the range in which `AUF` `TAKT` performs its estimation. For example, if you have a piece of music at 110 `BPM` which `AUF` `TAKT` wrongly estimates to be 220 `BPM`, it could be sufficient to reduce the upper tempo bound (i.e. the parameter `maxTempoInBPM`) to something like 150 `BPM`. If you have more detailed information about the beat structure for example some input given by a user by tapping, you could use the `aufTAKT::setKnownDownbeats` method to provide this information to `AUF` `TAKT` as follows:

```
// instantiate a container for known downbeats
std::vector<aufTAKT::KnownDownbeat> knownDownbeats;
// append known downbeat to the vector
constexpr float knownDownbeatTimeInS = 5.42f;
constexpr float tempoFromTheKnownDownbeatOnwardsInBPM = 164.f;
knownDownbeats.emplace_back (knownDownbeatTimeInS,
                             tempoFromTheKnownDownbeatOnwardsInBPM);
// call known downbeats setter before calling process
const aufTAKT::Error_t errorFromSettingKnownDownbeats =
    aufTaktInstance.setKnownDownbeats (knownDownbeats);
```

After doing so, you will need to call `aufTAKT::process()` again and retrieve the new result as shown above. If you have this kind of information available anyhow, you could also give it to `AUF` `TAKT` before any estimation by calling the above method before any call to `aufTAKT::process()`.

1.4 Command Line Usage Example

The compiled example is a command line application that reads audio files in the `WAV` format and performs beat tracking on them. The command line application will write all estimated information to `stdout`. The bare minimum usage example would be:

```
aufTAKTCl -i path/to/input_file.wav
```

Optionally, `aufTAKTCl` can also write the results to text files of your choice. By passing file paths behind the `-b` and `-d` flags respectively, `aufTAKTCl` will write line separated beat and downbeat locations given in seconds respectively. When given a file path with the `-r` flag, `aufTAKTCl` will create a file in which every line contains the following information for each beat found:

```
<timeMarkerInS> | <beatCountInBar> / <numBeatsInBar>
```

For further information simply call

```
aufTAKTCl --help
```

1.5 Support

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

[zplane.development](https://github.com/zplane/development)

grunewaldstr. 83
d-10965 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:

[zplane](#) 6

3 Class Index

3.1 Class List

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

[zplane::aufTAKT](#) 7

[zplane::aufTAKT::Beat](#) 14

[zplane::aufTAKT::KnownDownbeat](#) 15

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

[aufTAKT/aufTAKT.h](#)
Interface of the aufTAKT class 17

5 Namespace Documentation

5.1 zplane Namespace Reference

Classes

- class [aufTAKT](#)

6 Class Documentation

6.1 `zplane::aufTAKT` Class Reference

```
#include <aufTAKT/aufTAKT.h>
```

Classes

- struct [Beat](#)
- struct [KnownDownbeat](#)

Public Types

- enum `Error_t` {
 [kNoError](#), [kMemError](#), [kInvalidFunctionParamError](#), [kNotInitializedError](#),
 [kPreProcessNeverCalledError](#), [kCannotBeCalledAtThisStageError](#), [kUnfinishedPreProcessingError](#),
 [kNoResultAvailableError](#),
 [kEmptyResultError](#), [kInvalidDownbeatTimeError](#), [kDownbeatsOutOfOrderError](#),
 [kDownbeatsTooCloseToEachOtherError](#),
 [kInvalidTempoError](#), [kInvalidMeterError](#), [kInvalidDataChunkError](#), [kUnknownError](#),
 [kNumErrors](#) }

Public Member Functions

- [aufTAKT](#) ()
- [~aufTAKT](#) ()
- [Error_t initialize](#) (float `sampleRate`, int `numberOfChannels`, int `maxBlockSize`, float `minTempoInBPM=60.f`, float `maxTempoInBPM=240.f`, `std::vector< int > meters=std::vector< int >()`)
- bool [isInitialized](#) ()
- [Error_t preProcess](#) (float const *const *const `inputBuffer`, int `numberOfFrames`)
- [Error_t finishPreProcessing](#) (bool `flushBuffers=true`)
- [Error_t process](#) ()
- [Error_t getResult](#) (`std::vector< aufTAKT::Beat > &beatResult`)
- [Error_t getOverallTempo](#) (float &`tempoInBPM`)
- [Error_t setBeatTrackingParameters](#) (float `minTempoInBPM`, float `maxTempoInBPM`, `std::vector< int > meters`)
- [Error_t setKnownDownbeats](#) (const `std::vector< KnownDownbeat > &knownDownbeats`)
- [Error_t clearKnownDownbeats](#) ()
- [Error_t getDataChunk](#) (void *`preAllocatedDataChunk`)
- [Error_t getDataChunkSizeInBytes](#) (`std::size_t &dataChunkSizeInBytes`)
- [Error_t setDataChunk](#) (void *`dataChunk`)
- [Error_t reset](#) ()

Static Public Member Functions

- static const char * [getVersion](#) ()
- static const char * [getBuildDate](#) ()

6.1.1 Detailed Description

Definition at line 42 of file aufTAKT.h.

6.1.2 Member Enumeration Documentation

Error_t `enum zplane::aufTAKT::Error_t`

Enumerator

| | |
|---|---|
| <code>kNoError</code> | no error occurred |
| <code>kMemError</code> | memory allocation failed |
| <code>kInvalidFunctionParamError</code> | one or more function parameters are not valid |
| <code>kNotInitializedError</code> | instance has not been initialized yet |
| <code>kPreProcessNeverCalledError</code> | finishPreProcessing() can only be called if <code>preProcess</code> has at least been called once |
| <code>kCanNotBeCalledAtThisStageError</code> | further audio can't be added via <code>preProcess</code> if <code>finishPreProcessing</code> was called with <code>flushBuffers=true</code> or a data chunk was set, <code>finishPreProcessing</code> can't be called if a data chunk was set |
| <code>kUnfinishedPreProcessingError</code> | finishPreProcessing() has to be called before process() or <code>setKnownDownbeat()</code> |
| <code>kNoResultAvailableError</code> | process() method was never called, therefore there's no available result |
| <code>kEmptyResultError</code> | the result is empty, this is most likely caused by too little audio in the buffer |
| <code>kInvalidDownbeatTimeError</code> | given downbeat time is less than 0 or greater or equal than the audio length preprocessed by the algorithm |
| <code>kDownbeatsOutOfOrderError</code> | downbeats need to be given ordered by the time they appear |
| <code>kDownbeatsTooCloseToEachOtherError</code> | downbeats must be spaced at least one bar away from each other with respect to their corresponding tempo and meter |
| <code>kInvalidTempoError</code> | given tempo in BPM outside of aufTAKT 's range, s. docs. |
| <code>kInvalidMeterError</code> | given meter not in aufTAKT 's meter range, s. docs. |
| <code>kInvalidDataChunkError</code> | data chunk was invalid |
| <code>kUnknownError</code> | unknown error occurred |
| <code>kNumErrors</code> | |

Definition at line 45 of file aufTAKT.h.

```
46     {
47         kNoError,
48         kMemError,
49         kInvalidFunctionParamError,
50         kNotInitializedError,
51         kPreProcessNeverCalledError,
52         kCanNotBeCalledAtThisStageError,
53         kUnfinishedPreProcessingError,
54         kNoResultAvailableError,
55         kEmptyResultError,
56         kInvalidDownbeatTimeError,
57         kDownbeatsOutOfOrderError,
58         kDownbeatsTooCloseToEachOtherError,
59         kInvalidTempoError,
60         kInvalidMeterError,
61         kInvalidDataChunkError,
62         kUnknownError,
63
64         kNumErrors
65     };
```

6.1.3 Constructor & Destructor Documentation

aufTAKT() `zplane::aufTAKT::aufTAKT ()`

~aufTAKT() `zplane::aufTAKT::~~aufTAKT ()`

6.1.4 Member Function Documentation

clearKnownDownbeats() `Error_t zplane::aufTAKT::clearKnownDownbeats ()`

Clear all known downbeats if any have been set. Otherwise this method does not have an effect.

Returns

Error_t: Returns an error code.

finishPreProcessing() `Error_t zplane::aufTAKT::finishPreProcessing (bool flushBuffers = true)`

Finish the pre-processing step. Needs to be called once, before `process()` can be called. Should you plan to add further audio to the buffer via `preProcess(...)`, set the `flushBuffers` flag to false.

Parameters

| | |
|---------------------|--|
| <i>flushBuffers</i> | |
|---------------------|--|

Returns

Error_t: Returns an error code

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

getDataChunk() `Error_t zplane::aufTAKT::getDataChunk (void * preAllocatedDataChunk)`

Returns internal analysis data after finished 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\(\)](#). Note that this does not include parameters or any KnownDownbeats previously set.

Parameters

| | |
|------------------------------|---|
| <i>preAllocatedDataChunk</i> | Pointer to the start of the allocated block for the dataChunk |
|------------------------------|---|

Returns

Error_t: Returns errors if called before finishPreProcess() has ever been called - will return kUnfinishedPreProcessingError in that case.

getDataChunkSizeInBytes() `Error_t zplane::aufTAKT::getDataChunkSizeInBytes (std::size_t & dataChunkSizeInBytes)`

Returns the length in bytes to be pre allocated in order to properly call [getDataChunk\(\)](#).

getOverallTempo() `Error_t zplane::aufTAKT::getOverallTempo (float & tempoInBPM)`

Get the overall tempo of all audio in buffer. Note that this is the average over the whole audio previously processed. This estimate therefore might seem off, if multiple tempi are present in the piece of music given to [aufTAKT](#).

Parameters

| | |
|-------------------|--|
| <i>tempoInBPM</i> | Reference to a pre-allocated float in which aufTAKT writes the estimated tempo in beats per minute |
|-------------------|--|

Returns

Error_t: Returns an error code

```
getResult() Error_t zplane::aufTAKT::getResult (
    std::vector< aufTAKT::Beat > & beatResult )
```

Get the result of a beat and downbeat estimation. Can only be called after `process()` has been called.

Parameters

| | |
|-------------------|---|
| <i>beatResult</i> | pre-allocated and empty vector of beats, which will be filled inside this method. |
|-------------------|---|

Returns

Error_t: Returns an error code

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

```
initialize() Error_t zplane::aufTAKT::initialize (
    float sampleRate,
    int numberOfChannels,
    int maxBlockSize,
    float minTempoInBPM = 60.f,
    float maxTempoInBPM = 240.f,
    std::vector< int > meters = std::vector< int > ( ) )
```

Initialize an instance of `aufTAKT`. Must be called before using any of `aufTAKT`'s functionality. If called on an already initialized instance of `aufTAKT`, the instance gets re-initialized. Please refer to `setBeatTrackingParameters(.)` if it's necessary to re-initialize.

Parameters

| | |
|-------------------------|--|
| <i>sampleRate</i> | Sample rate of the input signal in Hertz. |
| <i>numberOfChannels</i> | Number of channels in the input signal. |
| <i>minTempoInBPM</i> | Lower bound for tempo for <code>aufTAKT</code> to consider. Must not be lower than 50 BPM. |
| <i>maxTempoInBPM</i> | Upper bound for tempo for <code>aufTAKT</code> to consider. Must not exceed 250 BPM. |
| <i>meters</i> | List of meters for <code>aufTAKT</code> to consider. E.g. 3 for 3/4, 4 for 4/4 but also 7 for 7/8. Must not contain elements less than 1 or greater than 10. Note that the computational complexity of parts of <code>aufTAKT</code> scale linearly with the number of beats i.e. with the sum of the elements of this vector. If this vector is empty (as here by default), it will be set to {3, 4}. |

Returns

Error_t: Returns an error code.

isInitialized() `bool zplane::aufTAKT::isInitialized ()`

preProcess() `Error_t zplane::aufTAKT::preProcess (`
`float const *const *const inputBuffer,`
`int numberOfFrames)`

Pre-Process a block of audio. This method can be called multiple times in order to provide successive chunks of the input audio signal. This method can also be called after [finishPreProcessing\(\)](#) and [process\(\)](#) having been called. All audio given to [aufTAKT](#) via this method will be held in a buffer until [reset\(\)](#) is called. If you plan on calling this method again after you call [finishPreProcessing\(...\)](#), you must set the `flushBuffer` parameter of aforementioned method to false.

Parameters

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

Returns

Error_t: Returns an error code

process() `Error_t zplane::aufTAKT::process ()`

Perform beat and downbeat estimation. This is where the major chunk of computational load happens. After calling this method, [getResult\(\)](#) can be called. If the afterwards obtained result was not correct, you can either set new parameters via [setBeatTrackingParameters\(\)](#) or help [aufTAKT](#) with a clue given via [setKnownDownbeats\(\)](#) before calling this method again.

Returns

Error_t: Returns an error code

reset() `Error_t zplane::aufTAKT::reset ()`

Resets the state of the SDK to the same state as directly after calling [initialize\(\)](#)

Returns

Error_t : Returns an error code.

setBeatTrackingParameters() `Error_t zplane::aufTAKT::setBeatTrackingParameters`

```
(
    float minTempoInBPM,
    float maxTempoInBPM,
    std::vector< int > meters )
```

(Re)set the parameters used for beat estimation.

Parameters

| | |
|----------------------|--|
| <i>minTempoInBPM</i> | Lower bound for tempo for <code>aufTAKT</code> to consider. Must not be lower than 50 BPM. |
| <i>maxTempoInBPM</i> | Upper bound for tempo for <code>aufTAKT</code> to consider. Must not exceed 250 BPM. |
| <i>meters</i> | Vector of meters for <code>aufTAKT</code> to consider. E.g. 3 for 3/4, 4 for 4/4 but also 7 for 7/8. Must not contain elements less than 1 or greater than 10. Note that the computational complexity of parts of <code>aufTAKT</code> scale linearly with the number of beats i.e. with the sum of the elements of this vector. |

Returns

`Error_t`: Returns an error code

setDataChunk() `Error_t zplane::aufTAKT::setDataChunk (`
`void * dataChunk)`

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

Parameters

| | |
|------------------|-----------------------|
| <i>dataChunk</i> | Pointer to data chunk |
|------------------|-----------------------|

Returns

`Error_t`: If data recovery was performed without error, otherwise `kInvalidDataChunk`

setKnownDownbeats() `Error_t zplane::aufTAKT::setKnownDownbeats (`
`const std::vector< KnownDownbeat > & knownDownbeats)`

This method can be used to incorporate any previous knowledge about downbeat positions. We use the aforementioned `KnownDownbeat` struct to represent a single downbeat. This method has to be called before a call to `process()` to reflect the changes in the beat/downbeat estimation. Calling this method successively simply replaces the last list of known downbeats.

Parameters

| | |
|-----------------------|---|
| <i>knownDownbeats</i> | A vector of known or suspected downbeats given as instances of the KnownDownbeat struct |
|-----------------------|---|

Returns

Error_t: Returns an error code.

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

- [aufTAKT/aufTAKT.h](#)

6.2 zplane::aufTAKT::Beat Struct Reference

```
#include <aufTAKT/aufTAKT.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.

6.2.1 Detailed Description

Struct representing instances of beats in a result

Definition at line 135 of file [aufTAKT.h](#).

6.2.2 Member Data Documentation

beatCountInBar unsigned int `zplane::aufTAKT::Beat::beatCountInBar`

beat number in the bar, e.g. 1 for a downbeat etc.

Definition at line 137 of file [aufTAKT.h](#).

numBeatsInBar unsigned int `zplane::aufTAKT::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 138 of file [aufTAKT.h](#).

timeInS float zplane::aufTAKT::Beat::timeInS

time marker corresponding to this beat in seconds

Definition at line 136 of file aufTAKT.h.

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

- [aufTAKT/aufTAKT.h](#)

6.3 zplane::aufTAKT::KnownDownbeat Struct Reference

```
#include <aufTAKT/aufTAKT.h>
```

Public Member Functions

- [KnownDownbeat](#) (float [downbeatTimeInS](#), float [tempoFromTheDownbeatOnwardsInBPM](#), unsigned int [meter=4](#), float [includedTempoRangeDeltaFromDownbeatInBPM=20.f](#))

Public Attributes

- const float [downbeatTimeInS](#)
- const float [tempoFromTheDownbeatOnwardsInBPM](#)
- const unsigned int [meter](#)
- const float [includedTempoRangeDeltaFromDownbeatInBPM](#)

6.3.1 Detailed Description

Struct reflecting an instance of a known downbeat along with the tempo and meter corresponding to it. Needed for the `setKnownDownbeat()` method.

Please note that this struct does not implement any sanity checks of some kind e.g. a `downbeatTimeInS` set to `-1e10f` will not cause an error here, but later on in the `setKnownDownbeat()` method. This is due to the fact that this class does not have any information about the current state of the `aufTAKT` algorithm.

Parameters

| | |
|---|--|
| <i>downbeatTimeInS</i> | Point in time corresponding to the known (or suspected) downbeat. |
| <i>tempoFromTheDownbeatOnwardsInBPM</i> | Tempo corresponding to the known downbeat. |
| <i>meter</i> | Meter corresponding to the known downbeat. E.g. 3 for 3/4 or 4 to represent 4/4. <code>aufTAKT</code> 's prediction will be bound to this given meter at any point after the <code>knownDownbeat</code> and before any following <code>knownDownbeat</code> which might come with another meter. |

Parameters

| | |
|---|--|
| <i>includedTempoRangeDeltaFromDownbeatInBPM</i> | After the known downbeat, aufTAKT's estimation will be bounded in the range <i>tempoFromTheDownbeatOnwardsInBPM +/- includedTempoRangeDeltaFromDownbeatInBPM</i> |
|---|--|

Definition at line 191 of file aufTAKT.h.

6.3.2 Constructor & Destructor Documentation

```
KnownDownbeat() zplane::aufTAKT::KnownDownbeat::KnownDownbeat (
    float downbeatTimeInS,
    float tempoFromTheDownbeatOnwardsInBPM,
    unsigned int meter = 4,
    float includedTempoRangeDeltaFromDownbeatInBPM = 20.f ) [inline]
```

Definition at line 194 of file aufTAKT.h.

```
198         : downbeatTimeInS (downbeatTimeInS),
199           tempoFromTheDownbeatOnwardsInBPM (
tempoFromTheDownbeatOnwardsInBPM), meter (
meter),
200           includedTempoRangeDeltaFromDownbeatInBPM (
includedTempoRangeDeltaFromDownbeatInBPM) {};
```

6.3.3 Member Data Documentation

```
downbeatTimeInS const float zplane::aufTAKT::KnownDownbeat::downbeatTimeInS
```

Definition at line 200 of file aufTAKT.h.

```
includedTempoRangeDeltaFromDownbeatInBPM const float zplane::aufTAKT::KnownDownbeat::includedTempoRangeDeltaFromDownbeatInBPM
```

Definition at line 204 of file aufTAKT.h.

```
meter const unsigned int zplane::aufTAKT::KnownDownbeat::meter
```

Definition at line 203 of file aufTAKT.h.

```
tempoFromTheDownbeatOnwardsInBPM const float zplane::aufTAKT::KnownDownbeat::tempoFromTheDownbeatOnwardsInBPM
```

Definition at line 202 of file aufTAKT.h.

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

- aufTAKT/[aufTAKT.h](#)

7 File Documentation

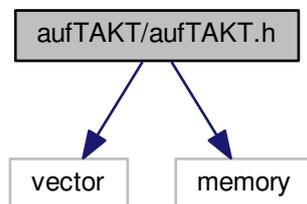
7.1 /work/project/docs/docugen.txt File Reference

7.2 aufTAKT/aufTAKT.h File Reference

interface of the aufTAKT class.

```
#include <vector>
#include <memory>
```

Include dependency graph for aufTAKT.h:



Classes

- class [zplane::aufTAKT](#)
- struct [zplane::aufTAKT::Beat](#)
- struct [zplane::aufTAKT::KnownDownbeat](#)

Namespaces

- [zplane](#)

7.2.1 Detailed Description

interface of the aufTAKT class.

:

Index

`/work/project/docs/docugen.txt`, 17
~aufTAKT
 zplane::aufTAKT, 9

aufTAKT/aufTAKT.h, 17
aufTAKT
 zplane::aufTAKT, 9

beatCountInBar
 zplane::aufTAKT::Beat, 14

clearKnownDownbeats
 zplane::aufTAKT, 9

downbeatTimeInS
 zplane::aufTAKT::KnownDownbeat,
 16

Error_t
 zplane::aufTAKT, 8

finishPreProcessing
 zplane::aufTAKT, 9

getBuildDate
 zplane::aufTAKT, 10

getDataChunk
 zplane::aufTAKT, 10

getDataChunkSizeInBytes
 zplane::aufTAKT, 10

getOverallTempo
 zplane::aufTAKT, 10

getResult
 zplane::aufTAKT, 11

getVersion
 zplane::aufTAKT, 11

includedTempoRangeDeltaFromDownbeat↔
 InBPM
 zplane::aufTAKT::KnownDownbeat,
 16

initialize
 zplane::aufTAKT, 11

isInitialized
 zplane::aufTAKT, 12

KnownDownbeat
 zplane::aufTAKT::KnownDownbeat,
 16

meter
 zplane::aufTAKT::KnownDownbeat,
 16

numBeatsInBar
 zplane::aufTAKT::Beat, 14

preProcess
 zplane::aufTAKT, 12

process
 zplane::aufTAKT, 12

reset
 zplane::aufTAKT, 12

setBeatTrackingParameters
 zplane::aufTAKT, 12

setDataChunk
 zplane::aufTAKT, 13

setKnownDownbeats
 zplane::aufTAKT, 13

tempoFromTheDownbeatOnwardsInBPM
 zplane::aufTAKT::KnownDownbeat,
 16

timeInS
 zplane::aufTAKT::Beat, 14

zplane, 6
zplane::aufTAKT::Beat, 14
 beatCountInBar, 14
 numBeatsInBar, 14
 timeInS, 14
zplane::aufTAKT::KnownDownbeat, 15
 downbeatTimeInS, 16
 includedTempoRangeDeltaFromDownbeat↔
 InBPM, 16
 KnownDownbeat, 16
 meter, 16
 tempoFromTheDownbeatOnwards↔
 InBPM, 16
zplane::aufTAKT, 7
 ~aufTAKT, 9
 aufTAKT, 9
 clearKnownDownbeats, 9
 Error_t, 8
 finishPreProcessing, 9
 getBuildDate, 10

[getDataChunk, 10](#)
[getDataChunkSizeInBytes, 10](#)
[getOverallTempo, 10](#)
[getResult, 11](#)
[getVersion, 11](#)
[initialize, 11](#)
[isInitialized, 12](#)
[preProcess, 12](#)
[process, 12](#)
[reset, 12](#)
[setBeatTrackingParameters, 12](#)
[setDataChunk, 13](#)
[setKnownDownbeats, 13](#)