
# INTONE 1.1.1

by zplane.development



August 30, 2023

# Contents

# 1 INTONE Documentation

## 1.1 Introduction

INTONE is a formant-preserving, low-latency pitch correction algorithm designed to work in real time with monophonic vocal and instrumental recordings.

INTONE is controlled by multiple parameters that enable a great palette of sonic results, ranging from natural-sounding correction that preserves expressive characteristics such as vibrato, all the way to the hyper-corrected vocals known and loved by fans of pop, hip hop, and trap.

This guide contains the information you need to get started using INTONE: it provides an API description, C++ usage examples, and illustrations of how the parameters affect pitch correction to achieve your desired results. At the end, you will also find detailed documentation of the INTONE interface, describing all its methods and structs.

## 1.2 API Documentation

### 1.2.1 Memory Allocation

The INTONE SDK does not allocate buffers handled by the calling application. The input buffers have to be allocated by the calling application. Audio buffers are allocated as double arrays of [channels][SamplesPerChannel].

### 1.2.2 Naming Conventions

When talking about frames, the number of audio samples per channel is meant. I.e. 512 stereo frames correspond to 1024 float values (samples). If the sample size is 32bit float, one sample has a memory usage of 4 bytes.

### 1.2.3 Required Functions

The following methods have to be used (in the order they appear here) to obtain pitch correction results from INTONE. For more details, see the individual method documentation and the example below.

- **Intone::initialize (float sampleRate, int numberOfChannels, int maxBlock↩ Size);** Initalizes an INTONE instance. See parameters below. Default values output uncorrected pitches.
- **Intone::process (float const∗ const∗ const inputBuffer, float∗∗ outputBuffer, int numberOfFrames);** Feed audio data (samples) to INTONE, obtaining a buffer with corrected audio as output.

### 1.2.4 C++ Usage example

The complete code referenced here can be found in the example source file IntoneClMain.cpp. We first create an INTONE instance. After instantiation, we can use one of two variants of the **Intone::initialize()** method to prepare INTONE for use. Note that the relationship `samplingFrequency / maxBlockSize` determines the lowest detectable frequency (i.e., what is the largest wave cycle that fits into one block?) With this in mind, we can initialize the instance directly with an `int maxBlock↩ Size` or with a float value for the lowest frequency in Hertz expected in the input

signal. Internally, INTONE will choose the lowest power-of-two block size that covers the specified lowest frequency in Hz. To initialize by specifying a block size, use:

```
Intone IntoneInstance;
IntoneInstance.initialize (inputFile.GetSampleRate(),
                           inputFile.GetNumOfChannels(),
                           static_cast<int> (maxBlockSize));
```

Otherwise, specify a lowest detectable frequency (100.f in this example), and a maximum block size. Note that there is a fifth implicit parameter (set to `false` by default), `allowVariableInputBlocksizes`.

```
    IntoneInstance.initialize (inputFile.GetSampleRate(),
                               inputFile.GetNumOfChannels(),
                               100.f,
                               maxBlockSize);
```

This initializes the instance with default parameters, which will not correct input pitches. After the instance was initialized with a lower frequency bound, we can check the resulting internal block size:

```
    int maxBlkSize = IntoneInstance.getMaxBlockSize();
```

Conversely, we can also check the lowest detectable frequency, given the instance's max block size:

```
    float lowestFreq = IntoneInstance.getLowestDetectableFrequency();
```

In order to obtain pitch-corrected audio, we first set the desired correction amount to a float value between 0 and 1. A value of 1 has the strictest, most noticeable correction effect.

```
    IntoneInstance.setCorrectionAmount (correctionAmount);
```

If you do not specify a pitch mapping, INTONE will adhere to the chromatic scale, rounding input pitches to the closest MIDI pitch. If you wish to correct pitches according to a scale, we provide a pitch mapping helper with simple major and minor scales. In this example we use the F sharp (fis) major scale:

```
    zplane::PitchMappingHelper::GeneratePitchMapping
                               (IntoneInstance, "fis", "major");
```

For more fine-grained control, you can enable and disable individual pitch classes using the function **Intone::setPitchClassAllowed (PitchClass pitchClass, bool isAllowed)**.

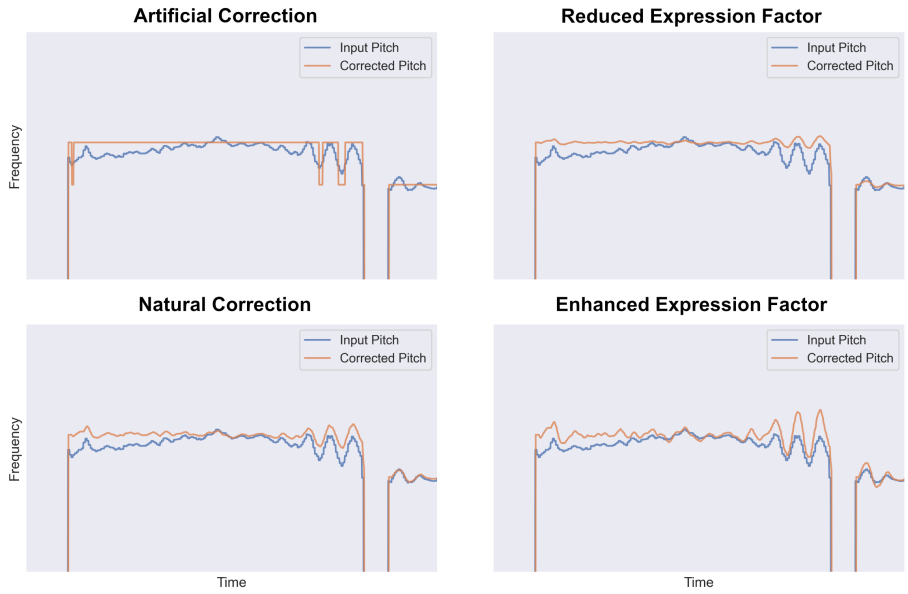



Figure 1: Pitch Correction Options

Depending on your use case, you may want to achieve either natural-sounding results that preserve expression, or more artificial output reminiscent of chart-topping artists. INTONE provides the ability to correct the *baseline* or underlying pitch while preserving natural expression. The expression factor controls this behavior, which you can set via **Intone::setExpressionFactor (float expressionFactor)**. To better visualize how it influences the output, see Figure 1. In the top-left panel we show an artificial-sounding example which can be achieved with the maximum correction amount of 1.0, coupled with an expression factor of 0. In the top-right panel we show an example where the slightly out-of-tune input pitch (blue curve) is corrected and the vibrato extension is reduced to half of the original, using an expression factor of 0.5. In the bottom-left panel we show an example of natural correction, where the baseline pitch has been corrected and the full amount of vibrato is preserved, using the following settings:

```
float correctionAmount = 1.0f;
IntoneInstance.setCorrectionAmount (correctionAmount);
float expressionFactor = 1.0f;
IntoneInstance.setExpressionFactor (expressionFactor);
```

Finally, in the bottom-right panel, we show how vibrato can be enhanced or incremented beyond the original amount with an expression factor of 2. If you wish to first transpose the signal, before applying pitch correction, you can do so by setting a global offset in semitones – an offset of 0 will leave the signal unaffected:

```
float globalShift = 0.f;
IntoneInstance.setGlobalOffset (globalShift);
```

INTONE is initialized by default with a standard tuning frequency of 440.0 Hz. If you wish to set a different value, you can use the following function:

```
IntoneInstance.setTuningFrequency (450.f);
```

### 1.3 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 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 zplane Namespace Reference

### Classes

- class Intone
- class PitchMappingHelper

# 6 Class Documentation

## 6.1 zplane::Intone Class Reference

`#include <Intone/Intone.h>`

### Public Types

- enum Error_t {
  kNoError, kMemError, kInvalidFunctionParamError, kNotInitializedError,
  kAlreadyInitializedError, kUnknownError, kNumErrors }
- enum PitchClass {
  C, CSharp, D, DSharp,
  E, F, FSharp, G,
  GSharp, A, ASharp, B,
  kNumOfPitchClasses }

### Public Member Functions

- Intone ()
- ~Intone ()
- Error_t initialize (float sampleRate, int numberOfChannels, float minFreqHz, int desiredInputBlocksize, bool allowVariableInputBlocksizes=false)

  *Initialize an instance. Must be called before using any functionality.*
- bool isInitialized ()

  *Is the Intone instance initialized?*
- Error_t process (float const *const *const inputBuffer, float **outputBuffer, int numberOfFrames)

  *Performs pitch correction on the input buffer, returning the results in the output buffer.*
- int getLatencyInSamples ()

  *Returns the algorithm latency in samples.*
- int getMaxBlockSize ()

  *Returns the maximum allowed block size that the instance was initialized with.*
- float getLowestDetectableFrequency ()

  *Returns the lowest frequency that can be detected given the current block size. Lower frequencies have larger periods, which require larger block sizes.*
- Error_t setCorrectionAmount (float newAmount)

  *: Set how strictly divergent pitches will be rounded to the corresponding MIDI pitch. Use together with correction naturalness and expression factor.*

- Error_t setPitchClassAllowed (PitchClass pitchClass, bool isAllowed)

  *Enable or disable a pitch class to behave as a valid target for quantization.*
- bool getPitchClassAllowed (PitchClass pitchClass)

  *Query wheter a certain pitch class is currently allowed or disabled.*
- Error_t setExpressionFactor (float expressionFactor)

  *Control the amount of expressive deviation (vibrato) from the center of a corrected note. Use together with correction amount and correction naturalness.*
- Error_t setGlobalOffset (float offsetInSemitones)

  *Apply a global pitch offset (in semitones) to the input, before pitch correction.*
- float getTuningFrequency ()

  *Get the tuning frequency, instance is initialized to 440.0 Hz by default.*
- Intone::Error_t setTuningFrequency (float tuningFrequency)

  *Set the tuning frequency. Default at initialization is 440.0 Hz.*

### Static Public Member Functions

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

#### 6.1.1 Detailed Description

Definition at line 39 of file Intone.h.

#### 6.1.2 Member Enumeration Documentation

**Error_t** `enum zplane::Intone::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 |
| kAlreadyInitializedError | instance has already been initialized |
| kUnknownError | unknown error occurred |
| kNumErrors | |

Definition at line 43 of file Intone.h.

```
44         {
45             kNoError,
46             kMemError,
47             kInvalidFunctionParamError,
48             kNotInitializedError,
49             kAlreadyInitializedError,
50             kUnknownError,
51             kNumErrors
52         };
```

**PitchClass** `enum zplane::Intone::PitchClass`

Enumerator

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

Definition at line 54 of file Intone.h.

```
55          {
56              C,
57              CSharp,
58              D,
59              DSharp,
60              E,
61              F,
62              FSharp,
63              G,
64              GSharp,
65              A,
66              ASharp,
67              B,
68              kNumOfPitchClasses
69          };
```

### 6.1.3 Constructor & Destructor Documentation

**Intone()** `zplane::Intone::Intone ( )`

**~Intone()** `zplane::Intone::~Intone ( )`

### 6.1.4 Member Function Documentation

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

**getLatencyInSamples()** `int zplane::Intone::getLatencyInSamples ( )`

Returns the algorithm latency in samples.

Returns

int

**getLowestDetectableFrequency()** `float zplane::Intone::getLowestDetectable↩ Frequency ( )`

Returns the lowest frequency that can be detected given the current block size. Lower frequencies have larger periods, which require larger block sizes.

Returns

int

**getMaxBlockSize()** `int zplane::Intone::getMaxBlockSize ( )`

Returns the maximum allowed block size that the instance was initialized with.

Returns

int

**getPitchClassAllowed()** `bool zplane::Intone::getPitchClassAllowed ( PitchClass pitchClass )`

Query wheter a certain pitch class is currently allowed or disabled.

Parameters

| | |
|---|---|
| *pitchClass* | The pitch class to be queried. |

Returns

bool: True or False

**getTuningFrequency()** `float zplane::Intone::getTuningFrequency ( )`

Get the tuning frequency, instance is initialized to 440.0 Hz by default.

Returns

float

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

**initialize()** `Error_t zplane::Intone::initialize (`
`float sampleRate,`
`int numberOfChannels,`
`float minFreqHz,`
`int desiredInputBlocksize,`
`bool allowVariableInputBlocksizes = false )`

Initialize an instance. Must be called before using any functionality.

Parameters

| | |
|---|---|
| *sampleRate* | Sample rate of the input signal in Hertz. |
| *numberOfChannels* | Number of channels in the input signal. |
| *minFreqHz* | Lowest expected frequency in Hz, translates internally to a power-of-two block size. |
| *desiredInputBlocksize* | Maximum block size in samples that will be passed to the instance. |
| *allowVariableInputBlocksizes* | enable varying input blocksize up until the desiredInputBlocksize, this results in a higher latency though |

Returns

Error_t: returns an error or noError

**isInitialized()** `bool zplane::Intone::isInitialized ( )`

Is the Intone instance initialized?

Returns

true
false

**process()** `Error_t zplane::Intone::process (`
`float const *const *const inputBuffer,`
`float ** outputBuffer,`
`int numberOfFrames )`

Performs pitch correction on the input buffer, returning the results in the output buffer.

Parameters

| | |
|---|---|
| *inputBuffer* | double pointer to the input buffer of samples [channels][samples] |
| *outputBuffer* | double pointer to the output buffer of samples [channels][samples] |
| *numberOfFrames* | the number of input frames |

Returns

Error_t: returns an error or noError

**setCorrectionAmount()** `Error_t zplane::Intone::setCorrectionAmount ( float newAmount )`

: Set how strictly divergent pitches will be rounded to the corresponding MIDI pitch. Use together with correction naturalness and expression factor.

Parameters

| | |
|---|---|
| *newAmount* | Desired amount of pitch correction, between 0 and 1. A value of 1 has the strictest, most noticeable correction effect. |

Returns

Error_t: returns an error or noError

**setExpressionFactor()** `Error_t zplane::Intone::setExpressionFactor ( float expressionFactor )`

Control the amount of expressive deviation (vibrato) from the center of a corrected note. Use together with correction amount and correction naturalness.

Parameters

| | |
|---|---|
| *expressionFactor* | Float between 0.f and 2.0f: 0 removes vibrato; 1 preserves the original amount; values greater than 1 will exaggerate expression. |

Returns

Error_t

**setGlobalOffset()** `Error_t zplane::Intone::setGlobalOffset ( float offsetInSemitones )`

Apply a global pitch offset (in semitones) to the input, before pitch correction.

Parameters

| | |
|---|---|
| *offsetInSemitones* | Desired global shift in semitones. |

Returns

Error_t

**setPitchClassAllowed()** `Error_t zplane::Intone::setPitchClassAllowed ( PitchClass pitchClass, bool isAllowed )`

Enable or disable a pitch class to behave as a valid target for quantization.

Parameters

| | |
|---|---|
| *pitchClass* | The pitch class to be affected. |
| *isAllowed* | true if notes should be quantized to this pitch class, false otherwise. |

Returns

Error_t: returns an error or noError

Referenced by zplane::PitchMappingHelper::GeneratePitchMapping().

**setTuningFrequency()** `Intone::Error_t zplane::Intone::setTuningFrequency ( float tuningFrequency )`

Set the tuning frequency. Default at initialization is 440.0 Hz.

Parameters

| | |
|---|---|
| *tuningFrequency* | |

Returns

Error_t if tuningFrequency is equal to or lower than zero

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

- Intone/Intone.h

## 6.2 zplane::PitchMappingHelper Class Reference

`#include <Intone/PitchMappingHelper.h>`

### Static Public Member Functions

- static Intone::Error_t GeneratePitchMapping (Intone &pInstance, std::string key, std::string targetModeName)

### 6.2.1 Detailed Description

Definition at line 51 of file PitchMappingHelper.h.

### 6.2.2 Member Function Documentation

**GeneratePitchMapping()** `static Intone::Error_t zplane::PitchMappingHelper↩
::GeneratePitchMapping (`

`Intone & pInstance,`

`std::string key,`

`std::string targetModeName ) [inline], [static]`

Definition at line 55 of file PitchMappingHelper.h.

References zplane::Intone::kInvalidFunctionParamError, zplane::Intone::kNoError, and zplane::Intone::setPitchClassAllowed().

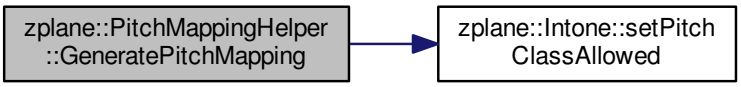
```
56              {
57                  Intone::Error_t errorOut = Intone::kNoError;
58
59                  std::string keyNames[12] = { "c", "cis", "d", "dis", "e", "f", "fis", "g", "gis", "a", "ais
    ", "b" };
60                  int modeMajor[] = { 0, 2, 4, 5, 7, 9, 11 };
61                  int modeMinor[] = { 0, 2, 3, 5, 7, 8, 10 };
62
63                  std::map<std::string, std::vector<int> > modes;
64
65                  modes["major"] = std::vector<int> (modeMajor, modeMajor + sizeof (modeMajor) / sizeof (
    modeMajor[0]));
66                  modes["minor"] = std::vector<int> (modeMinor, modeMinor + sizeof (modeMinor) / sizeof (
    modeMinor[0]));
67
68                  // validate the given key, source mode and target mode
69                  if (std::find (keyNames, keyNames + 12, key) == keyNames + 12)
70                  {
71                      errorOut = Intone::kInvalidFunctionParamError;

72                  }
73
74                  if (modes.find (targetModeName) == modes.end())
75                  {
76                      errorOut = Intone::kInvalidFunctionParamError;

77                  }
78
79                  std::vector<int> targetMode = modes[targetModeName];
80
81                  int keyIndex = static_cast<int> (std::distance (keyNames, std::find (keyNames, keyNames + 1
    2, key)));
82
83                  // reset to chromatic before applying new rules
84                  for (int i = 0; i < 12; i++)
85                  {
86                      pInstance.setPitchClassAllowed (static_cast<Intone::PitchClass> (i), true);
87                  }
88
89                  for (int sourceNote = 0; sourceNote < 12; sourceNote++)
90                  {
91                      auto it = std::lower_bound(targetMode.begin(), targetMode.end(), sourceNote);
92
93                      int semitoneDifference = 0;
94
95                      it != targetMode.end()
96                          ? semitoneDifference = (sourceNote - *it)
97                          : semitoneDifference = INT_MAX;
98
99                      if ( (semitoneDifference) > targetMode.back())
100                      {
101                          pInstance.setPitchClassAllowed (static_cast<Intone::PitchClass> ((sourceNote +
    keyIndex) % 12), false);
102                      }
103                      else if (semitoneDifference != 0)
104                      {
```

```
105                         pInstance.setPitchClassAllowed (static_cast<Intone::PitchClass> ((sourceNote +
    keyIndex) % 12), false);
106                         pInstance.setPitchClassAllowed (static_cast<Intone::PitchClass> ((sourceNote +
    keyIndex + static_cast<int>(semitoneDifference)) % 12), true);
107                     }
108
109                 }
110
111                 return errorOut;
112
113             };
```

Here is the call graph for this function:




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

- Intone/PitchMappingHelper.h

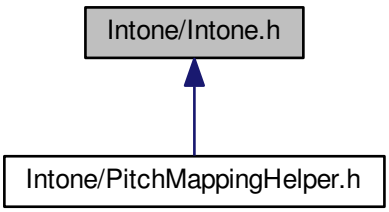
# 7 File Documentation

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

## 7.2 Intone/Intone.h File Reference

interface of the Intone class.

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



### Classes

- class zplane::Intone

### Namespaces

- zplane

#### 7.2.1 Detailed Description

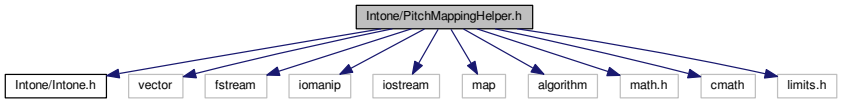
interface of the Intone class.
:

## 7.3 Intone/PitchMappingHelper.h File Reference

Pitch Mapping Helper.

```
#include "Intone/Intone.h"
#include <vector>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <algorithm>
#include <math.h>
#include <cmath>
#include <limits.h>
```

Include dependency graph for PitchMappingHelper.h:




### Classes

- class zplane::PitchMappingHelper

### Namespaces

- zplane

### Macros

- #define _USE_MATH_DEFINES

#### 7.3.1 Detailed Description

Pitch Mapping Helper.
:

### 7.3.2 Macro Definition Documentation

**_USE_MATH_DEFINES** `#define _USE_MATH_DEFINES`
Definition at line 45 of file PitchMappingHelper.h.

# Index