

HOTSPOT SS18 Wie klingt ein Bild?

6

Erzeugt von Doxygen 1.8.14

Inhaltsverzeichnis

1	Verzeichnis der Namensbereiche	1
1.1	Liste aller Namensbereiche	1
2	Klassen-Verzeichnis	3
2.1	Auflistung der Klassen	3
3	Datei-Verzeichnis	5
3.1	Auflistung der Dateien	5
4	Dokumentation der Namensbereiche	7
4.1	bitmap-Namensbereichsreferenz	7
4.1.1	Ausführliche Beschreibung	7
4.2	Processing-Namensbereichsreferenz	7
4.2.1	Ausführliche Beschreibung	8
4.2.2	Dokumentation der Aufzählungstypen	8
4.2.2.1	ProcessingAlgorithm	8
5	Klassen-Dokumentation	9
5.1	Processing::AdditionalSettings Strukturreferenz	9
5.1.1	Ausführliche Beschreibung	10
5.1.2	Dokumentation der Datenelemente	10
5.1.2.1	settings1	10
5.1.2.2	settings2	10
5.1.2.3	settings3	10
5.1.2.4	settings4	11

5.2	bitmap::BitmapFileHeader Strukturreferenz	11
5.2.1	Ausführliche Beschreibung	11
5.2.2	Dokumentation der Datenelemente	11
5.2.2.1	bfFileSize	12
5.2.2.2	bfRes1	12
5.2.2.3	bfRes2	12
5.2.2.4	bfStartAdress	12
5.2.2.5	bfType	12
5.3	bitmap::BitmapInfoHeader Strukturreferenz	12
5.3.1	Ausführliche Beschreibung	13
5.3.2	Dokumentation der Datenelemente	13
5.3.2.1	biBitsPerPixel	13
5.3.2.2	biColorPlanes	13
5.3.2.3	biColorsPerColorPalette	14
5.3.2.4	biCompression	14
5.3.2.5	biHeight	14
5.3.2.6	biHorizontalResolution	14
5.3.2.7	biImageSize	14
5.3.2.8	biImportantColorsUsed	14
5.3.2.9	biSize	15
5.3.2.10	biVerticalResolution	15
5.3.2.11	biWidth	15
5.4	Image Klassenreferenz	15
5.4.1	Ausführliche Beschreibung	16
5.4.2	Beschreibung der Konstruktoren und Destruktoren	16
5.4.2.1	Image() [1/3]	16
5.4.2.2	Image() [2/3]	16
5.4.2.3	Image() [3/3]	17
5.4.2.4	~Image()	17
5.4.3	Dokumentation der Elementfunktionen	17

5.4.3.1	getHeight()	17
5.4.3.2	getPixel()	17
5.4.3.3	getWidth()	18
5.4.3.4	operator=() [1/2]	18
5.4.3.5	operator=() [2/2]	18
5.4.3.6	readHeader()	18
5.4.3.7	readImageData()	18
5.4.4	Dokumentation der Datenelemente	19
5.4.4.1	bitmapData	19
5.4.4.2	bitmapFileHeader	19
5.4.4.3	bitmapInfoHeader	19
5.4.4.4	file	19
5.5	Pixel Strukturreferenz	19
5.5.1	Ausführliche Beschreibung	20
5.5.2	Dokumentation der Datenelemente	20
5.5.2.1	b	20
5.5.2.2	g	20
5.5.2.3	r	20
5.6	Processing::Processing Klassenreferenz	21
5.6.1	Ausführliche Beschreibung	22
5.6.2	Beschreibung der Konstruktoren und Destruktoren	22
5.6.2.1	Processing() [1/2]	22
5.6.2.2	Processing() [2/2]	22
5.6.3	Dokumentation der Elementfunktionen	22
5.6.3.1	lrScan()	22
5.6.3.2	lrScan_no_threshold()	23
5.6.3.3	operator=() [1/2]	23
5.6.3.4	operator=() [2/2]	23
5.6.3.5	start()	23
5.6.3.6	triplet()	24

5.6.3.7	triplet_jump()	24
5.6.3.8	ud_lr_scan()	24
5.6.3.9	udScan()	25
5.6.3.10	udScan_no_threshold()	25
5.6.4	Dokumentation der Datenelemente	25
5.6.4.1	image	25
5.6.4.2	sound	25
5.6.4.3	wave	26
5.7	Processing::ProcessingSettings Strukturreferenz	26
5.7.1	Ausführliche Beschreibung	26
5.7.2	Dokumentation der Datenelemente	26
5.7.2.1	audioVolume	26
5.7.2.2	samples	26
5.7.2.3	samplingFrequency	27
5.8	Sound Klassenreferenz	27
5.8.1	Ausführliche Beschreibung	27
5.8.2	Beschreibung der Konstruktoren und Destruktoren	28
5.8.2.1	Sound() [1/2]	28
5.8.2.2	Sound() [2/2]	28
5.8.2.3	~Sound()	28
5.8.3	Dokumentation der Elementfunktionen	28
5.8.3.1	addFrequency()	28
5.8.3.2	getBuffer()	29
5.8.3.3	resetBuffer()	29
5.8.4	Dokumentation der Datenelemente	29
5.8.4.1	audioVolume	29
5.8.4.2	buffer	29
5.8.4.3	bufferSize	30
5.8.4.4	lastPos	30
5.8.4.5	samplingFrequency	30

5.9	Wave Klassenreferenz	30
5.9.1	Ausführliche Beschreibung	31
5.9.2	Beschreibung der Konstruktoren und Destruktoren	31
5.9.2.1	Wave() [1/2]	31
5.9.2.2	Wave() [2/2]	32
5.9.2.3	~Wave()	32
5.9.3	Dokumentation der Elementfunktionen	32
5.9.3.1	closeFile()	32
5.9.3.2	finishHeader()	32
5.9.3.3	prepareHeader()	32
5.9.3.4	writeSamples()	32
5.9.4	Dokumentation der Datenelemente	33
5.9.4.1	audioFormat	33
5.9.4.2	bitsPerSample	33
5.9.4.3	blockAlign	33
5.9.4.4	byteRate	33
5.9.4.5	file	34
5.9.4.6	numChannels	34
5.9.4.7	sampleRate	34
5.9.4.8	subChunk1Size	34
5.9.4.9	waveHeader	34
6	Datei-Dokumentation	35
6.1	Source/Image.cpp-Dateireferenz	35
6.2	Source/Image.h-Dateireferenz	35
6.3	Source/Processing.cpp-Dateireferenz	36
6.3.1	Makro-Dokumentation	36
6.3.1.1	PRINT_PROGRESS	36
6.3.2	Dokumentation der Funktionen	36
6.3.2.1	printProgress()	36
6.4	Source/Processing.h-Dateireferenz	36
6.5	Source/Quelle.cpp-Dateireferenz	37
6.5.1	Dokumentation der Funktionen	37
6.5.1.1	main()	37
6.6	Source/Sound.cpp-Dateireferenz	38
6.6.1	Makro-Dokumentation	38
6.6.1.1	_USE_MATH_DEFINES	38
6.7	Source/Sound.h-Dateireferenz	38
6.8	Source/Wave.cpp-Dateireferenz	38
6.9	Source/Wave.h-Dateireferenz	38
	Index	39

Kapitel 1

Verzeichnis der Namensbereiche

1.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

bitmap	Verwaltungsinformationen zu Bitmap Dateien	7
Processing	Beeinhaltet Datenstrukturen zur Umwandlung von Bildinformationen in Toene	7

Kapitel 2

Klassen-Verzeichnis

2.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Processing::AdditionalSettings	
Zusaetzliche Einstellungen spezifisch fuer die verschienden Algorithmen	9
bitmap::BitmapFileHeader	
Dateiheader fuer .bmp	11
bitmap::BitmapInfoHeader	
Bitmapheader. Enthael Informationen ueber die Eigenschaften des Bildes	12
Image	
Klasse zum oeffnen und auslesen von Bitmap Bilddateien	15
Pixel	
Pixel 8 Bit RGB Werte	19
Processing::Processing	
Klasse zur Umwandlung von Bilddaten in Tondaten	21
Processing::ProcessingSettings	
Grundeinstellungen fuer die Tonerzeugung	26
Sound	
Klasse zur Erzeugung von Toenen	27
Wave	
Klasse zur Erzeugung von .wav Audiodateien	30

Kapitel 3

Datei-Verzeichnis

3.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Source/Image.cpp	35
Source/Image.h	35
Source/Processing.cpp	36
Source/Processing.h	36
Source/Quelle.cpp	37
Source/Sound.cpp	38
Source/Sound.h	38
Source/Wave.cpp	38
Source/Wave.h	38

Kapitel 4

Dokumentation der Namensbereiche

4.1 bitmap-Namensbereichsreferenz

Verwaltungsinformationen zu Bitmap Dateien.

Klassen

- struct [BitmapFileHeader](#)
Dateiheader fuer .bmp.
- struct [BitmapInfoHeader](#)
Bitmapheader. Enthaelte Informationen ueber die Eigenschaften des Bildes.

4.1.1 Ausführliche Beschreibung

Verwaltungsinformationen zu Bitmap Dateien.

4.2 Processing-Namensbereichsreferenz

Beeinhaltet Datenstrukturen zur Umwandlung von Bildinformationen in Toene.

Klassen

- struct [AdditionalSettings](#)
Zusaetzliche Einstellungen spezifisch fuer die verschienden Algorithmen.
- class [Processing](#)
Klasse zur Umwandlung von Bilddaten in Tondaten.
- struct [ProcessingSettings](#)
Grundeinstellungen fuer die Tonerzeugung.

Aufzählungen

- enum `ProcessingAlgorithm` {
 `ProcessingAlgorithm::LR_SCAN` = 0, `ProcessingAlgorithm::LR_SCAN_NO_THRESHOLD`, `ProcessingAlgorithm::UD_SCAN`,
 `ProcessingAlgorithm::UD_SCAN_NO_THRESHOLD`,
 `ProcessingAlgorithm::TRIPLET`, `ProcessingAlgorithm::TRIPLET_JMP`, `ProcessingAlgorithm::UD_LR_SCAN`
}

Verfuegbare Algorithmen.

4.2.1 Ausführliche Beschreibung

Beeinhaltet Datenstrukturen zur Umwandlung von Bildinformationen in Toene.

4.2.2 Dokumentation der Aufzählungstypen

4.2.2.1 ProcessingAlgorithm

```
enum Processing::ProcessingAlgorithm [strong]
```

Verfuegbare Algorithmen.

Aufzählungswerte

LR_SCAN	
LR_SCAN_NO_THRESHOLD	
UD_SCAN	
UD_SCAN_NO_THRESHOLD	
TRIPLET	
TRIPLET_JMP	
UD_LR_SCAN	

Kapitel 5

Klassen-Dokumentation

5.1 Processing::AdditionalSettings Strukturreferenz

Zusaetzliche Einstellungen spezifisch fuer die verschiedenen Algorithmen.

```
#include <Processing.h>
```

Öffentliche Attribute

- uint64_t [settings1](#) = 0
Zusaetzliche Einstellung spezifisch fuer die verschiedenen Algorithmen
LR_SCAN: Maximale Frequenz
LR_SCAN_NO_THRESHOLD: Maximale Frequenz
UD_SCAN: Maximale Frequenz
UD_SCAN_NO_THRESHOLD: Maximale Frequenz
TRIPLET: -
TRIPLET_JMP: -
UD_LR_SCAN: Maximale Frequenz.
- uint64_t [settings2](#) = 0
Zusaetzliche Einstellung spezifisch fuer die verschiedenen Algorithmen
LR_SCAN: Aktivierungsschwelle fuer [Pixel](#)
LR_SCAN_NO_THRESHOLD: -
UD_SCAN: Aktivierungsschwelle fuer [Pixel](#)
UD_SCAN_NO_THRESHOLD: -
TRIPLET: Tondauer abhaengig von dem Blauanteil (0-255)
TRIPLET_JMP: Tondauer abhaengig von der Pixelintensitaet
UD_LR_SCAN: Tondauer abhaengig vom Blauanteil.
- uint64_t [settings3](#) = 0
Zusaetzliche Einstellung spezifisch fuer die verschiedenen Algorithmen
LR_SCAN: -
LR_SCAN_NO_THRESHOLD: -
UD_SCAN: -
UD_SCAN_NO_THRESHOLD: -
TRIPLET: Anzahl der Triplets die man pro Frequenz ueberspringen soll
TRIPLET_JMP: Anzahl der Zyklen durch die Bilddaten (z.B. 128)
UD_LR_SCAN: -.
- uint64_t [settings4](#) = 0
Zusaetzliche Einstellung spezifisch fuer die verschiedenen Algorithmen
LR_SCAN: [Pixel](#) invertieren
LR_SCAN_NO_THRESHOLD: [Pixel](#) invertieren
UD_SCAN: [Pixel](#) invertieren
UD_SCAN_NO_THRESHOLD: [Pixel](#) invertieren
TRIPLET: [Pixel](#) invertieren
TRIPLET_JMP: [Pixel](#) invertieren
UD_LR_SCAN: [Pixel](#) invertieren.

5.1.1 Ausführliche Beschreibung

Zusaetzliche Einstellungen spezifisch fuer die verschienden Algorithmen.

5.1.2 Dokumentation der Datenelemente

5.1.2.1 settings1

```
uint64_t Processing::AdditionalSettings::settings1 = 0
```

Zusaetzliche Einstellung spezifisch fuer die verschiedenen Algorithmen

LR_SCAN: Maximale Frequenz

LR_SCAN_NO_THRESHOLD: Maximale Frequenz

UD_SCAN: Maximale Frequenz

UD_SCAN_NO_THRESHOLD: Maximale Frequenz

TRIPLET: -

TRIPLET_JMP: -

UD_LR_SCAN: Maximale Frequenz.

5.1.2.2 settings2

```
uint64_t Processing::AdditionalSettings::settings2 = 0
```

Zusaetzliche Einstellunge spezifisch fuer die verschiedenen Algorithmen

LR_SCAN: Aktivierungsschwelle fuer [Pixel](#)

LR_SCAN_NO_THRESHOLD: -

UD_SCAN: Aktivierungsschwelle fuer [Pixel](#)

UD_SCAN_NO_THRESHOLD: -

TRIPLET: Tondauer abhaengig von dem Blauanteil (0-255)

TRIPLET_JMP: Tondauer abhaengig von der Pixelintensitaet

UD_LR_SCAN: Tondauer abhaengig vom Blauanteil.

5.1.2.3 settings3

```
uint64_t Processing::AdditionalSettings::settings3 = 0
```

Zusaetzliche Einstellunge spezifisch fuer die verschiedenen Algorithmen

LR_SCAN: -

LR_SCAN_NO_THRESHOLD: -

UD_SCAN: -

UD_SCAN_NO_THRESHOLD: -

TRIPLET: Anzahl der Triplets die man pro Frequenz ueberspringen soll

TRIPLET_JMP: Anzahl der Zyklen durch die Bilddaten (z.B. 128)

UD_LR_SCAN: -.

5.1.2.4 settings4

```
uint64_t Processing::AdditionalSettings::settings4 = 0
```

Zusaetzliche Einstellunge spezifisch fuer die verschiedenen Algorithmen

LR_SCAN: [Pixel](#) invertieren

LR_SCAN_NO_THRESHOLD: [Pixel](#) invertieren

UD_SCAN: [Pixel](#) invertieren

UD_SCAN_NO_THRESHOLD: [Pixel](#) invertieren

TRIPLER: [Pixel](#) invertieren

TRIPLER_JMP: [Pixel](#) invertieren

UD_LR_SCAN: [Pixel](#) invertieren.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Source/[Processing.h](#)

5.2 bitmap::BitmapFileHeader Strukturreferenz

Dateiheader fuer .bmp.

```
#include <Image.h>
```

Öffentliche Attribute

- uint16_t [bfType](#)
ASCII Zeichen "BM" notwendig.
- uint32_t [bfFileSize](#)
Groesse der .bmp Datei in bytes.
- uint16_t [bfRes1](#)
Muss 0 sein.
- uint16_t [bfRes2](#)
Muss 0 sein.
- uint32_t [bfStartAddress](#)
Offset zu den Bilddaten von Beginn der Datei.

5.2.1 Ausführliche Beschreibung

Dateiheader fuer .bmp.

5.2.2 Dokumentation der Datenelemente

5.2.2.1 bfFileSize

```
uint32_t bitmap::BitmapFileHeader::bfFileSize
```

Groesse der .bmp Datei in bytes.

5.2.2.2 bfRes1

```
uint16_t bitmap::BitmapFileHeader::bfRes1
```

Muss 0 sein.

5.2.2.3 bfRes2

```
uint16_t bitmap::BitmapFileHeader::bfRes2
```

Muss 0 sein.

5.2.2.4 bfStartAdress

```
uint32_t bitmap::BitmapFileHeader::bfStartAdress
```

Offset zu den Bilddaten von Beginn der Datei.

5.2.2.5 bfType

```
uint16_t bitmap::BitmapFileHeader::bfType
```

ASCII Zeichen "BM" notwendig.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Source/Image.h](#)

5.3 bitmap::BitmapInfoHeader Strukturreferenz

Bitmapheader. Enthaelte Informationen ueber die Eigenschaften des Bildes.

```
#include <Image.h>
```

Öffentliche Attribute

- `uint32_t biSize`
Groesse dieses Headers in Byte.
- `uint32_t biWidth`
Breite des Bildes in [Pixel](#).
- `uint32_t biHeight`
Hoehe des Bildes in [Pixel](#).
- `uint16_t biColorPlanes`
In .bmp nicht verwendet.
- `uint16_t biBitsPerPixel`
Farbtiefe in bpp (8)
- `uint32_t biCompression`
Nicht beachtet.
- `uint32_t biImageSize`
Groesse der Bilddaten.
- `uint32_t biHorizontalResolution`
Nicht beachtet.
- `uint32_t biVerticalResolution`
Nicht beachtet.
- `uint32_t biColorsPerColorPalette`
Nicht beachtet.
- `uint32_t biImportantColorsUsed`
Nicht beachtet.

5.3.1 Ausführliche Beschreibung

Bitmapheader. Enthael Informationen ueber die Eigenschaften des Bildes.

5.3.2 Dokumentation der Datenelemente

5.3.2.1 biBitsPerPixel

```
uint16_t bitmap::BitmapInfoHeader::biBitsPerPixel
```

Farbtiefe in bpp (8)

5.3.2.2 biColorPlanes

```
uint16_t bitmap::BitmapInfoHeader::biColorPlanes
```

In .bmp nicht verwendet.

5.3.2.3 biColorsPerColorPalette

```
uint32_t bitmap::BitmapInfoHeader::biColorsPerColorPalette
```

Nicht beachtet.

5.3.2.4 biCompression

```
uint32_t bitmap::BitmapInfoHeader::biCompression
```

Nicht beachtet.

5.3.2.5 biHeight

```
uint32_t bitmap::BitmapInfoHeader::biHeight
```

Hoehe des Bildes in [Pixel](#).

5.3.2.6 biHorizontalResolution

```
uint32_t bitmap::BitmapInfoHeader::biHorizontalResolution
```

Nicht beachtet.

5.3.2.7 biImageSize

```
uint32_t bitmap::BitmapInfoHeader::biImageSize
```

Groesse der Bilddaten.

5.3.2.8 biImportantColorsUsed

```
uint32_t bitmap::BitmapInfoHeader::biImportantColorsUsed
```

Nicht beachtet.

5.3.2.9 biSize

```
uint32_t bitmap::BitmapInfoHeader::biSize
```

Groesse dieses Headers in Byte.

5.3.2.10 biVerticalResolution

```
uint32_t bitmap::BitmapInfoHeader::biVerticalResolution
```

Nicht beachtet.

5.3.2.11 biWidth

```
uint32_t bitmap::BitmapInfoHeader::biWidth
```

Breite des Bildes in [Pixel](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Source/Image.h](#)

5.4 Image Klassenreferenz

Klasse zum oeffnen und auslesen von Bitmap Bilddateien.

```
#include <Image.h>
```

Öffentliche Methoden

- [Image](#) ()=delete
Standardkonstruktor.
- [Image](#) ([Image](#) &&other)=default
Standard Movekonstruktor.
- [Image](#) (std::string fileName)
Konstruktor. Oeffnet .bmp Datei mit dem angegebenen Namen.
- [Image](#) & operator= (const [Image](#) &other)=delete
Zuweisungen fuer [Image](#) nicht verfuegbar.
- [Image](#) & operator= ([Image](#) &&other)=delete
Zuweisungen fuer [Image](#) nicht verfuegbar.
- const [Pixel](#) & getPixel (int y, int x) const
[Pixel](#) an der Stelle (y,x) auslesen.
- uint32_t getHeight () const
Hoehe des Bildes erhalten.
- uint32_t getWidth () const
Breite des Bildes erhalten.
- ~[Image](#) ()

Private Methoden

- void `readHeader` ()
Header der .bmp Datei auslesen, um Verwaltungsinformationen wie z.B. Hoehe und Breite zu erhalten.
- void `readImageData` ()
Bilddaten aus der .bmp Datei auslesen und in diesem Objekt speichern.

Private Attribute

- `bitmap::BitmapFileHeader` `bitmapFileHeader`
Informationen ueber die Datei (Datentyp,Bytes,Startadresse der Bilddaten..)
- `bitmap::BitmapInfoHeader` `bitmapInfoHeader`
Informationen ueber das Bild (Breite, Hoehe, Farben..)
- `std::vector< std::vector< Pixel > >` `bitmapData`
2D Vector mit den Pixeldaten des Bildes
- `std::ifstream` `file`
Inputfilestream fuer die zu oeffnende Bilddatei.

5.4.1 Ausführliche Beschreibung

Klasse zum oeffnen und auslesen von Bitmap Bilddateien.

5.4.2 Beschreibung der Konstruktoren und Destruktoren

5.4.2.1 `Image()` [1/3]

```
Image::Image ( ) [delete]
```

Standardkonstruktor.

5.4.2.2 `Image()` [2/3]

```
Image::Image (
    Image && other ) [default]
```

Standard Movekonstruktor.

Parameter

<i>other</i>	<code>Image</code> rvalue
--------------	---------------------------

5.4.2.3 Image() [3/3]

```
Image::Image (
    std::string fileName )
```

Konstruktor. Oeffnet .bmp Datei mit dem angegebenen Namen.

Parameter

<i>fileName</i>	Name der .bmp Datei als String
-----------------	--------------------------------

5.4.2.4 ~Image()

```
Image::~Image ( )
```

5.4.3 Dokumentation der Elementfunktionen

5.4.3.1 getHeight()

```
uint32_t Image::getHeight ( ) const [inline]
```

Hoehe des Bildes erhalten.

Rückgabe

Hoehe als uint32_t

5.4.3.2 getPixel()

```
const Pixel& Image::getPixel (
    int y,
    int x ) const [inline]
```

Pixel an der Stelle (y,x) auslesen.

Parameter

<i>y</i>	Koordinate Y
<i>x</i>	Koordinate X

Rückgabe

Referenz auf ein konstanten [Pixel](#) der sich an Stelle (y,x) befindet.

5.4.3.3 getWidth()

```
uint32_t Image::getWidth ( ) const [inline]
```

Breite des Bildes erhalten.

Rückgabe

Breite als uint32_t

5.4.3.4 operator=() [1/2]

```
Image& Image::operator= (
    const Image & other ) [delete]
```

Zuweisungen fuer [Image](#) nicht verfuegbar.

5.4.3.5 operator=() [2/2]

```
Image& Image::operator= (
    Image && other ) [delete]
```

Zuweisungen fuer [Image](#) nicht verfuegbar.

5.4.3.6 readHeader()

```
void Image::readHeader ( ) [private]
```

Header der .bmp Datei auslesen, um Verwaltungsinformationen wie z.B. Hoehe und Breite zu erhalten.

5.4.3.7 readImageData()

```
void Image::readImageData ( ) [private]
```

Bilddaten aus der .bmp Datei auslesen und in diesem Objekt speichern.

5.4.4 Dokumentation der Datenelemente

5.4.4.1 bitmapData

```
std::vector<std::vector<Pixel> > Image::bitmapData [private]
```

2D Vector mit den Pixeldaten des Bildes

5.4.4.2 bitmapFileHeader

```
bitmap::BitmapFileHeader Image::bitmapFileHeader [private]
```

Informationen ueber die Datei (Datentyp,Bytes,Startadresse der Bilddaten..)

5.4.4.3 bitmapInfoHeader

```
bitmap::BitmapInfoHeader Image::bitmapInfoHeader [private]
```

Informationen ueber das Bild (Breite, Hoehe, Farben..)

5.4.4.4 file

```
std::ifstream Image::file [private]
```

Inputfilestream fuer die zu oeffnende Bilddatei.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [Source/Image.h](#)
- [Source/Image.cpp](#)

5.5 Pixel Strukturreferenz

[Pixel](#) 8 Bit RGB Werte.

```
#include <Image.h>
```

Öffentliche Attribute

- `uint8_t r`
Rotanteil.
- `uint8_t g`
Gruenanteil.
- `uint8_t b`
Blauanteil.

5.5.1 Ausführliche Beschreibung

`Pixel` 8 Bit RGB Werte.

5.5.2 Dokumentation der Datenelemente

5.5.2.1 `b`

```
uint8_t Pixel::b
```

Blauanteil.

5.5.2.2 `g`

```
uint8_t Pixel::g
```

Gruenanteil.

5.5.2.3 `r`

```
uint8_t Pixel::r
```

Rotanteil.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Source/Image.h](#)

5.6 Processing::Processing Klassenreferenz

Klasse zur Umwandlung von Bilddaten in Tondaten.

```
#include <Processing.h>
```

Öffentliche Methoden

- void **start** (const std::string &imageFile, const std::string &outputFile, **ProcessingAlgorithm** algorithm, const **ProcessingSettings** &settings, const **AdditionalSettings** &additionalSettings)
Starte die Umwandlung eines Bildes in eine Audiodatei, anhand eines vorgegebenen Algorithmus.
- **Processing** ()=default
Standardkonstruktor.
- **Processing** (const **Processing** &other)=delete
Kopieren nicht moeglich.
- **Processing** & **operator=** (const **Processing** &other)=delete
Zuweisung nicht moeglich.
- **Processing** & **operator=** (**Processing** &&other)=delete
Movezuweisung nicht moeglich.

Private Methoden

- void **lrScan** (const **AdditionalSettings** &additionalSettings)
*Bild wird von Links nach Rechts spaltenweise durchgescannt. Jede Zeile steht fuer eine Frequenz. Je intensiver RGB vom **Pixel** desto hoeher die Amplitude der jeweiligen Frequenz. **Pixel** werden nur betrachtet, wenn der Durschnitt von RGB > Aktivierungsschwelle ist.*
- void **lrScan_no_threshold** (const **AdditionalSettings** &additionalSettings)
*Bild wird von Links nach Rechts spaltenweise durchgescannt. Jede Zeile steht fuer eine Frequenz. Je intensiver RGB vom **Pixel** desto hoeher die Amplitude der jeweiligen Frequenz. Keine Aktivierungsgrenze.*
- void **udScan** (const **AdditionalSettings** &additionalSettings)
*Bild wird von Oben nach Unten zweilenweise durchgescannt. Jede Spalte steht fuer eine Frequenz. Je intensiver R↔GB vom **Pixel** desto hoeher die Amplitude der jeweiligen Frequenz. **Pixel** werden nur betrachtet, wenn der Durschnitt von RGB > Aktivierungsschwelle ist.*
- void **udScan_no_threshold** (const **AdditionalSettings** &additionalSettings)
*Bild wird von Oben nach Unten zweilenweise durchgescannt. Jede Spalte steht fuer eine Frequenz. Je intensiver RGB vom **Pixel** desto hoeher die Amplitude der jeweiligen Frequenz. Keine Aktivierungsschwelle.*
- void **triplet** (const **AdditionalSettings** &additionalSettings)
Bild wird immer um 3 Byte (Triplet) oder Vielfache durchgesprungen. Jede Farbe hat eine Eigenschaft bei der Tonerzeugung Rot: Lautstaerke ; Gruen: Tonhoehe; Blau: Tondauer.
- void **triplet_jump** (const **AdditionalSettings** &additionalSettings)
Bild wird mit einem Offset abhaengig von der Farbe durchgesprungen Rot: Lautstarke, Gruen: Tonhoehe; Blau: keine Funktion.
- void **ud_lr_scan** (const **AdditionalSettings** &additionalSettings)
*Das Bild wird von oben nach unten und von links nach rechts gleichzeitig gescannt. Die Senkrechte zur X-Achse und die Parallele erzeugen beide jeweils einen Ton.
Jeder Ton wird abhaengig von jeweiligen Farbanteil erzeugt.
Rot bestimmt die Frequenz, Gruen bestimmt die Lautstaerke und der gemeinsame Blauanteil die Dauer der beiden Frequenzen.*

Private Attribute

- `std::unique_ptr< Image > image`
- `std::unique_ptr< Sound > sound`
- `std::unique_ptr< Wave > wave`

5.6.1 Ausführliche Beschreibung

Klasse zur Umwandlung von Bilddaten in Tondaten.

5.6.2 Beschreibung der Konstruktoren und Destruktoren

5.6.2.1 Processing() [1/2]

```
Processing::Processing::Processing ( ) [default]
```

Standardkonstruktor.

5.6.2.2 Processing() [2/2]

```
Processing::Processing::Processing (
    const Processing & other ) [delete]
```

Kopieren nicht moeglich.

5.6.3 Dokumentation der Elementfunktionen

5.6.3.1 lrScan()

```
void Processing::Processing::lrScan (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird von Links nach Rechts spaltenweise durchgescannt. Jede Zeile steht fuer eine Frequenz. Je intensiver R↔GB vom [Pixel](#) desto hoeher die Amplitude der jeweiligen Frequenz. [Pixel](#) werden nur betrachtet, wenn der Durschnitt von RGB > Aktivierungsschwelle ist.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.2 lrScan_no_threshold()

```
void Processing::Processing::lrScan_no_threshold (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird von Links nach Rechts spaltenweise durchgescannt. Jede Zeile steht fuer eine Frequenz. Je intensiver RGB vom Pixel desto hoeher die Amplitude der jeweiligen Frequenz. Keine Aktivierungsgrenze.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.3 operator=() [1/2]

```
Processing& Processing::Processing::operator= (
    const Processing & other ) [delete]
```

Zuweisung nicht moeglich.

5.6.3.4 operator=() [2/2]

```
Processing& Processing::Processing::operator= (
    Processing && other ) [delete]
```

Movezuweisung nicht moeglich.

5.6.3.5 start()

```
void Processing::Processing::start (
    const std::string & imageFile,
    const std::string & outputFile,
    ProcessingAlgorithm algorithm,
    const ProcessingSettings & settings,
    const AdditionalSettings & additionalSettings )
```

Starte die Umwandlung eines Bildes in eine Audiodatei, anhand eines vorgegebenen Algorithmus.

Parameter

<i>imageFile</i>	Dateiname vom Bild
<i>outputFile</i>	Dateiname der zu erstellenden Audiodatei
<i>algorithm</i>	Algorithmus der fuer die Umwandlung genutzt wird
<i>settings</i>	Allgemeine Einstellungen fuer die Tonerzeugung
<i>additionalSettings</i>	Spezifische Einstellungen fuer den jeweiligen Algorithmus

5.6.3.6 triplet()

```
void Processing::Processing::triplet (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird immer um 3 Byte (Triplet) oder Vielfache durchgesprungen. Jede Farbe hat eine Eigenschaft bei der Tonerzeugung Rot: Lautstaerke ; Gruen: Tonhoehe; Blau: Tondauer.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.7 triplet_jump()

```
void Processing::Processing::triplet_jump (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird mit einem Offset abhaengig von der Farbe durchgesprungen Rot: Lautstarke, Gruen: Tonhoehe; Blau: keine Funktion.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.8 ud_lr_scan()

```
void Processing::Processing::ud_lr_scan (
    const AdditionalSettings & additionalSettings ) [private]
```

Das Bild wird von oben nach unten und von links nach rechts gleichzeitig gescannt. Die Senkrechte zur X-Achse und die Parallele erzeugen beide jeweils einen Ton.

Jeder Ton wird abhaengig von jeweiligen Farbanteil erzeugt.

Rot bestimmt die Frequenz, Gruen bestimmt die Lautstaerke und der gemeinsame Blauanteil die Dauer der beiden Frequenzen.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.9 udScan()

```
void Processing::Processing::udScan (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird von Oben nach Unten zeilenweise durchgescannt. Jede Spalte steht fuer eine Frequenz. Je intensiver RGB vom Pixel desto hoeher die Amplitude der jeweiligen Frequenz. Pixel werden nur betrachtet, wenn der Durchschnitt von RGB > Aktivierungsschwelle ist.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.3.10 udScan_no_threshold()

```
void Processing::Processing::udScan_no_threshold (
    const AdditionalSettings & additionalSettings ) [private]
```

Bild wird von Oben nach Unten zeilenweise durchgescannt. Jede Spalte steht fuer eine Frequenz. Je intensiver RGB vom Pixel desto hoeher die Amplitude der jeweiligen Frequenz. Keine Aktivierungsschwelle.

Parameter

<i>additionalSettings</i>	
---------------------------	--

5.6.4 Dokumentation der Datenelemente

5.6.4.1 image

```
std::unique_ptr<Image> Processing::Processing::image [private]
```

5.6.4.2 sound

```
std::unique_ptr<Sound> Processing::Processing::sound [private]
```

5.6.4.3 wave

```
std::unique_ptr<Wave> Processing::Processing::wave [private]
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Source/[Processing.h](#)
- Source/[Processing.cpp](#)

5.7 Processing::ProcessingSettings Strukturreferenz

Grundeinstellungen fuer die Tonerzeugung.

```
#include <Processing.h>
```

Öffentliche Attribute

- double [audioVolume](#) = 50
Grundlautstaerke.
- double [samplingFrequency](#) = 44100
Abtastrate.
- size_t [samples](#) = 128
Anzahl der Samples pro Frequenzgenerierung, erhoeht die Tondauer&Rechenaufwand.

5.7.1 Ausführliche Beschreibung

Grundeinstellungen fuer die Tonerzeugung.

5.7.2 Dokumentation der Datenelemente

5.7.2.1 audioVolume

```
double Processing::ProcessingSettings::audioVolume = 50
```

Grundlautstaerke.

5.7.2.2 samples

```
size_t Processing::ProcessingSettings::samples = 128
```

Anzahl der Samples pro Frequenzgenerierung, erhoeht die Tondauer&Rechenaufwand.

5.7.2.3 samplingFrequency

```
double Processing::ProcessingSettings::samplingFrequency = 44100
```

Abtastrate.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Source/[Processing.h](#)

5.8 Sound Klassenreferenz

Klasse zur Erzeugung von Toenen.

```
#include <Sound.h>
```

Öffentliche Methoden

- [Sound](#) ()
Standardkonstruktor.
- [Sound](#) (double [samplingFrequency](#), double [audioVolume](#), size_t [bufferSize](#))
Konstruktor.
- void [addFrequency](#) (double volume, double frequency)
Einen Ton mit einer bestimmter Frequenz&Amplitude aufaddieren.
- const std::vector< double > & [getBuffer](#) () const
Den zuvor beschriebenen Puffer auslesen.
- void [resetBuffer](#) ()
Puffer loeschen/zuruecksetzen.
- [~Sound](#) ()=default

Private Attribute

- std::vector< double > [buffer](#)
Puffer fuer die Audiodaten.
- const double [samplingFrequency](#) { DEFAULT_SAMPLINGRATE }
Abtastfrequenz welche zur Tonerzeugung genutzt wird.
- const double [audioVolume](#) { DEFAULT_AUDIOVOLUME }
Grundlautstaerke.
- double [lastPos](#) { 0 }
Letzte Position der Sinuswelle bei verlassen der Funktion addFrequency.
- const size_t [bufferSize](#) { DEFAULT_BUFFERSIZE }
Groesse des Puffers.

5.8.1 Ausführliche Beschreibung

Klasse zur Erzeugung von Toenen.

5.8.2 Beschreibung der Konstruktoren und Destruktoren

5.8.2.1 Sound() [1/2]

```
Sound::Sound ( )
```

Standardkonstruktor.

5.8.2.2 Sound() [2/2]

```
Sound::Sound (
    double samplingFrequency,
    double audioVolume,
    size_t bufferSize )
```

Konstruktor.

Parameter

<i>samplingFrequency</i>	Abtaste rate fuer Tonerzeugung
<i>audioVolume</i>	Grundlautstaerke
<i>bufferSize</i>	Puffergroesse. Je groesser desto groesser/laenger ein Sample und Rechenaufwand

5.8.2.3 ~Sound()

```
Sound::~Sound ( ) [default]
```

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 addFrequency()

```
void Sound::addFrequency (
    double volume,
    double frequency )
```

Einen Ton mit einer bestimmter Frequenz&Amplitude aufaddieren.

Parameter

<i>volume</i>	Amplitude (Lautstärke) der Frequenz
<i>frequency</i>	Frequenz in Hz

5.8.3.2 getBuffer()

```
const std::vector< double > & Sound::getBuffer ( ) const
```

Den zuvor beschriebenen Puffer auslesen.

Rückgabe

Konstante Referenz auf den Puffer

5.8.3.3 resetBuffer()

```
void Sound::resetBuffer ( )
```

Puffer löschen/zurücksetzen.

5.8.4 Dokumentation der Datenelemente**5.8.4.1 audioVolume**

```
const double Sound::audioVolume { DEFAULT_AUDIOVOLUME } [private]
```

Grundlautstärke.

5.8.4.2 buffer

```
std::vector<double> Sound::buffer [private]
```

Puffer für die Audiodaten.

5.8.4.3 bufferSize

```
const size_t Sound::bufferSize { DEFAULT_BUFFERSIZE } [private]
```

Groesse des Puffers.

5.8.4.4 lastPos

```
double Sound::lastPos { 0 } [private]
```

Letzte Position der Sinuswelle bei verlassen der Funktion addFrequency.

5.8.4.5 samplingFrequency

```
const double Sound::samplingFrequency { DEFAULT_SAMPLINGRATE } [private]
```

Abtastfrequenz welche zur Tonerzeugung genutzt wird.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Source/[Sound.h](#)
- Source/[Sound.cpp](#)

5.9 Wave Klassenreferenz

Klasse zur Erzeugung von .wav Audiodateien.

```
#include <Wave.h>
```

Öffentliche Methoden

- [Wave](#) (std::string fileName, uint32_t [sampleRate](#))
Konstruktor.
- [Wave](#) ()
Standardkonstruktor.
- template<typename T >
void [writeSamples](#) (T data)
Schreibt Daten in Audiodatei.
- void [finishHeader](#) ()
Header fertig machen. Erst nutzen wenn Audiodateien fertig beschrieben wurde.
- void [closeFile](#) ()
Audiodatei schliessen.
- [~Wave](#) ()

Private Methoden

- void `prepareHeader()`
Bereitet Header für Audiodatei vor.

Private Attribute

- `std::ofstream file {"WieKlingtEinBild.wav", std::ios::binary}`
Outputfilestream. Genutzt um Audiodaten in Datei zu schreiben.
- `struct {`
 - `const uint32_t subChunk1Size {16}`
Keine zusätzlichen Daten.
 - `const uint16_t audioFormat {1}`
Keine Kompression.
 - `const uint16_t numChannels {1}`
Anzahl der Channel (hier: 1)
 - `uint32_t sampleRate {44100}`
Abtastrate.
 - `const uint32_t byteRate {static_cast<uint32_t>(sampleRate * numChannels * DEFAULT_BYTES_PER_SAMPLE)}`
*Abtastrate * Frame Grösse.*
 - `const uint16_t blockAlign {static_cast<uint16_t>(numChannels * DEFAULT_BYTES_PER_SAMPLE)}`
*Alignment. numChannels * DEFAULT_BYTES_PER_SAMPLE.*
 - `const uint16_t bitsPerSample {DEFAULT_BITS_PER_SAMPLE}`
Bits pro Samplewert pro Kanal.
- `} waveHeader`
Header der .wav Datei.

5.9.1 Ausführliche Beschreibung

Klasse zur Erzeugung von .wav Audiodateien.

5.9.2 Beschreibung der Konstruktoren und Destruktoren

5.9.2.1 Wave() [1/2]

```
Wave::Wave (
    std::string fileName,
    uint32_t sampleRate )
```

Konstruktor.

Parameter

<i>fileName</i>	Dateiname der zu erzeugenden Audiodatei (.wav)
<i>sampleRate</i>	Abtastrate der Audiodatei

5.9.2.2 Wave() [2/2]

```
Wave::Wave ( )
```

Standardkonstruktor.

5.9.2.3 ~Wave()

```
Wave::~~Wave ( )
```

5.9.3 Dokumentation der Elementfunktionen

5.9.3.1 closeFile()

```
void Wave::closeFile ( )
```

Audiodatei schliessen.

5.9.3.2 finishHeader()

```
void Wave::finishHeader ( )
```

Header fertig machen. Erst nutzen wenn Audiodateien fertig beschrieben wurde.

5.9.3.3 prepareHeader()

```
void Wave::prepareHeader ( ) [private]
```

Bereitet Header für Audiodatei vor.

5.9.3.4 writeSamples()

```
template<typename T >  
void Wave::writeSamples (   
    T data ) [inline]
```

Schreibt Daten in Audiodatei.

Template Parameters

<i>T</i>	Typ von Daten z.B. uint16_t
----------	--------------------------------

Parameter

<i>data</i>	Daten die geschrieben werden sollen
-------------	-------------------------------------

5.9.4 Dokumentation der Datenelemente

5.9.4.1 audioFormat

```
const uint16_t Wave::audioFormat {1}
```

Keine Kompression.

5.9.4.2 bitsPerSample

```
const uint16_t Wave::bitsPerSample {DEFAULT_BITS_PER_SAMPLE}
```

Bits pro Samplewert pro Kanal.

5.9.4.3 blockAlign

```
const uint16_t Wave::blockAlign {static_cast<uint16_t>(numChannels * DEFAULT_BYTES_PER_SAMPLE↵LE) }
```

Alignment. numChannels * DEFAULT_BYTES_PER_SAMPLE.

5.9.4.4 byteRate

```
const uint32_t Wave::byteRate {static_cast<uint32_t>(sampleRate * numChannels * DEFAULT_BYTE↵S_PER_SAMPLE) }
```

Abtastrate* Frame Groesse.

5.9.4.5 file

```
std::ofstream Wave::file {"WieKlingtEinBild.wav", std::ios::binary} [private]
```

Outputfilestream. Genutzt um Audiodaten in Datei zu schreiben.

5.9.4.6 numChannels

```
const uint16_t Wave::numChannels {1}
```

Anzahl der Channel (hier: 1)

5.9.4.7 sampleRate

```
uint32_t Wave::sampleRate {44100}
```

Abtastrate.

5.9.4.8 subChunk1Size

```
const uint32_t Wave::subChunk1Size {16}
```

Keine zusaetzlichen Daten.

5.9.4.9 waveHeader

```
struct { ... } Wave::waveHeader [private]
```

Header der .wav Datei.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [Source/Wave.h](#)
- [Source/Wave.cpp](#)

Kapitel 6

Datei-Dokumentation

6.1 Source/Image.cpp-Dateireferenz

```
#include "Image.h"  
#include <iostream>
```

6.2 Source/Image.h-Dateireferenz

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

Klassen

- struct `bitmap::BitmapFileHeader`
Dateiheader fuer .bmp.
- struct `bitmap::BitmapInfoHeader`
Bitmapheader. Enthael Informationen ueber die Eigenschaften des Bildes.
- struct `Pixel`
Pixel 8 Bit RGB Werte.
- class `Image`
Klasse zum oeffnen und auslesen von Bitmap Bilddateien.

Namensbereiche

- `bitmap`
Verwaltungsinformationen zu Bitmap Dateien.

6.3 Source/Processing.cpp-Dateireferenz

```
#include "Processing.h"  
#include <iostream>
```

Makrodefinitionen

- `#define PRINT_PROGRESS`

Funktionen

- `void printProgress (int x, int xMax)`

6.3.1 Makro-Dokumentation

6.3.1.1 PRINT_PROGRESS

```
#define PRINT_PROGRESS
```

6.3.2 Dokumentation der Funktionen

6.3.2.1 printProgress()

```
void printProgress (  
    int x,  
    int xMax )
```

6.4 Source/Processing.h-Dateireferenz

```
#include "Image.h"  
#include "Sound.h"  
#include "Wave.h"  
#include <memory>
```

Klassen

- struct [Processing::ProcessingSettings](#)
Grundeinstellungen fuer die Tonerzeugung.
- struct [Processing::AdditionalSettings](#)
Zusaetzliche Einstellungen spezifisch fuer die verschiedenen Algorithmen.
- class [Processing::Processing](#)
Klasse zur Umwandlung von Bilddaten in Tondaten.

Namensbereiche

- [Processing](#)
Beeinhaltet Datenstrukturen zur Umwandlung von Bildinformationen in Toene.

Aufzählungen

- enum [Processing::ProcessingAlgorithm](#) {
[Processing::ProcessingAlgorithm::LR_SCAN](#) = 0, [Processing::ProcessingAlgorithm::LR_SCAN_NO_THRESHOLD](#),
[Processing::ProcessingAlgorithm::UD_SCAN](#), [Processing::ProcessingAlgorithm::UD_SCAN_NO_THRESHOLD](#),
[Processing::ProcessingAlgorithm::TRIPLET](#), [Processing::ProcessingAlgorithm::TRIPLET_JMP](#), [Processing::ProcessingAlgorithm::TRIPLET_JMP_NO_THRESHOLD](#),
 }
Verfuegbare Algorithmen.

6.5 Source/Quelle.cpp-Dateireferenz

```
#include <iostream>
#include "Processing.h"
```

Funktionen

- int [main](#) (int argc, char *argv[])

6.5.1 Dokumentation der Funktionen

6.5.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

6.6 Source/Sound.cpp-Dateireferenz

```
#include "Sound.h"  
#include <cmath>
```

Makrodefinitionen

- `#define _USE_MATH_DEFINES`

6.6.1 Makro-Dokumentation

6.6.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

6.7 Source/Sound.h-Dateireferenz

```
#include <vector>
```

Klassen

- class [Sound](#)
Klasse zur Erzeugung von Toenen.

6.8 Source/Wave.cpp-Dateireferenz

```
#include "Wave.h"
```

6.9 Source/Wave.h-Dateireferenz

```
#include <fstream>
```

Klassen

- class [Wave](#)
Klasse zur Erzeugung von .wav Audiodateien.

Index

- `_USE_MATH_DEFINES`
 - Sound.cpp, 38
 - `~Image`
 - Image, 17
 - `~Sound`
 - Sound, 28
 - `~Wave`
 - Wave, 32
 - `addFrequency`
 - Sound, 28
 - `audioFormat`
 - Wave, 33
 - `audioVolume`
 - Processing::ProcessingSettings, 26
 - Sound, 29
 - `b`
 - Pixel, 20
 - `bfFileSize`
 - bitmap::BitmapFileHeader, 11
 - `bfRes1`
 - bitmap::BitmapFileHeader, 12
 - `bfRes2`
 - bitmap::BitmapFileHeader, 12
 - `bfStartAdress`
 - bitmap::BitmapFileHeader, 12
 - `bfType`
 - bitmap::BitmapFileHeader, 12
 - `biBitsPerPixel`
 - bitmap::BitmapInfoHeader, 13
 - `biColorPlanes`
 - bitmap::BitmapInfoHeader, 13
 - `biColorsPerColorPalette`
 - bitmap::BitmapInfoHeader, 13
 - `biCompression`
 - bitmap::BitmapInfoHeader, 14
 - `biHeight`
 - bitmap::BitmapInfoHeader, 14
 - `biHorizontalResolution`
 - bitmap::BitmapInfoHeader, 14
 - `biImageSize`
 - bitmap::BitmapInfoHeader, 14
 - `biImportantColorsUsed`
 - bitmap::BitmapInfoHeader, 14
 - `biSize`
 - bitmap::BitmapInfoHeader, 14
 - `biVerticalResolution`
 - bitmap::BitmapInfoHeader, 15
 - `biWidth`
 - bitmap::BitmapInfoHeader, 15
- `bitmap`, 7
 - `bitmap::BitmapFileHeader`, 11
 - `bfFileSize`, 11
 - `bfRes1`, 12
 - `bfRes2`, 12
 - `bfStartAdress`, 12
 - `bfType`, 12
 - `bitmap::BitmapInfoHeader`, 12
 - `biBitsPerPixel`, 13
 - `biColorPlanes`, 13
 - `biColorsPerColorPalette`, 13
 - `biCompression`, 14
 - `biHeight`, 14
 - `biHorizontalResolution`, 14
 - `biImageSize`, 14
 - `biImportantColorsUsed`, 14
 - `biSize`, 14
 - `biVerticalResolution`, 15
 - `biWidth`, 15
 - `bitmapData`
 - Image, 19
 - `bitmapFileHeader`
 - Image, 19
 - `bitmapInfoHeader`
 - Image, 19
 - `bitsPerSample`
 - Wave, 33
 - `blockAlign`
 - Wave, 33
 - `buffer`
 - Sound, 29
 - `bufferSize`
 - Sound, 29
 - `byteRate`
 - Wave, 33
 - `closeFile`
 - Wave, 32
 - `file`
 - Image, 19
 - Wave, 33
 - `finishHeader`
 - Wave, 32
 - `g`
 - Pixel, 20
 - `getBuffer`
 - Sound, 29

- getHeight
 - Image, [17](#)
- getPixel
 - Image, [17](#)
- getWidth
 - Image, [18](#)
- Image, [15](#)
 - ~Image, [17](#)
 - bitmapData, [19](#)
 - bitmapFileHeader, [19](#)
 - bitmapInfoHeader, [19](#)
 - file, [19](#)
 - getHeight, [17](#)
 - getPixel, [17](#)
 - getWidth, [18](#)
 - Image, [16](#)
 - operator=, [18](#)
 - readHeader, [18](#)
 - readImageData, [18](#)
- image
 - Processing::Processing, [25](#)
- lastPos
 - Sound, [30](#)
- IrScan
 - Processing::Processing, [22](#)
- IrScan_no_threshold
 - Processing::Processing, [23](#)
- main
 - Quelle.cpp, [37](#)
- numChannels
 - Wave, [34](#)
- operator=
 - Image, [18](#)
 - Processing::Processing, [23](#)
- PRINT_PROGRESS
 - Processing.cpp, [36](#)
- Pixel, [19](#)
 - b, [20](#)
 - g, [20](#)
 - r, [20](#)
- prepareHeader
 - Wave, [32](#)
- printProgress
 - Processing.cpp, [36](#)
- Processing, [7](#)
 - Processing::Processing, [22](#)
 - ProcessingAlgorithm, [8](#)
- Processing.cpp
 - PRINT_PROGRESS, [36](#)
 - printProgress, [36](#)
- Processing::AdditionalSettings, [9](#)
 - settings1, [10](#)
 - settings2, [10](#)
 - settings3, [10](#)
 - settings4, [10](#)
- Processing::Processing, [21](#)
 - image, [25](#)
 - IrScan, [22](#)
 - IrScan_no_threshold, [23](#)
 - operator=, [23](#)
 - Processing, [22](#)
 - sound, [25](#)
 - start, [23](#)
 - triplet, [24](#)
 - triplet_jump, [24](#)
 - ud_Ir_scan, [24](#)
 - udScan, [24](#)
 - udScan_no_threshold, [25](#)
 - wave, [25](#)
- Processing::ProcessingSettings, [26](#)
 - audioVolume, [26](#)
 - samples, [26](#)
 - samplingFrequency, [26](#)
- ProcessingAlgorithm
 - Processing, [8](#)
- Quelle.cpp
 - main, [37](#)
- r
 - Pixel, [20](#)
- readHeader
 - Image, [18](#)
- readImageData
 - Image, [18](#)
- resetBuffer
 - Sound, [29](#)
- sampleRate
 - Wave, [34](#)
- samples
 - Processing::ProcessingSettings, [26](#)
- samplingFrequency
 - Processing::ProcessingSettings, [26](#)
 - Sound, [30](#)
- settings1
 - Processing::AdditionalSettings, [10](#)
- settings2
 - Processing::AdditionalSettings, [10](#)
- settings3
 - Processing::AdditionalSettings, [10](#)
- settings4
 - Processing::AdditionalSettings, [10](#)
- Sound, [27](#)
 - ~Sound, [28](#)
 - addFrequency, [28](#)
 - audioVolume, [29](#)
 - buffer, [29](#)
 - bufferSize, [29](#)
 - getBuffer, [29](#)
 - lastPos, [30](#)
 - resetBuffer, [29](#)
 - samplingFrequency, [30](#)

- Sound, [28](#)
- sound
 - Processing::Processing, [25](#)
- Sound.cpp
 - _USE_MATH_DEFINES, [38](#)
- Source/Image.cpp, [35](#)
- Source/Image.h, [35](#)
- Source/Processing.cpp, [36](#)
- Source/Processing.h, [36](#)
- Source/Quelle.cpp, [37](#)
- Source/Sound.cpp, [38](#)
- Source/Sound.h, [38](#)
- Source/Wave.cpp, [38](#)
- Source/Wave.h, [38](#)
- start
 - Processing::Processing, [23](#)
- subChunk1Size
 - Wave, [34](#)
- triplet
 - Processing::Processing, [24](#)
- triplet_jump
 - Processing::Processing, [24](#)
- ud_lr_scan
 - Processing::Processing, [24](#)
- udScan
 - Processing::Processing, [24](#)
- udScan_no_threshold
 - Processing::Processing, [25](#)
- Wave, [30](#)
 - ~Wave, [32](#)
 - audioFormat, [33](#)
 - bitsPerSample, [33](#)
 - blockAlign, [33](#)
 - byteRate, [33](#)
 - closeFile, [32](#)
 - file, [33](#)
 - finishHeader, [32](#)
 - numChannels, [34](#)
 - prepareHeader, [32](#)
 - sampleRate, [34](#)
 - subChunk1Size, [34](#)
 - Wave, [31](#), [32](#)
 - waveHeader, [34](#)
 - writeSamples, [32](#)
- wave
 - Processing::Processing, [25](#)
- waveHeader
 - Wave, [34](#)
- writeSamples
 - Wave, [32](#)