

Wiesn-Run

Erzeugt von Doxygen 1.8.6

Mon Jun 29 2015 19:35:39

Inhaltsverzeichnis

1	Ausstehende Aufgaben	1
2	Hierarchie-Verzeichnis	3
2.1	Klassenhierarchie	3
3	Klassen-Verzeichnis	5
3.1	Auflistung der Klassen	5
4	Datei-Verzeichnis	7
4.1	Auflistung der Dateien	7
5	Klassen-Dokumentation	9
5.1	Audio Klassenreferenz	9
5.1.1	Ausführliche Beschreibung	9
5.1.2	Beschreibung der Konstruktoren und Destrukturen	10
5.1.2.1	Audio	10
5.1.2.2	~Audio	10
5.1.3	Dokumentation der Elementfunktionen	10
5.1.3.1	getSample	10
5.1.3.2	getSamplenumber	10
5.1.3.3	getSource	11
5.1.3.4	getVolume	11
5.1.3.5	setVolume	11
5.2	AudioControl Klassenreferenz	11
5.2.1	Ausführliche Beschreibung	12
5.2.2	Beschreibung der Konstruktoren und Destrukturen	12
5.2.2.1	AudioControl	12
5.2.2.2	~AudioControl	12
5.2.3	Dokumentation der Elementfunktionen	13
5.2.3.1	playInitialize	13
5.2.3.2	update	13
5.3	audioCooldownstruct Strukturreferenz	15
5.3.1	Ausführliche Beschreibung	15

5.4	audioCooldownStruct Strukturreferenz	15
5.4.1	Ausführliche Beschreibung	16
5.5	audioDistanceStruct Strukturreferenz	16
5.5.1	Ausführliche Beschreibung	17
5.6	audioStruct Strukturreferenz	17
5.6.1	Ausführliche Beschreibung	18
5.7	collisionStruct Strukturreferenz	18
5.7.1	Ausführliche Beschreibung	19
5.8	compareGameObjects Strukturreferenz	19
5.8.1	Ausführliche Beschreibung	19
5.9	compareScores Strukturreferenz	19
5.9.1	Ausführliche Beschreibung	20
5.10	Enemy Klassenreferenz	20
5.10.1	Beschreibung der Konstruktoren und Destruktoren	21
5.10.1.1	Enemy	21
5.10.2	Dokumentation der Elementfunktionen	21
5.10.2.1	getDeath	21
5.10.2.2	getDeathCooldown	21
5.10.2.3	getFireCooldown	21
5.10.2.4	getHealth	21
5.10.2.5	getInflictedDamage	22
5.10.2.6	setDeath	22
5.10.2.7	setHealth	22
5.11	Game Klassenreferenz	22
5.11.1	Ausführliche Beschreibung	23
5.11.2	Beschreibung der Konstruktoren und Destruktoren	23
5.11.2.1	Game	23
5.11.3	Dokumentation der Elementfunktionen	23
5.11.3.1	setState	23
5.11.3.2	start	24
5.11.3.3	step	24
5.11.3.4	timerEvent	25
5.12	GameObject Klassenreferenz	25
5.12.1	Beschreibung der Konstruktoren und Destruktoren	26
5.12.1.1	GameObject	26
5.12.1.2	GameObject	26
5.13	Input Klassenreferenz	26
5.13.1	Ausführliche Beschreibung	27
5.13.2	Beschreibung der Konstruktoren und Destruktoren	27
5.13.2.1	Input	27

5.13.2.2	~Input	28
5.13.3	Dokumentation der Elementfunktionen	28
5.13.3.1	eventFilter	28
5.13.3.2	getAndDeleteLastKey	28
5.13.3.3	getKeyactions	28
5.14	Menu Klassenreferenz	29
5.14.1	Ausführliche Beschreibung	30
5.14.2	Beschreibung der Konstruktoren und Destruktoren	30
5.14.2.1	Menu	30
5.14.3	Dokumentation der Elementfunktionen	30
5.14.3.1	addEntry	30
5.14.3.2	changeSelection	30
5.14.3.3	clear	31
5.14.3.4	displayInit	31
5.14.3.5	displayUpdate	31
5.14.3.6	getEntry	31
5.14.3.7	getSelection	32
5.14.3.8	getTitle	32
5.14.3.9	getType	32
5.15	Menu::menuEntry Strukturreferenz	32
5.15.1	Ausführliche Beschreibung	33
5.16	MovingObject Klassenreferenz	33
5.16.1	Dokumentation der Elementfunktionen	34
5.16.1.1	flipHorizontal	34
5.16.1.2	updatePosition	34
5.17	PaDeviceInfo Strukturreferenz	34
5.17.1	Ausführliche Beschreibung	34
5.17.2	Dokumentation der Datenelemente	34
5.17.2.1	defaultHighInputLatency	34
5.17.2.2	defaultLowInputLatency	35
5.17.2.3	hostApi	35
5.18	PaHostApiInfo Strukturreferenz	35
5.18.1	Ausführliche Beschreibung	35
5.18.2	Dokumentation der Datenelemente	35
5.18.2.1	defaultInputDevice	35
5.18.2.2	defaultOutputDevice	35
5.18.2.3	deviceCount	35
5.18.2.4	name	36
5.18.2.5	structVersion	36
5.18.2.6	type	36

5.19	PaHostErrorInfo Strukturreferenz	36
5.19.1	Ausführliche Beschreibung	36
5.19.2	Dokumentation der Datenelemente	36
5.19.2.1	errorCode	36
5.19.2.2	errorText	36
5.19.2.3	hostApiType	36
5.20	PaStreamCallbackTimeInfo Strukturreferenz	37
5.20.1	Ausführliche Beschreibung	37
5.20.2	Dokumentation der Datenelemente	37
5.20.2.1	currentTime	37
5.20.2.2	inputBufferAdcTime	37
5.20.2.3	outputBufferDacTime	37
5.21	PaStreamInfo Strukturreferenz	37
5.21.1	Ausführliche Beschreibung	38
5.21.2	Dokumentation der Datenelemente	38
5.21.2.1	inputLatency	38
5.21.2.2	outputLatency	38
5.21.2.3	sampleRate	38
5.21.2.4	structVersion	38
5.22	PaStreamParameters Strukturreferenz	38
5.22.1	Ausführliche Beschreibung	39
5.22.2	Dokumentation der Datenelemente	39
5.22.2.1	channelCount	39
5.22.2.2	device	39
5.22.2.3	hostApiSpecificStreamInfo	39
5.22.2.4	sampleFormat	39
5.22.2.5	suggestedLatency	39
5.23	Player Klassenreferenz	40
5.23.1	Beschreibung der Konstruktoren und Destruktoren	41
5.23.1.1	Player	41
5.23.2	Dokumentation der Elementfunktionen	41
5.23.2.1	decreaseAlcoholLevel	41
5.23.2.2	getAlcoholLevel	41
5.23.2.3	getAmmunatiuon	41
5.23.2.4	getFireCooldown	42
5.23.2.5	getHealth	42
5.23.2.6	getImmunityCooldown	42
5.23.2.7	getInflictedDamage	42
5.23.2.8	increaseAlcoholLevel	42
5.23.2.9	inJump	42

5.23.2.10	receiveDamage	43
5.23.2.11	setHealth	43
5.23.2.12	setImmunityCooldown	43
5.23.2.13	update	43
5.24	AudioControl::playStruct Strukturreferenz	43
5.25	PowerUp Klassenreferenz	44
5.25.1	Ausführliche Beschreibung	44
5.25.2	Beschreibung der Konstruktoren und Destruktoren	45
5.25.2.1	PowerUp	45
5.25.2.2	~PowerUp	45
5.25.3	Dokumentation der Elementfunktionen	45
5.25.3.1	getAlcoholLevelBonus	45
5.25.3.2	getAmmunationBonus	45
5.25.3.3	getHealthBonus	45
5.25.3.4	getImmunityCooldownBonus	46
5.25.3.5	getPowerUPType	46
5.26	scoreStruct Strukturreferenz	46
5.26.1	Ausführliche Beschreibung	46
5.27	Shoot Klassenreferenz	47
5.27.1	Beschreibung der Konstruktoren und Destruktoren	47
5.27.1.1	Shoot	47
5.27.2	Dokumentation der Elementfunktionen	48
5.27.2.1	getInflictedDamage	48
5.27.2.2	getOrigin	48
5.28	stateStruct Strukturreferenz	48
5.28.1	Ausführliche Beschreibung	49
6	Datei-Dokumentation	51
6.1	Wiesn-Run/src/portaudio.h-Dateireferenz	51
6.1.1	Ausführliche Beschreibung	53
6.1.2	Makro-Dokumentation	53
6.1.2.1	paClipOff	53
6.1.2.2	paCustomFormat	53
6.1.2.3	paDitherOff	54
6.1.2.4	paFloat32	54
6.1.2.5	paFormatIsSupported	54
6.1.2.6	paFramesPerBufferUnspecified	54
6.1.2.7	paInputOverflow	54
6.1.2.8	paInputUnderflow	54
6.1.2.9	paInt16	54

6.1.2.10	paInt24	55
6.1.2.11	paInt32	55
6.1.2.12	paInt8	55
6.1.2.13	paNeverDropInput	55
6.1.2.14	paNoDevice	55
6.1.2.15	paNoFlag	55
6.1.2.16	paNonInterleaved	55
6.1.2.17	paOutputOverflow	56
6.1.2.18	paOutputUnderflow	56
6.1.2.19	paPlatformSpecificFlags	56
6.1.2.20	paPrimeOutputBuffersUsingStreamCallback	56
6.1.2.21	paPrimingOutput	56
6.1.2.22	paUInt8	56
6.1.2.23	paUseHostApiSpecificDeviceSpecification	56
6.1.3	Dokumentation der benutzerdefinierten Typen	57
6.1.3.1	PaDeviceIndex	57
6.1.3.2	PaDeviceInfo	57
6.1.3.3	PaError	57
6.1.3.4	PaHostApiIndex	57
6.1.3.5	PaHostApiInfo	57
6.1.3.6	PaHostApiTypeId	57
6.1.3.7	PaHostErrorInfo	57
6.1.3.8	PaSampleFormat	58
6.1.3.9	PaStream	58
6.1.3.10	PaStreamCallback	58
6.1.3.11	PaStreamCallbackFlags	59
6.1.3.12	PaStreamCallbackResult	59
6.1.3.13	PaStreamCallbackTimeInfo	60
6.1.3.14	PaStreamFinishedCallback	60
6.1.3.15	PaStreamFlags	60
6.1.3.16	PaStreamInfo	60
6.1.3.17	PaStreamParameters	60
6.1.3.18	PaTime	61
6.1.4	Dokumentation der Aufzählungstypen	61
6.1.4.1	PaHostApiTypeId	61
6.1.4.2	PaStreamCallbackResult	61
6.1.5	Dokumentation der Funktionen	61
6.1.5.1	Pa_AbortStream	61
6.1.5.2	Pa_CloseStream	61
6.1.5.3	Pa_GetDefaultHostApi	61

6.1.5.4	Pa_GetDefaultInputDevice	62
6.1.5.5	Pa_GetDefaultOutputDevice	62
6.1.5.6	Pa_GetDeviceCount	62
6.1.5.7	Pa_GetDeviceInfo	62
6.1.5.8	Pa_GetErrorText	63
6.1.5.9	Pa_GetHostApiCount	63
6.1.5.10	Pa_GetHostApiInfo	63
6.1.5.11	Pa_GetLastHostErrorInfo	63
6.1.5.12	Pa_GetSampleSize	64
6.1.5.13	Pa_GetStreamCpuLoad	64
6.1.5.14	Pa_GetStreamHostApiType	64
6.1.5.15	Pa_GetStreamInfo	64
6.1.5.16	Pa_GetStreamReadAvailable	65
6.1.5.17	Pa_GetStreamTime	65
6.1.5.18	Pa_GetStreamWriteAvailable	65
6.1.5.19	Pa_GetVersion	65
6.1.5.20	Pa_GetVersionText	65
6.1.5.21	Pa_HostApiDeviceIndexToDeviceIndex	65
6.1.5.22	Pa_HostApiTypeIdToHostApiIndex	66
6.1.5.23	Pa_Initialize	66
6.1.5.24	Pa_IsFormatSupported	66
6.1.5.25	Pa_IsStreamActive	67
6.1.5.26	Pa_IsStreamStopped	67
6.1.5.27	Pa_OpenDefaultStream	67
6.1.5.28	Pa_OpenStream	68
6.1.5.29	Pa_ReadStream	69
6.1.5.30	Pa_SetStreamFinishedCallback	69
6.1.5.31	Pa_Sleep	70
6.1.5.32	Pa_StartStream	70
6.1.5.33	Pa_StopStream	70
6.1.5.34	Pa_Terminate	70
6.1.5.35	Pa_WriteStream	70

Kapitel 1

Ausstehende Aufgaben

Element `Enemy::Enemy` (int posX, int posY, int speedX, objectType enemy)

Skalieren der Werte und fireCooldown erhöhen

Element `Game::start` ()

hier wird das Startmenü übersprungen

Element `Game::step` ()

Levelende hier abfragen

Element `Game::timerEvent` (QTimerEvent *event)

return von step...

Element `GameObject::GameObject` (int posX, int posY, objectType type)

Grafik für die Security

Element `Player::decreaseAlcoholLevel` (int decreaseLevel)

Überflüssig, da nie aufgerufen. Auch wenn der Name es nicht vermuten lässt: increaseAlcoholLevel kann den Level auch verringern und wird benutzt.

Klasse `scoreStruct`

Das Konzept der Alkohol-Punkte muss noch ausgearbeitet werden.

Autor

Simon

Klasse `stateStruct`

Diese Struktur ist vermutlich überflüssig.

Autor

Simon

Kapitel 2

Hierarchie-Verzeichnis

2.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

Audio	9
AudioControl	11
audioCooldownstruct	15
audioCooldownStruct	15
audioDistanceStruct	16
audioStruct	17
collisionStruct	18
compareGameObjects	19
compareScores	19
Menu	29
Menu::menuEntry	32
PaDeviceInfo	34
PaHostApiInfo	35
PaHostErrorInfo	36
PaStreamCallbackTimeInfo	37
PaStreamInfo	37
PaStreamParameters	38
AudioControl::playStruct	43
QGraphicsPixmapItem	
GameObject	25
MovingObject	33
Enemy	20
Player	40
Shoot	47
PowerUp	44
QObject	
Game	22
Input	26
scoreStruct	46
stateStruct	48

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

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

Audio

Audio-Klasse Die Audio-Klasse erzeugt Audioobjekte. Mehrere Instanzen dieser Klasse werden in der Klasse [AudioControl](#) erstellt. Jedes Audioobjekt liest die zum Objekt gehörigen Audiosamples ein und übergibt diese an die Kontrollklasse [Audiocontrol](#), welche im Anschluss die Samples aller Objekte mischt und abspielt. Die einzelnen Methoden werden in der [audio.cpp](#) erklärt . . . 9

AudioControl

AudioControl-Klasse Die [AudioControl](#)-Klasse synchronisiert alle [Audio](#)ausgabeanweisungen und spielt passende [Audio](#)objekte ab. Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt. Die einzelnen Methoden werden in der [audiocontrol.cpp](#) erklärt 11

audioCooldownstruct

Struktur für [audioevents](#) mit ihrer abspielzeit als [Cooldown](#) 15

audioCooldownStruct

Typdef Struct mit Konstanten für den [Audiocooldown](#) jedes [AudioTypes](#) In diesen Konstanten wird festgelegt wie viele millisekunden für ein Event (mit "id=...") eines [audioTypes](#) trotz verschwinden in der Grafik nachwievor [audioStructs](#) gesendet werden. Ein 0 bedeutet, dass kein [Cooldown](#) erfolgt, die [Audiostructs](#) werden hier solange gesendet wie das Event sichtbar ist 15

audioDistanceStruct

Typdef Struct mit Konstanten für die [Distance](#) jedes [AudioTypes](#) In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit id=...) eines [audioTypes](#) vom Spieler standardmäßig auftritt [Werbereich 0 (beim spieler) bis 1(maximale Distanz des Fensters). Ist die Konstante -1 ist die [Distance](#) eines Events vom Typ [audioType](#) variabel und muss von der [Gamelogik](#) bestimmt werden 16

audioStruct

Struktur für einzelne [Audio](#) Events [AudioControl](#) arbeitet Events von dieser Struktur ab. Jedes [Audioevent](#) hat eine eindeutige int "id", einen enum->integer Gruppen "type" und eine float "distance" und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds ändern kann. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung im Gamefenster). Die Standarddistanzwerte sind in "typedef struct [audioDistance](#)" für jeden [AudioStruct](#) "type" definiert 17

collisionStruct

Struktur für die Events Enthält [affectedObject](#) als Objekt, aus dessen Sicht die Kollision berechnet wurde. [affectedObject](#) ist immer ein [MovingObject](#), [causingObject](#) kann beides sein. Die Art und Richtung der Kollision werden mit gespeichert 18

compareGameObjects

Vergleicht zwei [GameObjects](#), bezüglich der X-Position 19

compareScores	Vergleich zweier Scores Der Vergleich findet über die Summe der Punkte in den einzelnen Kategorien statt. Der Operator im struct ist mit größer (>) programmiert, da die Liste absteigend sortiert werden soll	19
Enemy	20
Game	Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels. Innerhalb der main.cpp wird eine Instanz dieser Klasse angelegt, aus der heraus das gesamte Spiel läuft. Die einzelnen Methoden werden in der game.cpp jeweils erklärt	22
GameObject	25
Input	Input-Klasse Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben. Eine Instanz dieser Klasse wird innerhalb der game.h angelegt. Die einzelnen Methoden werden in der input.cpp erklärt	26
Menu	Menü-Klasse eine Instanz repräsentiert ein Menü mit diesen Funktionen:	29
Menu::menuEntry	Struct zur Beschreibung eines Menü-Eintrags	32
MovingObject	33
PaDeviceInfo	34
PaHostApiInfo	35
PaHostErrorInfo	36
PaStreamCallbackTimeInfo	37
PaStreamInfo	37
PaStreamParameters	38
Player	40
AudioControl::playStruct	43
PowerUp	Klasse für Power-Ups	44
scoreStruct	Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. Alkohol-Punkte erhält der Spieler für einen gewissen Pegel in einem Zeitabschnitt	46
Shoot	47
stateStruct	Struktur für die States des Spiels Sowohl Sound- als auch Grafik-Ausgabe erhalten aus den States Informationen darüber, was gerade im Spiel passiert, z.B. dass gerade der Spieler angreift, ein Gegner stirbt etc	48

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Wiesn-Run/src/ audio.h	??
Wiesn-Run/src/ audiocontrol.h	??
Wiesn-Run/src/ definitions.h	??
Wiesn-Run/src/ enemy.h	??
Wiesn-Run/src/ game.h	??
Wiesn-Run/src/ gameobject.h	??
Wiesn-Run/src/ input.h	??
Wiesn-Run/src/ menu.h	??
Wiesn-Run/src/ movingobject.h	??
Wiesn-Run/src/ player.h	??
Wiesn-Run/src/ portaudio.h	
The portable PortAudio API	51
Wiesn-Run/src/ powerup.h	??
Wiesn-Run/src/ shoot.h	??

Kapitel 5

Klassen-Dokumentation

5.1 Audio Klassenreferenz

Audio-Klasse Die Audio-Klasse erzeugt Audioobjekte. Mehrere Instanzen dieser Klasse werden in der Klasse [Audio-Control](#) erstellt. Jedes Audioobjekt liest die zum Objekt gehörigen Audiosamples ein und übergibt diese an die Kontrollklasse Audiocontrol, welche im Anschluss die Samples aller Objekte mischt und abspielt. Die einzelnen Methoden werden in der audio.cpp erklärt.

```
#include <audio.h>
```

Öffentliche Methoden

- [Audio](#) (std::string state_name)
[Audio::Audio](#) Konstruktor instanziert ein Objekt der Klasse [Audio](#).
- [~Audio](#) ()
[Audio::~~Audio](#) Destruktor löscht ein Objekt der Klasse [Audio](#).
- std::string [getSource](#) ()
[Audio::getSource](#) "getSource" gibt bei Aufruf den Namen des Objektes zurück welcher dem Pfad in der Ressourcendatenbank entspricht.
- float [getSample](#) (int pos)
[Audio::getSample](#) "getSample" gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 16 bit zurück.
- int [getSamplenumber](#) ()
[Audio::getSamplenumber](#) "getSamplenumber" gibt bei Aufruf die Anzahl an Samples der zu Audioobjekt gehörigen Wave Datei zurück.
- short [getVolume](#) ()
[Audio::getVolume](#) "getVolume" gibt bei Aufruf zurück wie laut die Samples im Moment eingestellt sind. Der Rückgabewert "volume" ist ein short Datentyp im Bereich (still) 0...1 (maximal laut)
- void [setVolume](#) (short volume_audio_obj)
[Audio::setVolume](#) "setVolume" setzt bei Aufruf für das [Audio](#) Objekt einen Lautstärkinformations Wert "volume". "volume" ist ein short Datentyp im Bereich (still) 0...1 (maximal laut).

5.1.1 Ausführliche Beschreibung

Audio-Klasse Die Audio-Klasse erzeugt Audioobjekte. Mehrere Instanzen dieser Klasse werden in der Klasse [Audio-Control](#) erstellt. Jedes Audioobjekt liest die zum Objekt gehörigen Audiosamples ein und übergibt diese an die Kontrollklasse Audiocontrol, welche im Anschluss die Samples aller Objekte mischt und abspielt. Die einzelnen Methoden werden in der audio.cpp erklärt.

Autor

Felix Pfreundtner

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 `Audio::Audio (std::string state_name)`

[Audio::Audio](#) Konstruktor instanziert ein Objekt der Klasse [Audio](#).

Autor

Felix Pfreundtner

5.1.2.2 `Audio::~~Audio ()`

[Audio::~~Audio](#) Destruktor löscht ein Objekt der Klasse [Audio](#).

Autor

Felix Pfreundtner

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 `float Audio::getSample (int pos)`

[Audio::getSample](#) "getSample" gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 16 bit zurück.

Rückgabe

float sample

Autor

Felix Pfreundtner

gebe Sample des Audioobjekts an der Stelle pos zurück

5.1.3.2 `int Audio::getSamplenumber ()`

[Audio::getSamplenumber](#) "getSamplenumber" gibt bei Aufruf die Anzahl an Samples der zu Audioobjekt gehörigen Wave Datei zurück.

Rückgabe

int samplenumber

Autor

Felix Pfreundtner

5.1.3.3 `std::string Audio::getSource ()`

[Audio::getSource](#) "getSource" gibt bei Aufruf den Namen des Objektes zurück welcher welcher dem Pfad in der Ressourcendatenbank entspricht.

Rückgabe

`std::string source`

Autor

Felix Pfreundtner

5.1.3.4 `short Audio::getVolume ()`

[Audio::getVolume](#) "getVolume" gibt bei Aufruf zurück wie laut die Samples im Moment eingestellt sind. Der Rückgabewert "volume" ist ein short Datentyp im Bereich (still) 0...1 (maximal laut)

Rückgabe

`short volume`

Autor

Felix Pfreundtner

5.1.3.5 `void Audio::setVolume (short volume_audio_obj)`

[Audio::setVolume](#) "setVolume" setzt bei Aufruf für das [Audio](#) Objekt einen Lautstärkinformations Wert "volume". "volume" ist ein short Datentyp im Bereich (still) 0...1 (maximal laut).

Autor

Felix Pfreundtner

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesen-Run/src/audio.h
- Wiesen-Run/src/audio.cpp

5.2 AudioControl Klassenreferenz

AudioControl-Klasse Die AudioControl-Klasse synchronisiert alle Audioausgabeeigenschaften und spielt passende Audioobjekte ab. Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt. Die einzelnen Methoden werden in der `audiocontrol.cpp` erklärt.

```
#include <audiocontrol.h>
```

Klassen

- struct [playStruct](#)

Öffentliche Typen

- enum **statusFilter** { **no**, **alcohol**, **life**, **lifecritical** }

Öffentliche Methoden

- [AudioControl](#) ()
[AudioControl::AudioControl](#) Konstruktor instanziert ein Objekt der Klasse [AudioControl](#).
- [~AudioControl](#) ()
[AudioControl::~~AudioControl](#) Destruktor löscht ein Objekt der Klasse [AudioControl](#).
- void [playInitialize](#) ()
[playInitialize](#) [playInitialize](#) initialisiert die Abspielbibliothek Portaudio, öffnet den PortAudio Stream pastream und startet eine Callback Audiowiedergabe
- void [update](#) (std::list< struct [audioStruct](#) > *audioevents)
[update](#) [update](#) aktualisiert die im Moment abgespielten, in der Liste "playevents" gespeicherten [playStruct](#)'s mit aktuellen [audioStruct](#)'s aus der Liste audioevents der GameLogik.

5.2.1 Ausführliche Beschreibung

AudioControl-Klasse Die AudioControl-Klasse synchronisiert alle Audioausgabeeinstellungen und spielt passende Audioobjekte ab. Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt. Die einzelnen Methoden werden in der [audiocontrol.cpp](#) erklärt.

Autor

Felix Pfreundtner

5.2.2 Beschreibung der Konstruktoren und Destrukturen

5.2.2.1 [AudioControl::AudioControl](#) ()

[AudioControl::AudioControl](#) Konstruktor instanziert ein Objekt der Klasse [AudioControl](#).

Autor

Felix Pfreundtner

erstelle für jede objektgruppe "type" ein audio Objekt welches unter anderem die Samples beinhaltet

setzte Status Filter auf kein Filter

setzte blockcounter auf 0 Blöcke

setzte Wartezeit von Portaudio auf 100 ms

setzte maximale Anzahl an Playevents auf 5 Wird der Wert höher gesetzt wird die Lautstärke geringer. Wird der Wert geringer gesetzt steigt die Gefahr des Clippings des Ausgabesignals Ein Normalisieren aller Ausgabeblöcke wäre möglich, würde jedoch 2D [Audio](#) nicht erlauben, da auch die Dynamik zwischen zwei Blöcken normalisiert wird -> Distanz- und somit Lautstärkeänderungen von Objekten werden mit normalisiert.

5.2.2.2 [AudioControl::~~AudioControl](#) ()

[AudioControl::~~AudioControl](#) Destruktor löscht ein Objekt der Klasse [AudioControl](#).

Autor

Felix Pfreundtner

Stoppe den Portaudio Stream

Schließe den Portaudio Stream

Beende PortAudio

5.2.3 Dokumentation der Elementfunktionen**5.2.3.1 void AudioControl::playInitialize ()**

playInitialize playInitialize initialisiert die Abspielbibliothek Portaudio, öffnet den PortAudio Stream pastream und startet eine Callback Audiowiedergabe

Parameter

<i>Q/list</i>	audioevents
---------------	-------------

Autor

Felix Pfreundtner

initialisiere Port [Audio](#)

Öffene Ausgabe Stream pastream

erstelle keine Eingangskänale

erstelle Mono [Audio](#) Ausgabe

setze Bittiefe der Audioausgabe 16 bit Integer

setze Samplerate der Audioausgabe zu 44100 Hz

setze Anzahl an Samples per Bufferblock auf 1024

verweise auf Static Callback Funktion

übergebe User-Data

Starte PortAudio Stream pastream

Pausiere Funktion wenn Audiostream gerade aktiv ist (Audiowiedergabe übernimmt Callback Funktion)

5.2.3.2 void AudioControl::update (std::list< struct audioStruct > * audioevents)

update update aktualisiert die im Moment abgespielten, in der Liste "playevents" gespeicherten [playStruct](#)'s mit aktuellen [audioStruct](#)'s aus der Liste audioevents der GameLogik.

Parameter

<i>Q/list</i>	audioevents
---------------	-------------

Autor

Felix Pfreundtner

erstelle neues temporäres [audioStruct](#), welches stets das aktuelle [audioStruct](#) Element der Liste audioevents beinhaltet.

erstelle neues temporäres [audioStruct](#), welches stets das aktuelle [playStruct](#) Element der Liste playevents beinhaltet.

erstelle Variable welche true ist wenn die id von newplaystruct bereits in playevents vorhanden ist

erstelle einen Iterator für playevents Liste

initialisiere status_filter mit no (Annahme kein Audioevent mit type status_alcohol / status_life / status_lifecritical in audioevents Liste)

initialisiere die Abspielinformation aller playstructs in playevents auf false (verhindere weiteres abspielen im nächsten Step)

aktualisiere die Abspielinformation und Distanzwerte aller playstructs aus playevents mit aktuellen GameLogik Werten iteriere über alle audioStructs der audioevents Liste

entnehme neues [audioStruct](#) aus audioevents Liste

setzte Variable nasidexistinpe auf false, da newaudiostruct bisher nicht in playevents gefunden wurde

iteriere über alle bereits bestehenden playStructs in Liste playevents

falls die id eines neuen audiostruct bereits in diesem [playStruct](#) von playevents vorhanden ist (also bereits abgespielt wird)

übernehmen die aktuellen Distanzwerte des neuen audiostructs und wandle sie in eine Volumen Information um (volume = 1 - distance).

setzte playnext auf true, da das playstruct auch im nächsten Step abgespielt werden soll

setzte Variable nasidexistinpe auf true, da newaudiostruct bisher in playevents gefunden wurde

wenn die id von newaudiostruct noch nicht in audioevents vorhanden ist (struct noch nicht abgespielt wird)

schreibe ID des neuen [audioStruct](#) in ein neues [playStruct](#)

schreibe Gruppen Type des neuen [audioStruct](#) in ein neues [playStruct](#)

übernehmen die aktuellen Distanzwerte des neuen audiostructs und wandle sie in eine Volumen Information um (volume = 1 - distance).

speichere einen Zeiger auf das (Audio-)Objekt in audioobjects in newplaystruct

setzte Abspielposition auf 0 Samples (Beginne Abspielen)

setzte playnext auf true, da das playstruct auch im nächsten Step abgespielt werden soll

füge das neue playstruct der Liste playevents hinzu

Lösche [audioStruct](#) aus audioevents Liste da alle Werte in playevents übernommen wurden

für alle Playevents in der playevents liste

falls Playevent im nächsten Step nicht mehr abgespielt werden sollen (playnext = false).

lösche playevent aus playevents List

falls playevent im nächsten Step abgespielt werden soll

falls Type des aktuellen playevents status_alcohol ist (niedrige Priorität)

setze Alkohol Status auf aktiv

falls Type des aktuellen playevents status_life ist (mittlere Priorität)

setze Alkohol Status auf aktiv

falls Type des aktuellen playevents status_lifecritical ist (höchste Priorität)

setze Alkohol Status auf aktiv

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/audiocontrol.h
- Wiesn-Run/src/audiocontrol.cpp

5.3 audioCooldownstruct Strukturreferenz

Struktur für audioevents mit ihrer abspielzeit als Cooldown.

```
#include <definitions.h>
```

Öffentliche Attribute

- struct [audioStruct](#) **audioEvent**
- std::chrono::duration< int,
std::milli > **cooldown**

5.3.1 Ausführliche Beschreibung

Struktur für audioevents mit ihrer abspielzeit als Cooldown.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.4 audioCooldownStruct Strukturreferenz

Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes In diesen Konstanten wird festgelegt wie viele millisekunden für ein Event (mit "id=...") eines audioTypes trotz verschwinden in der Grafik nachwievor audioStructs gesendet werden. Ein 0 bedeutet, dass kein Cooldown erfolgt, die Audiostructs werden hier solange gesendet wie das Event sichtbar ist.

```
#include <definitions.h>
```

Öffentliche Attribute

- std::chrono::duration< int,
std::milli > **scene_flyingbeer** = std::chrono::milliseconds(0)
- std::chrono::duration< int,
std::milli > **scene_enemy_security** = std::chrono::milliseconds(0)
- std::chrono::duration< int,
std::milli > **scene_enemy_tourist** = std::chrono::milliseconds(0)
- std::chrono::duration< int,
std::milli > **scene_enemy_boss** = std::chrono::milliseconds(0)
- std::chrono::duration< int,
std::milli > **scene_collision_obstacle** = std::chrono::milliseconds(306)
- std::chrono::duration< int,
std::milli > **scene_collision_enemy** = std::chrono::milliseconds(899)
- std::chrono::duration< int,
std::milli > **scene_collision_player** = std::chrono::milliseconds(2989)
- std::chrono::duration< int,
std::milli > **scene_collision_flyingbeer** = std::chrono::milliseconds(1211)
- std::chrono::duration< int,
std::milli > **powerup_beer** = std::chrono::milliseconds(2989)
- std::chrono::duration< int,
std::milli > **powerup_food** = std::chrono::milliseconds(1989)
- std::chrono::duration< int,
std::milli > **status_alcohol** = std::chrono::milliseconds(0)
- std::chrono::duration< int,
std::milli > **status_life** = std::chrono::milliseconds(0)

- `std::chrono::duration< int,`
`std::milli > status_lifecritical = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > status_dead = std::chrono::milliseconds(4989)`
- `std::chrono::duration< int,`
`std::milli > player_walk = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > player_jump = std::chrono::milliseconds(354)`
- `std::chrono::duration< int,`
`std::milli > background_menu = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > background_highscore = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > background_level1 = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > background_level2 = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > background_level3 = std::chrono::milliseconds(0)`
- `std::chrono::duration< int,`
`std::milli > background_startgame = std::chrono::milliseconds(2838)`
- `std::chrono::duration< int,`
`std::milli > background_levelfinished = std::chrono::milliseconds(5365)`

5.4.1 Ausführliche Beschreibung

Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes In diesen Konstanten wird festgelegt wie viele millisekunden für ein Event (mit "id=...") eines audioTypes trotz verschwinden in der Grafik nachwievor audioStructs gesendet werden. Ein 0 bedeutet, dass kein Cooldown erfolgt, die Audiostructs werden hier solange gesendet wie das Event sichtbar ist.

Autor

Felix

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.5 audioDistanceStruct Strukturreferenz

Typdef Struct mit Konstanten für die Distance jedes Audiotypes In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit id=...) eines audioTypes vom Spieler standardmäßig auftritt [Werbereich 0 (beim spieler) bis 1(maximale Distanz des Fensters). Ist die Konstante -1 ist die Distance eines Events vom Typ audioType variabel und muss von der Gamelogik bestimmt werden.

```
#include <definitions.h>
```

Öffentliche Attribute

- float **scene_flyingbeer** = -1
- float **scene_enemy_tourist** = -1
- float **scene_enemy_security** = -1
- float **scene_enemy_boss** = -1
- float **scene_collision_obstacle** = 0
- float **scene_collision_enemy** = 0

- float **scene_collision_player** = 0
- float **scene_collision_flyingbeer** = 0
- float **powerup_beer** = 0
- float **powerup_food** = 0
- float **status_alcohol** = 0
- float **status_life** = 0
- float **status_lifecritical** = 0
- float **status_dead** = 0
- float **player_walk** = 0
- float **player_jump** = 0
- float **background_menu** = 0.1
- float **background_highscore** = 0.1
- float **background_level1** = 0.3
- float **background_level2** = 0.3
- float **background_level3** = 0.3
- float **background_startgame** = 0.0
- float **background_levelfinished** = 0.0

5.5.1 Ausführliche Beschreibung

Typdef Struct mit Konstanten für die Distance jedes Audiotypes In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit id=...) eines audioTypes vom Spieler standardmäßig auftritt [Werbereich 0 (beim spieler) bis 1(maximale Distanz des Fensters). Ist die Konstante -1 ist die Distance eines Events vom Typ audioType variabel und muss von der Gamelogik bestimmt werden.

Autor

Felix

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.6 audioStruct Strukturreferenz

Struktur für einzelne [Audio](#) Events [AudioControl](#) arbeitet Events von dieser Struktur ab. Jedes Audioevent hat eine eindeutige int "id", einen enum->integer Gruppen "type" und eine float "distance" und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds ändern kann. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung im Gamefenster). Die Standarddistanzwerte sind in "typedef struct audio-Distance" für jeden AudioStruct "type" definiert.

```
#include <definitions.h>
```

Öffentliche Attribute

- int **id**
- audioType **type**
- float **distance**

5.6.1 Ausführliche Beschreibung

Struktur für einzelne [Audio](#) Events [AudioControl](#) arbeitet Events von dieser Struktur ab. Jedes Audioevent hat eine eindeutige int "id", einen enum->integer Gruppen "type" und eine float "distance" und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds ändern kann. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung im Gamefenster). Die Standarddistanzwerte sind in "typedef struct audioDistance" für jeden AudioStruct "type" definiert.

Alle in einem Step auftretenden [audioStruct](#)'s werden in einer std::list audioevents gesammelt ([game.h](#)) und über die Methode update() in jedem Step der Klasse Audiocontrol übergeben. Audiocontrol steuert den richtigen Abspieltyp jedes [audioStruct](#). Nach jedem Step wird die Liste gelöscht und wieder neu mit audioStructs gefüllt. [Audio](#) Events welche in der GameLogik nur einmal auftreten, wie ein Powerup aufnehmen, werden mit einem Cooldown Timer zusätzlich länger an die Liste audioevents angehängt um ein weiteres Abspielen zu garantieren. Die Dauer des Cooldown Timers ist in "typedef struct audioCooldown" für jeden AudioStruct "type" definiert.

Ist ein Event mit zu erfolgreicher Audioausgabe vorhanden wird ein [audioStruct](#) mit Eventname und aktueller Distanz des Audio-Events vom Spieler zum Event erstellt. Dieses Audiostruct wird an die Liste audioevents mit allen im Step stattfinden audioStructs angehängt. Ist ein Objekt / Event nachwievor aktiv in der Szene wird das Struct im nächsten Step wieder an die Liste audioevents angehängt und die [audioStruct](#) "id" konstant gehalten. Ist ein Objekt nicht mehr in der Szene zu sehen, so muss kein [audioStruct](#) übergeben werden. Die [audioStruct](#) "id" dieses Objekts wird im weiteren Spielverlauf nicht mehr verwendet.

Befindet sich z.B. ein Bier mit "id = ..." in der Szene, so ist der "type = scene_beer". In jedem Step muss in der Audio-Struktur die "distance" des Biers zum Spieler aktualisiert werden und an die Liste audioevents angehängt werden. Verschwindet das Bierobjekt so wird das [audioStruct](#) nicht mehr übergeben und die "id" nicht mehr verwendet. Gibt es mehrere Bierobjekte so wird das Struct mit Gruppen "type=scene_beer" mit verschiedenen "id"s an die Liste angehängt.

Läuft der Spieler im aktuellen Step so wird das [audioStruct](#) "player_walk" erstellt("distance" stets 0). Läuft er im nächsten Step nachwievor (hat also seine Position geändert) wird das Audiostruct wieder an die audioevents Liste angehängt. Läuft er nicht mehr wird es nicht mehr an die audioevents liste angehängt.

Ist gerade das Level 1 aktiv so wird in jedem Step ein [audioStruct](#) mit "ID=..." und "type=background_level1" an die Liste angehängt. Bei Background Musik ist "distance=0.5". Dies bewirkt dass sie leiser als Playersounds (distance = 0) abgespielt wird.

Autor

Felix Pfreundtner

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesen-Run/src/definitions.h

5.7 collisionStruct Strukturreferenz

Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde. affectedObject ist immer ein [MovingObject](#), causingObject kann beides sein. Die Art und Richtung der Kollision werden mit gespeichert.

```
#include <game.h>
```

Öffentliche Attribute

- [GameObject](#) * **affectedObject**
- [GameObject](#) * **causingObject**
- enum collisionDirection **direction**

5.7.1 Ausführliche Beschreibung

Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde. affected-Object ist immer ein [MovingObject](#), causingObject kann beides sein. Die Art und Richtung der Kollision werden mit gespeichert.

Autor

Simon, Johann(15.6)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.h

5.8 compareGameObjects Strukturreferenz

Vergleicht zwei GameObjects, bezüglich der X-Position.

Öffentliche Methoden

- bool **operator()** ([GameObject](#) *objA, [GameObject](#) *objB)

5.8.1 Ausführliche Beschreibung

Vergleicht zwei GameObjects, bezüglich der X-Position.

Parameter

<i>1.Objekt</i>	
<i>2.Objekt</i>	

Rückgabe

true, wenn 1.Objekt weiter links als 2.Objekt

Autor

Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.cpp

5.9 compareScores Strukturreferenz

Vergleich zweier Scores Der Vergleich findet über die Summe der Punkte in den einzelnen Kategorien statt. Der Operator im struct ist mit größer (>) programmiert, da die Liste absteigend sortiert werden soll.

Öffentliche Methoden

- bool **operator()** ([scoreStruct](#) scoreA, [scoreStruct](#) scoreB)

5.9.1 Ausführliche Beschreibung

Vergleich zweier Scores Der Vergleich findet über die Summe der Punkte in den einzelnen Kategorien statt. Der Operator im struct ist mit größer (>) programmiert, da die Liste absteigend sortiert werden soll.

Autor

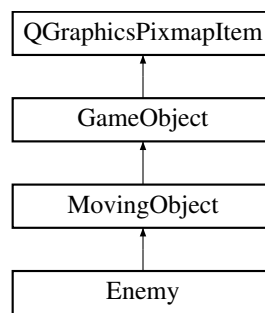
Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.cpp

5.10 Enemy Klassenreferenz

Klassendiagramm für Enemy:



Öffentliche Methoden

- **Enemy** (int posX, int posY, int speedX, objectType enemy)
Konstruktor für ein Enemy-Objekt.
- int **getHealth** () const
Enemy::getHealth Gibt Lebensstand zurück.
- void **setHealth** (int health)
Enemy::setHealth Lebensstand wird gesetzt.
- bool **receiveDamage** (int damage)
- int **getInflictedDamage** () const
Enemy::getInflictedDamage gibt Schaden zurück, den der gegner zufügt.
- int **getFireCooldown** () const
Enemy::getFireCooldown.
- bool **getDeath** () const
Enemy::getDeath Gibt an ob der Gegner Tot ist.
- void **setDeath** (bool death)
Enemy::setDeath Zustand-TOT wird gesetzt.
- int **getDeathCooldown** () const
Enemy::getDeathCooldown.
- virtual void **update** ()
Enemy::update führt Bewegungen des Gegners aus.

Weitere Geerbte Elemente

5.10.1 Beschreibung der Konstruktoren und Destruktoren

5.10.1.1 `Enemy::Enemy (int posX, int posY, int speedX, objectType enemy)`

Konstruktor für ein Enemy-Objekt.

Class [Enemy](#) lastUpdate: [update\(\)](#) 10.6 Johann

Parameter

<i>posX</i>	: X-Position
<i>posY</i>	: Y-Position
<i>speedX</i>	: Geschwindigkeit in X-Richtung

[Noch zu erledigen](#) Skalieren der Werte und fireCooldown erhöhen

5.10.2 Dokumentation der Elementfunktionen

5.10.2.1 `bool Enemy::getDeath () const`

[Enemy::getDeath](#) Gibt an ob der Gegner Tot ist.

Rückgabe

: Zustand - TOT

5.10.2.2 `int Enemy::getDeathCooldown () const`

[Enemy::getDeathCooldown](#).

Rückgabe

deathCooldown

5.10.2.3 `int Enemy::getFireCooldown () const`

[Enemy::getFireCooldown](#).

Rückgabe

fireCooldown

5.10.2.4 `int Enemy::getHealth () const`

[Enemy::getHealth](#) Gibt Lebensstand zurück.

Rückgabe

: Lebensstand

5.10.2.5 int Enemy::getInflictedDamage () const

[Enemy::getInflictedDamage](#) gibt Schaden zurück, den der gegner zufügt.

Rückgabe

: Schaden

5.10.2.6 void Enemy::setDeath (bool *death*)

[Enemy::setDeath](#) Zustand-TOT wird gesetzt.

Parameter

<i>death</i>	: Zustand-TOT
--------------	---------------

5.10.2.7 void Enemy::setHealth (int *health*)

[Enemy::setHealth](#) Lebensstand wird gesetzt.

Parameter

<i>health</i>	: Lebensstand
---------------	---------------

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

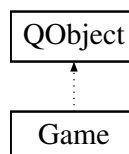
- Wiesn-Run/src/enemy.h
- Wiesn-Run/src/enemy.cpp

5.11 Game Klassenreferenz

Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels. Innerhalb der main.cpp wird eine Instanz dieser Klasse angelegt, aus der heraus das gesamte Spiel läuft. Die einzelnen Methoden werden in der game.cpp jeweils erklärt.

```
#include <game.h>
```

Klassendiagramm für Game:



Öffentliche Methoden

- [Game](#) (int argc, char *argv[])
Konstruktor Initialisiert den appPointer.
- int [step](#) ()
Startet das die Game-Loop, wird einmalig von main() aufgerufen.
- int [run](#) (QApplication &app)
Startet die Mockup QApplication app.
- int [start](#) ()

Die Startfunktion, erstellt Fenster und Menüs, wird von `main()` aufgerufen Grafik und Inputs (Flo,Felix): Erstelle Q-Application app mit `QGraphicsView` Widget `window` (Eventfilter installiert) und Zeiger input auf `Input` Objekt. Um Funktionen der Tastatur Eingabe entwickeln zu können ist ein Qt Widget Fenster nötig. Auf dem Widget wird ein Eventfilter installiert welcher kontinuierlich Tastatureingaben mitloggt. Die Eingaben werden in dem Objekt der `Input` Klasse gespeichert und können über `getKeyactions()` abgerufen werden.

- void `setState` (enum `gameState` `newState`)
setzt den Spielstatus

Öffentliche Attribute

- struct `stateStruct` `gameStats`
- std::list< struct `collisionStruct` > `collisionsToHandle`

Geschützte Methoden

- void `timerEvent` (QTimerEvent *event)
wird regelmäßig aufgerufen event muss drinstehen, damit der Timer die Funktion aufruft

5.11.1 Ausführliche Beschreibung

Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels. Innerhalb der `main.cpp` wird eine Instanz dieser Klasse angelegt, aus der heraus das gesamte Spiel läuft. Die einzelnen Methoden werden in der `game.cpp` jeweils erklärt.

funktion `handleCollisions` hinzugefügt

Autor

Simon, Johann, Felix

5.11.2 Beschreibung der Konstruktoren und Destruktoren

5.11.2.1 `Game::Game (int argc, char * argv[])`

Konstruktor Initialisiert den `appPointer`.

Parameter

<code>argc</code>	
<code>argv</code>	

Autor

Rupert

Initialisiert den `appPointer` mit der `QApplication`

5.11.3 Dokumentation der Elementfunktionen

5.11.3.1 void `Game::setState (enum gameState newState)`

setzt den Spielstatus

Parameter

<i>newState</i>	
-----------------	--

Autor

Rupert

5.11.3.2 `int Game::start ()`

Die Startfunktion, erstellt Fenster und Menüs, wird von `main()` aufgerufen Grafik und Inputs (Flo,Felix): Erstelle Q-Application app mit `QGraphicsView` Widget window (Eventfilter installiert) und Zeiger input auf `Input` Objekt. Um Funktionen der Tastatur Eingabe entwickeln zu können ist ein Qt Widget Fenster nötig. Auf dem Widget wird ein Eventfilter installiert welcher kontinuierlich Tastatureingaben mitloggt. Die Eingaben werden in dem Objekt der `Input` Klasse gespeichert und können über `getKeyactions()` abgerufen werden.

Logik (Rupert): Außerdem wird ein Timer gestartet, der in jedem Intervall `timerEvent(...)` aufruft, wo dann `step()` aufgerufen wird. Das ist dann unsere Game-Loop. Der Timer funktioniert auch bei 5ms Intervall noch genau. Menüs (Rupert): Alle Menüs werden angelegt

`gameState` wird auf `gameMenuStart` gesetzt, dh das Spiel startet im Startmenü

Rückgabe

Rückgabewert von `app.exec()`

Autor

Rupert

Installiere Event Filter zum Loggen der Keyboard Eingabe

Erstelle Audiocontrol Objekt zum Einlesen der Audiodatein und speichern der Ausgabeparameter

Erstelle einen neuen Thread `portaudiothread`. Initialisiere dort `PortAudio` und beginne eine Audioausgabe zu erzeugen.

Noch zu erledigen hier wird das Startmenü übersprungen

5.11.3.3 `int Game::step ()`

Startet das die Game-Loop, wird einmalig von `main()` aufgerufen.

Game-Loop Diese Funktion wird von `timerEvent()` aufgerufen und ist für den kompletten Ablauf des Spiels verantwortlich. grober Ablauf: LOOP:

- Timer starten
- Neue Objekte zur Welt hinzufügen
- alte Objekte löschen
- `Input` auslesen
- Bewegungen berechnen
- Kollisionskontrolle
- Bewegungen korrigieren

- Events behandeln (Treffer..)
- Grafik rendern und ausgeben
- [Audio](#) ausgeben
- verbleibende Zeit im Slot berechnen (Timer auslesen)
- entsprechend warten goto LOOP

Rückgabe

0 bei fehlerfreiem Beenden

Autor

Rupert

Tasten abfragen

Zeit seit dem letzten Aufruf ausrechnen und ausgeben

[Noch zu erledigen](#) Levelende hier abfragen

delete List [audioStruct](#) elements in list and fill it in the next step again

5.11.3.4 `void Game::timerEvent (QTimerEvent * event)` [protected]

wird regelmäßig aufgerufen event muss drinstehen, damit der Timer die Funktion aufruft

Parameter

<i>event</i>	
--------------	--

Autor

Rupert

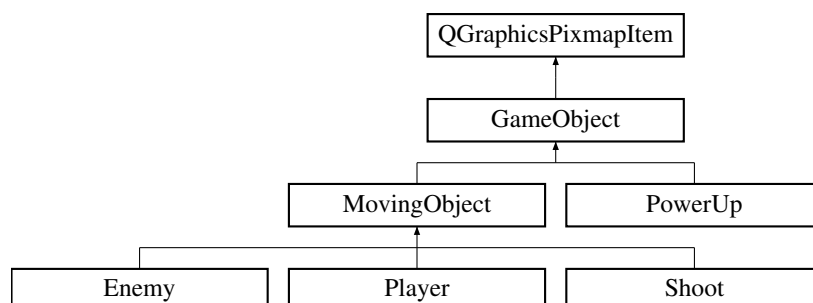
[Noch zu erledigen](#) return von step...

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/game.h
- Wiesn-Run/src/game.cpp

5.12 GameObject Klassenreferenz

Klassendiagramm für GameObject:



Öffentliche Methoden

- [GameObject](#) (int posX, int posY, int length, int height, objectType type)
GameObject::GameObject Konstruktor.
- [GameObject](#) (int posX, int posY, objectType type)
- int **getPosX** () const
- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- objectType **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Attribute

- int **posX**
- int **posY**

5.12.1 Beschreibung der Konstruktoren und Destruktoren

5.12.1.1 [GameObject::GameObject](#) (int *posX*, int *posY*, int *length*, int *height*, objectType *type*)

[GameObject::GameObject](#) Konstruktor.

Parameter

<i>length</i>	: Länge
<i>height</i>	: Höhe
<i>type</i>	: Typ
<i>posX</i>	: X-Position
<i>posY</i>	: Y-Position
<i>colType</i>	: Kollisionstyp

Autor

Johann

5.12.1.2 [GameObject::GameObject](#) (int *posX*, int *posY*, objectType *type*)

Noch zu erledigen Grafik für die Security

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

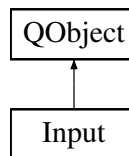
- Wiesn-Run/src/gameobject.h
- Wiesn-Run/src/gameobject.cpp

5.13 Input Klassenreferenz

Input-Klasse Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben. Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt. Die einzelnen Methoden werden in der input.cpp erklärt.

```
#include <input.h>
```

Klassendiagramm für Input:



Öffentliche Typen

- enum **Keyaction** {
noKey, **Left**, **Right**, **Up**,
Down, **Jump_Right**, **Shoot**, **Exit**,
Enter }

Öffentliche Methoden

- **Input** ()
Input::Input Konstruktor instanziert ein Objekt der Klasse **Input**.
- **~Input** ()
Input::~~Input Destruktor löscht ein Objekt der Klasse **Input**.
- **QSet< int > getKeyactions** ()
Input::getKeyactions *getKeyactions* gibt bei Aufruf die Instanzvariable *keyactions* zurück. Jede Tastaturkombination besitzt eine Integer ID welche im QSet *keyactions* gespeichert ist. Die IDs sind über die Enumeration *Input::Keyaction* mit lesbaren Spielbefehlen verknüpft. Möchte man nun beispielsweise abfragen ob der Spieler im Moment schießt so überprüft man: *input->getKeyactions().contains(Input::Keyaction::Shoot) == True*.
- **Keyaction getAndDeleteLastKey** ()
Input::getAndDeleteLastKey Gibt letzte gedrückte Taste zurück und löscht diese. Wird für die Menüführung gebraucht, da ein dauerhaftes Auswerten der Tasten dort zu Sprüngen beim Auswählen der Menü Einträge führt.

Geschützte Methoden

- **bool eventFilter** (QObject *obj, QEvent *event)
Input::eventFilter *eventFilter* sammelt alle im Momment gepressten Tastatur Eingaben und speichert die integer IDs in der Instanzvariable *keyevents*. wird eine Taste nicht mehr gedrückt wird die id in *keyevents* gelöscht wird eine Taste neu gedrückt wird die id in *keyevents* hinzugefügt.

5.13.1 Ausführliche Beschreibung

Input-Klasse Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben. Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt. Die einzelnen Methoden werden in der [input.cpp](#) erklärt.

Autor

Felix Pfreundtner

5.13.2 Beschreibung der Konstruktoren und Destruktoren

5.13.2.1 Input::Input ()

Input::Input Konstruktor instanziert ein Objekt der Klasse **Input**.

Autor

Felix Pfreundtner

5.13.2.2 Input::~~Input ()

[Input::~~Input](#) Destruktor löscht ein Objekt der Klasse [Input](#).

Autor

Felix Pfreundtner

5.13.3 Dokumentation der Elementfunktionen

5.13.3.1 bool Input::eventFilter (QObject * obj, QEvent * event) [protected]

[Input::eventFilter](#) eventFilter sammelt alle im Momment gepressten Tastatur Eingaben und speichert die integer IDs in der Instanzvariable keyevents. wird eine Taste nicht mehr gedrückt wird die id in keyevents gelöscht wird eine Taste neu gedrückt wird die id in keyevents hinzugefügt.

Parameter

<i>QObject</i>	*obj, QEvent *event
----------------	---------------------

Rückgabe

Boolean

Autor

Felix Pfreundtner

5.13.3.2 Input::Keyaction Input::getAndDeleteLastKey ()

[Input::getAndDeleteLastKey](#) Gibt letzte gedrückte Taste zurück und löscht diese. Wird für die Menüführung gebraucht, da ein dauerhaftes Auswerten der Tasten dort zu Sprüngen beim Auswählen der Menü Einträge führt.

Rückgabe

Enum Keyaction

Autor

Rupert, Felix

5.13.3.3 QSet< int > Input::getKeyactions ()

[Input::getKeyactions](#) getKeyactions gibt bei Aufruf die Instanzvariable keyactions zurück. Jede Tastaturkombination besitzt eine Integer ID welche im QSet keyactions gespeichert ist. Die IDs sind über die Enumeration Input::Keyaction mit lesbaren Spielbefehlen verknüpft. Möchte man nun beispielsweise abfragen ob der Spieler im Moment schießt so überprüft man: `input->getKeyactions().contains(Input::Keyaction::Shoot) == True`.

Rückgabe

Instanzvariable keyactions

Autor

Felix Pfreundtner

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/input.h
- Wiesn-Run/src/input.cpp

5.14 Menu Klassenreferenz

Menü-Klasse eine Instanz repräsentiert ein Menü mit diesen Funktionen:

```
#include <menu.h>
```

Klassen

- struct `menuEntry`
Struct zur Beschreibung eines Menü-Eintrags.

Öffentliche Typen

- enum `menuSelectionChange` { **up**, **down** }
wird von der Menu-Klasse zur Auswahl-Änderung benötigt
- enum `menuType` { **normal**, **highscore** }
für verschiedene Menü-Typen (für Background-Musik)

Öffentliche Methoden

- `Menu` (std::string *menuTitle, `menuType` type=normal)
Menu-Konstruktor.
- void `clear` ()
löscht alle Einträge bis auf den ersten
- `menuType` `getType` ()
gibt den Typ zurück
- std::string * `getTitle` ()
gibt den Titel zurück
- int `displayInit` ()
Initialisiert das angezeigt Menü
- int `displayUpdate` ()
Aktualisiert das angezeigt Menü
- int `addEntry` (std::string name, int id, bool clickable=false, gameState stateOnClick=(gameState) NULL)
Neuen Eintrag hinzufügen (evtl private -> Einträge nur im Konstruktor erstellen -> unterschiedlich viele Argumente)
- int `changeSelection` (`menuSelectionChange` changeType)
wird nach Tastendruck aufgerufen
- `Menu::menuEntry` * `getSelection` ()
Zeiger auf aktuelle gewählten Menüeintrag, sollte nach Enter aufgerufen werden.
- `Menu::menuEntry` * `getEntry` (int position)
Gibt Menü-Eintrag an der entsprechenden Position zurück.

Öffentliche Attribute

- `QGraphicsPixmapItem` * `background`
Zeiger auf die Menü-Scene und das Menü-Hintergrundbild.
- `QGraphicsScene` * `menuScene`

5.14.1 Ausführliche Beschreibung

Menü-Klasse eine Instanz repräsentiert ein Menü mit diesen Funktionen:

- Einträge hinzufügen
- aktuelle Auswahl ändern (nach Tastendruck)
- anzeigen

5.14.2 Beschreibung der Konstruktoren und Destruktoren

5.14.2.1 Menu::Menu (std::string * *menuTitle*, *menuType type* = normal)

Menu-Konstruktor.

Parameter

<i>Zeiger</i>	auf String mit Menu-Titel
---------------	---------------------------

Autor

Rupert

5.14.3 Dokumentation der Elementfunktionen

5.14.3.1 int Menu::addEntry (std::string *name*, int *id*, bool *clickable* = false, gameState *stateOnClick* = (gameState) NULL)

Neuen Eintrag hinzufügen (evtl private -> Einträge nur im Konstruktor erstellen -> unterschiedlich viele Argumente)

Neuen Eintrag hinzufügen.

Parameter

<i>name</i>	String, der angezeigt wird
<i>id</i>	zur eindeutigen Identifizierung, kann zB aus enum gecastet werden

Rückgabe

0 bei Erfolg

Autor

Rupert

5.14.3.2 int Menu::changeSelection (menuSelectionChange *changeType*)

wird nach Tastendruck aufgerufen

Parameter

<i>changeType</i>	entweder up oder down
-------------------	-----------------------

Rückgabe

0 bei Erfolg, -1 wenn kein klickbarer Eintrag vorhanden

Autor

Rupert

5.14.3.3 void Menu::clear ()

löscht alle Einträge bis auf den ersten

entfernt alle Einträge aus dem Menü Titel wird danach wieder hinzugefügt wird für Statistik und Highscore benötigt

5.14.3.4 int Menu::displayInit ()

Initialisiert das angezeigte Menü

Initialisiert das sichtbare Menü , muss immer nach anlegen der Menü Eintrags aufgerufen werden.

Rückgabe

0 bei Erfolg

Autor

Flo

5.14.3.5 int Menu::displayUpdate ()

Aktualisiert das angezeigte Menü

aktualisiert das sichtbare Menü

Rückgabe

0 bei Erfolg

Autor

Flo

5.14.3.6 struct Menu::menuEntry * Menu::getEntry (int *position*)

Gibt Menü-Eintrag an der entsprechenden Position zurück.

gibt Eintrag an der gesuchten Position zurück

Parameter

<i>position</i>	
-----------------	--

Rückgabe

Zeiger auf gefundenen Eintrag, sonst NULL

Autor

Rupert

Schleife startet beim ersten Element und geht bis zum letzten Element durch

5.14.3.7 struct Menu::menuEntry * Menu::getSelection ()

Zeiger auf aktuelle gewählten Menüeintrag, sollte nach Enter aufgerufen werden.

gibt den gewählten Eintrag zurück sollte nach Enter aufgerufen werden

Rückgabe

Zeiger auf [menuEntry](#) des aktuellen Eintrags, NULL bei Fehler

Autor

Rupert

5.14.3.8 std::string * Menu::getTitle ()

gibt den Titel zurück

gibt den Menü-Titel zurück

Rückgabe

Zeiger auf String

Autor

Rupert

5.14.3.9 Menu::menuType Menu::getType ()

gibt den Typ zurück

gibt den Menü-Typ zurück

Rückgabe

enum menuType

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/menu.h
- Wiesn-Run/src/menu.cpp

5.15 Menu::menuEntry Strukturreferenz

Struct zur Beschreibung eines Menü-Eintrags.

```
#include <menu.h>
```

Öffentliche Attribute

- std::string **name**
- int **id**
- int **position**
- bool **isClickable**
- bool **menuOnEnter**
- gameState **stateOnClick**
- QGraphicsTextItem **showEntry**

5.15.1 Ausführliche Beschreibung

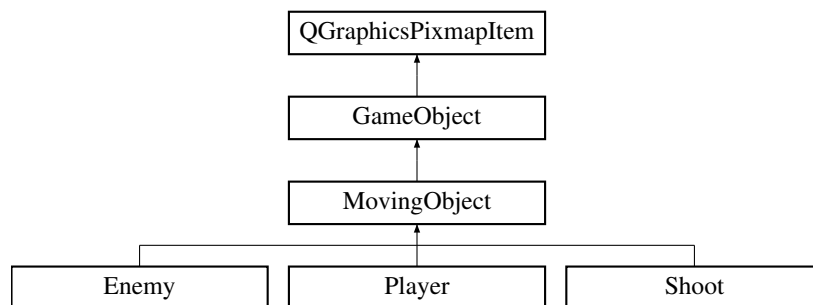
Struct zur Beschreibung eines Menü-Eintrags.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/menu.h

5.16 MovingObject Klassenreferenz

Klassendiagramm für MovingObject:



Öffentliche Methoden

- **MovingObject** (int posX, int posY, objectType type, int speedX, int speedY)
- void **setPosX** (int posX)
- void **setPosY** (int posY)
- int **getSpeedX** () const
- int **getSpeedY** () const
- void **setSpeedX** (int speedX)
- void **setSpeedY** (int speedY)
- void **setFramesDirection** (int framesDirection)
- int **getFramesDirection** ()
- virtual void **update** ()=0
- void **flipHorizontal** ()
spiegelt Grafiken an der Y-Achse
- void **swapImage** ()

Geschützte Methoden

- void **updatePosition** ()
überschreibt die X und Y Position gemäß SpeedXY.

Weitere Geerbte Elemente

5.16.1 Dokumentation der Elementfunktionen

5.16.1.1 void MovingObject::flipHorizontal ()

spiegelt Grafiken an der Y-Achse

Autor

Flo

5.16.1.2 void MovingObject::updatePosition () [protected]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/movingobject.h
- Wiesn-Run/src/movingobject.cpp

5.17 PaDeviceInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- int **structVersion**
- const char * **name**
- [PaHostApiIndex](#) **hostApi**
- int **maxInputChannels**
- int **maxOutputChannels**
- [PaTime](#) **defaultLowInputLatency**
- [PaTime](#) **defaultLowOutputLatency**
- [PaTime](#) **defaultHighInputLatency**
- [PaTime](#) **defaultHighOutputLatency**
- double **defaultSampleRate**

5.17.1 Ausführliche Beschreibung

A structure providing information and capabilities of PortAudio devices. Devices may support input, output or both input and output.

5.17.2 Dokumentation der Datenelemente

5.17.2.1 PaTime PaDeviceInfo::defaultHighInputLatency

Default latency values for robust non-interactive applications (eg. playing sound files).

5.17.2.2 PaTime PaDeviceInfo::defaultLowInputLatency

Default latency values for interactive performance.

5.17.2.3 PaHostApiIndex PaDeviceInfo::hostApi

note this is a host API index, not a type id

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.18 PaHostApiInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- int [structVersion](#)
- [PaHostApiTypeId](#) type
- const char * [name](#)
- int [deviceCount](#)
- [PaDeviceIndex](#) [defaultInputDevice](#)
- [PaDeviceIndex](#) [defaultOutputDevice](#)

5.18.1 Ausführliche Beschreibung

A structure containing information about a particular host API.

5.18.2 Dokumentation der Datenelemente

5.18.2.1 PaDeviceIndex PaHostApiInfo::defaultInputDevice

The default input device for this host API. The value will be a device index ranging from 0 to ([Pa_GetDeviceCount\(\)](#)-1), or [paNoDevice](#) if no default input device is available.

5.18.2.2 PaDeviceIndex PaHostApiInfo::defaultOutputDevice

The default output device for this host API. The value will be a device index ranging from 0 to ([Pa_GetDeviceCount\(\)](#)-1), or [paNoDevice](#) if no default output device is available.

5.18.2.3 int PaHostApiInfo::deviceCount

The number of devices belonging to this host API. This field may be used in conjunction with [Pa_HostApiDeviceIndexToDeviceIndex\(\)](#) to enumerate all devices for this host API.

Siehe auch

[Pa_HostApiDeviceIndexToDeviceIndex](#)

5.18.2.4 `const char* PaHostApiInfo::name`

A textual description of the host API for display on user interfaces.

5.18.2.5 `int PaHostApiInfo::structVersion`

this is struct version 1

5.18.2.6 `PaHostApiTypeId PaHostApiInfo::type`

The well known unique identifier of this host API

Siehe auch

[PaHostApiTypeId](#)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.19 PaHostErrorInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- [PaHostApiTypeId hostApiType](#)
- `long` [errorCode](#)
- `const char *` [errorText](#)

5.19.1 Ausführliche Beschreibung

Structure used to return information about a host error condition.

5.19.2 Dokumentation der Datenelemente

5.19.2.1 `long PaHostErrorInfo::errorCode`

the error code returned

5.19.2.2 `const char* PaHostErrorInfo::errorText`

a textual description of the error if available, otherwise a zero-length string

5.19.2.3 `PaHostApiTypeId PaHostErrorInfo::hostApiType`

the host API which returned the error code

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.20 PaStreamCallbackTimeInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- [PaTime inputBufferAdcTime](#)
- [PaTime currentTime](#)
- [PaTime outputBufferDacTime](#)

5.20.1 Ausführliche Beschreibung

Timing information for the buffers passed to the stream callback.

Time values are expressed in seconds and are synchronised with the time base used by [Pa_GetStreamTime\(\)](#) for the associated stream.

Siehe auch

[PaStreamCallback](#), [Pa_GetStreamTime](#)

5.20.2 Dokumentation der Datenelemente

5.20.2.1 PaTime PaStreamCallbackTimeInfo::currentTime

The time when the stream callback was invoked

5.20.2.2 PaTime PaStreamCallbackTimeInfo::inputBufferAdcTime

The time when the first sample of the input buffer was captured at the ADC input

5.20.2.3 PaTime PaStreamCallbackTimeInfo::outputBufferDacTime

The time when the first sample of the output buffer will output the DAC

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.21 PaStreamInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- int [structVersion](#)
- [PaTime inputLatency](#)
- [PaTime outputLatency](#)
- double [sampleRate](#)

5.21.1 Ausführliche Beschreibung

A structure containing unchanging information about an open stream.

Siehe auch

[Pa_GetStreamInfo](#)

5.21.2 Dokumentation der Datenelemente

5.21.2.1 PaTime PaStreamInfo::inputLatency

The input latency of the stream in seconds. This value provides the most accurate estimate of input latency available to the implementation. It may differ significantly from the suggestedLatency value passed to [Pa_OpenStream\(\)](#). The value of this field will be zero (0.) for output-only streams.

Siehe auch

[PaTime](#)

5.21.2.2 PaTime PaStreamInfo::outputLatency

The output latency of the stream in seconds. This value provides the most accurate estimate of output latency available to the implementation. It may differ significantly from the suggestedLatency value passed to [Pa_OpenStream\(\)](#). The value of this field will be zero (0.) for input-only streams.

Siehe auch

[PaTime](#)

5.21.2.3 double PaStreamInfo::sampleRate

The sample rate of the stream in Hertz (samples per second). In cases where the hardware sample rate is inaccurate and PortAudio is aware of it, the value of this field may be different from the sampleRate parameter passed to [Pa_OpenStream\(\)](#). If information about the actual hardware sample rate is not available, this field will have the same value as the sampleRate parameter passed to [Pa_OpenStream\(\)](#).

5.21.2.4 int PaStreamInfo::structVersion

this is struct version 1

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.22 PaStreamParameters Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- [PaDeviceIndex device](#)

- int [channelCount](#)
- [PaSampleFormat](#) sampleFormat
- [PaTime](#) suggestedLatency
- void * [hostApiSpecificStreamInfo](#)

5.22.1 Ausführliche Beschreibung

Parameters for one direction (input or output) of a stream.

5.22.2 Dokumentation der Datenelemente

5.22.2.1 int PaStreamParameters::channelCount

The number of channels of sound to be delivered to the stream callback or accessed by [Pa_ReadStream\(\)](#) or [Pa_WriteStream\(\)](#). It can range from 1 to the value of maxInputChannels in the [PaDeviceInfo](#) record for the device specified by the device parameter.

5.22.2.2 PaDeviceIndex PaStreamParameters::device

A valid device index in the range 0 to ([Pa_GetDeviceCount\(\)](#)-1) specifying the device to be used or the special constant paUseHostApiSpecificDeviceSpecification which indicates that the actual device(s) to use are specified in hostApiSpecificStreamInfo. This field must not be set to paNoDevice.

5.22.2.3 void* PaStreamParameters::hostApiSpecificStreamInfo

An optional pointer to a host api specific data structure containing additional information for device setup and/or stream processing. hostApiSpecificStreamInfo is never required for correct operation, if not used it should be set to NULL.

5.22.2.4 PaSampleFormat PaStreamParameters::sampleFormat

The sample format of the buffer provided to the stream callback, [a_ReadStream\(\)](#) or [Pa_WriteStream\(\)](#). It may be any of the formats described by the PaSampleFormat enumeration.

5.22.2.5 PaTime PaStreamParameters::suggestedLatency

The desired latency in seconds. Where practical, implementations should configure their latency based on these parameters, otherwise they may choose the closest viable latency instead. Unless the suggested latency is greater than the absolute upper limit for the device implementations should round the suggestedLatency up to the next practical value - ie to provide an equal or higher latency than suggestedLatency wherever possible. Actual latency values for an open stream may be retrieved using the inputLatency and outputLatency fields of the [PaStreamInfo](#) structure returned by [Pa_GetStreamInfo\(\)](#).

Siehe auch

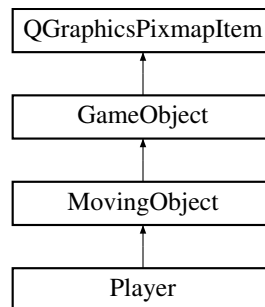
default*Latency in [PaDeviceInfo](#), *Latency in [PaStreamInfo](#)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [Wiesn-Run/src/portaudio.h](#)

5.23 Player Klassenreferenz

Klassendiagramm für Player:



Öffentliche Methoden

- **Player** (int posX, int posY, int speedX)
- int **getHealth** () const
Player::getHealth Gibt aktuellen Lebensstand zurück.
- void **setHealth** (int health)
Player::setHealth Lebensstand des Spielers wird gesetzt.
- void **increaseHealth** (int health)
- bool **receiveDamage** (int damage)
Player::receiveDamage.
- int **getAlcoholLevel** () const
Player::getAlcoholLevel Gibt den Pegel des Spielers zurück.
- void **increaseAlcoholLevel** (int additionalAlcohol)
Player::increaseAlcoholLevel AlkoholPegel wird verändert. Durch einen negativen Wert im Argument wird der Pegel gesenkt.
- void **decreaseAlcoholLevel** (int decreaseLevel)
Player::decreaseAlcoholLevel verringert den Pegel des Spielers.
- int **getAmmunatiuon** () const
Player::getAmmunatiuon Gibt verbleibende Munition zurück.
- void **increaseAmmunation** (int ammunationBonus)
Player::increaseAmmunation erhöht die verbleibende Munition des Spielers um 1.
- void **decreaseAmmunation** ()
Player::decreaseAmmunation verringert die verbleibende Munition des Spielers um 1.
- void **setFireCooldown** ()
- int **getFireCooldown** ()
Player::getFireCooldown.
- int **getInflictedDamage** () const
Player::getInflictedDamage.
- int **getImmunityCooldown** () const
Player::getImmunityCooldown.
- void **setImmunityCooldown** (int remainingTime)
Wird nicht benutzt 23.6.
- void **startJump** ()
beginnt einen Sprung Nur wenn der Spieler sich nicht in der Luft befindet
- bool **inJump** () const
gibt den Sprung-Zustande des Spielers zurück
- void **resetJumpState** ()

Gibt an dass der Spieler nicht in einem Sprung ist.

- void `abortJump` ()

Methode wird aufgerufen, wenn der Spieler bei einem Sprung mit einem Hinderniss zusammengestoßen ist.

- int `getEnemiesKilled` ()

`Player::getEnemiesKilled` Übergibt die Zahl getöteter Gegner.

- void `increaseEnemiesKilled` ()

Perhöht die Anzahl der getöteten Gegner um 1.

- virtual void `update` ()

`Player::update` führt die Bewegung des Spielers aus (über `updatePosition`) und verringert Cooldown-Variable.

Weitere Geerbte Elemente

5.23.1 Beschreibung der Konstruktoren und Destruktoren

5.23.1.1 `Player::Player (int posX, int posY, int speedX)`

Class `Player` lastUpdate: `update()` 10.6 Johann

5.23.2 Dokumentation der Elementfunktionen

5.23.2.1 void `Player::decreaseAlcoholLevel (int decreaseLevel)`

`Player::decreaseAlcoholLevel` verringert den Pegel des Spielers.

Noch zu erledigen Überflüssig, da nie aufgerufen. Auch wenn der Name es nicht vermuten lässt: `increaseAlcoholLevel` kann den Level auch verringern und wird benutzt.

Parameter

<code>decreaseLevel</code>	Wert um den der Pegel verringert wird
----------------------------	---------------------------------------

5.23.2.2 int `Player::getAlcoholLevel ()` const

`Player::getAlcoholLevel` Gibt den Pegel des Spielers zurück.

Rückgabe

: Alkoholpegel

5.23.2.3 int `Player::getAmmunatiuon ()` const

`Player::getAmmunatiuon` Gibt verbleibende Munition zurück.

Rückgabe

: verbleibende Munition

5.23.2.4 int Player::getFireCooldown ()

[Player::getFireCooldown](#).

Rückgabe

verbleibende Zeit bis nächster schuss möglich ist

5.23.2.5 int Player::getHealth () const

[Player::getHealth](#) Gibt aktuellen Lebensstand zurück.

Rückgabe

: Lebensstand

5.23.2.6 int Player::getImmunityCooldown () const

[Player::getImmunityCooldown](#).

Rückgabe

5.23.2.7 int Player::getInflictedDamage () const

[Player::getInflictedDamage](#).

Rückgabe

Schaden den der Spieler zufügt

5.23.2.8 void Player::increaseAlcoholLevel (int *additionalAlcohol*)

[Player::increaseAlcoholLevel](#) AlkoholPegel wird verändert. Durch einen negativen Wert im Argument wird der Pegel gesenkt.

Parameter

<i>additionalAlcohol</i>	Wert um den erhöht wird
--------------------------	-------------------------

5.23.2.9 bool Player::inJump () const

gibt den Sprung-Zustand des Spielers zurück

Rückgabe

5.23.2.10 `bool Player::receiveDamage (int damage)`

[Player::receiveDamage](#).

Rückgabe

Lebenszustand des Spielers: true = tot

5.23.2.11 `void Player::setHealth (int health)`

[Player::setHealth](#) Lebensstand des Spielers wird gesetzt.

Parameter

<i>health</i>	Lebensstand auf den der Spieler gesetzt wird
---------------	--

5.23.2.12 `void Player::setImmunityCooldown (int remainingTime)`

Wird nicht benutzt 23.6.

[Player::setImmunityCooldown](#) Zahl der Frames für Unverwundbarkeit wird gesetzt.

Parameter

<i>immunity-Cooldown</i>	Zahl der Frames
--------------------------	-----------------

5.23.2.13 `void Player::update () [virtual]`

[Player::update](#) führt die Bewegung des Spielers aus (über `updatePosition`) und verringert Cooldown-Variable.

Autor

Johann

Implementiert [MovingObject](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesen-Run/src/player.h
- Wiesen-Run/src/player.cpp

5.24 AudioControl::playStruct Strukturreferenz

Öffentliche Attribute

- int [id](#)
id des [playStruct](#)
- audioType [type](#)
type des [playStruct](#)
- float [volume](#)
Lautstärke des [playStruct](#).
- bool [playnext](#)
variable welche angibt ob sound im moment abgespielt wird

- [Audio](#) * [audioobject](#)

Zeiger auf das (Audio-)object des [playStruct](#), welches Eventgruppe "type" zugeordnet ist.

- int [position](#)

aktuelle Abspielposition in Audiobjekt in Samples (Beginn des Abspielblockes mit Länge 1024 Samples)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

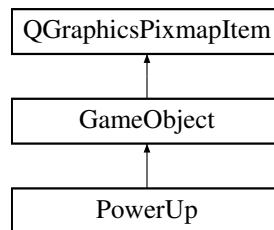
- Wiesn-Run/src/audiocontrol.h

5.25 PowerUp Klassenreferenz

Klasse für Power-Ups.

```
#include <powerup.h>
```

Klassendiagramm für PowerUp:



Öffentliche Methoden

- [PowerUp](#) (int posX, int posY, int healthBonus, int alcoholLevelBonus, int ammurationBonus, int immunity-CooldownBonus)

Konstruktor.

- [~PowerUp](#) ()

Destruktor.

- int [getHealthBonus](#) () const

Get-Methoden für die Objekteigenschaften.

- int [getAlcoholLevelBonus](#) () const

Gibt den Bonus auf Alcohollevel zurück.

- int [getAmmurationBonus](#) () const

Gibt den Bonus auf Munnition zurück.

- int [getImmunityCooldownBonus](#) () const

Gibt den Bonus auf Immunität zurück.

- powerUpType [getPowerUPType](#) () const

[PowerUp::getPowerUPType](#).

Weitere Geerbte Elemente

5.25.1 Ausführliche Beschreibung

Klasse für Power-Ups.

Autor

Johann

5.25.2 Beschreibung der Konstruktoren und Destruktoren

5.25.2.1 PowerUp::PowerUp (int *posX*, int *posY*, int *healthBonus*, int *alcoholLevelBonus*, int *ammunationBonus*, int *immunityCooldownBonus*)

Konstruktor.

Parameter

<i>posX</i>	
<i>posY</i>	
<i>length</i>	
<i>height</i>	
<i>healthBonus</i>	
<i>alcoholLevel-</i> <i>Bonus</i>	
<i>ammunation-</i> <i>Bonus</i>	
<i>immunity-</i> <i>CooldownBonus</i>	

Autor

Johann

5.25.2.2 PowerUp::~~PowerUp ()

Destruktor.

Autor

Johann

5.25.3 Dokumentation der Elementfunktionen

5.25.3.1 int PowerUp::getAlcoholLevelBonus () const

Gibt den Bonus auf Alcohollevel zurück.

Autor

Johann

5.25.3.2 int PowerUp::getAmmunationBonus () const

Gibt den Bonus auf Munnition zurück.

Autor

Johann

5.25.3.3 int PowerUp::getHealthBonus () const

Get-Methoden für die Objekteigenschaften.

Gibt den Bonus auf Leben zurück.

Autor

Johann

5.25.3.4 `int PowerUp::getImmunityCooldownBonus () const`

Gibt den Bonus auf Immunität zurück.

Autor

Johann

5.25.3.5 `powerUpType PowerUp::getPowerUPType () const`

[PowerUp::getPowerUPType.](#)

Rückgabe

Art des powerups

Autor

Johann

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/powerup.h
- Wiesn-Run/src/powerup.cpp

5.26 scoreStruct Strukturreferenz

Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. Alkohol-Punkte erhält der Spieler für einen gewissen Pegel in einem Zeitabschnitt.

```
#include <definitions.h>
```

Öffentliche Attribute

- `std::string name`
- `int totalPoints`
- `int distanceCovered`
- `int alcoholPoints`
- `int enemiesKilled`

5.26.1 Ausführliche Beschreibung

Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. Alkohol-Punkte erhält der Spieler für einen gewissen Pegel in einem Zeitabschnitt.

Noch zu erledigen Das Konzept der Alkohol-Punkte muss noch ausgearbeitet werden.

Autor

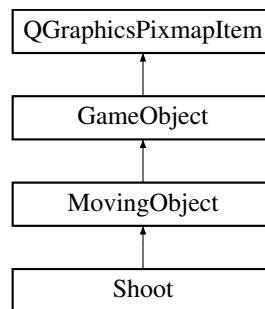
Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.27 Shoot Klassenreferenz

Klassendiagramm für Shoot:



Öffentliche Methoden

- **Shoot** (int posX, int posY, int direction, objectType origin)
Konstruktor für einen Schuss(Bierkrug)
- int **getInflictedDamage** () const
Shoot::getInflictedDamage gibt den Schaden den der Schuss zufügt zurück.
- objectType **getOrigin** ()
Shoot::getOrigin gibt den Ursprung des Bierkrugs zurück, Wer hat ihn geworfen (Player/Enemy)
- virtual void **update** ()

Weitere Geerbte Elemente

5.27.1 Beschreibung der Konstruktoren und Destruktoren

5.27.1.1 Shoot::Shoot (int posX, int posY, int direction, objectType origin)

Konstruktor für einen Schuss(Bierkrug)

Parameter

<i>posX</i>	: x-Position
<i>posY</i>	: y-Position
<i>origin</i>	: Schuss Erzeuger

Schuss bewegt sich dreimal so schnell wie der spieler Größe des Bierkruges festgesetzt (erste idee)

Autor

Johann

5.27.2 Dokumentation der Elementfunktionen

5.27.2.1 `int Shoot::getInflictedDamage () const`

`Shoot::getInflictedDamage` gibt den Schaden den der Schuss zufügt zurück.

Rückgabe

Schaden

Autor

Johann

5.27.2.2 `objectType Shoot::getOrigin ()`

`Shoot::getOrigin` gibt den Ursprung des Bierkrugs zurück, Wer hat ihn geworfen (Player/Enemy)

Rückgabe

Ursprung des Bierkruges

Autor

Johann

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/shoot.h
- Wiesn-Run/src/shoot.cpp

5.28 `stateStruct` Strukturreferenz

Struktur für die States des Spiels Sowohl Sound- als auch Grafik-Ausgabe erhalten aus den States Informationen darüber, was gerade im Spiel passiert, z.B. dass gerade der Spieler angreift, ein Gegner stirbt etc.

```
#include <definitions.h>
```

Öffentliche Attribute

- `bool gameOver`
- `int actLevel`
- `int audioID_Background`
- `bool playerJumping`
- `bool playerAttacking`
- `bool playerRunning`
- `bool playerThrowing`
- `bool playerHit`
- `bool enemyAttacking`
- `bool enemyThrowing`
- `bool enemyDead`
- `bool beerCollected`
- `bool chickenCollected`

5.28.1 Ausführliche Beschreibung

Struktur für die States des Spiels Sowohl Sound- als auch Grafik-Ausgabe erhalten aus den States Informationen darüber, was gerade im Spiel passiert, z.B. dass gerade der Spieler angreift, ein Gegner stirbt etc.

Noch zu erledigen Diese Struktur ist vermutlich überflüssig.

Autor

Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

Kapitel 6

Datei-Dokumentation

6.1 Wiesn-Run/src/portaudio.h-Dateireferenz

The portable PortAudio API.

Klassen

- struct [PaHostApiInfo](#)
- struct [PaHostErrorInfo](#)
- struct [PaDeviceInfo](#)
- struct [PaStreamParameters](#)
- struct [PaStreamCallbackTimeInfo](#)
- struct [PaStreamInfo](#)

Makrodefinitionen

- #define [paNoDevice](#) (([PaDeviceIndex](#))-1)
- #define [paUseHostApiSpecificDeviceSpecification](#) (([PaDeviceIndex](#))-2)
- #define [paFloat32](#) (([PaSampleFormat](#)) 0x00000001)
- #define [paInt32](#) (([PaSampleFormat](#)) 0x00000002)
- #define [paInt24](#) (([PaSampleFormat](#)) 0x00000004)
- #define [paInt16](#) (([PaSampleFormat](#)) 0x00000008)
- #define [paInt8](#) (([PaSampleFormat](#)) 0x00000010)
- #define [paUInt8](#) (([PaSampleFormat](#)) 0x00000020)
- #define [paCustomFormat](#) (([PaSampleFormat](#)) 0x00010000)
- #define [paNonInterleaved](#) (([PaSampleFormat](#)) 0x80000000)
- #define [paFormatIsSupported](#) (0)
- #define [paFramesPerBufferUnspecified](#) (0)
- #define [paNoFlag](#) (([PaStreamFlags](#)) 0)
- #define [paClipOff](#) (([PaStreamFlags](#)) 0x00000001)
- #define [paDitherOff](#) (([PaStreamFlags](#)) 0x00000002)
- #define [paNeverDropInput](#) (([PaStreamFlags](#)) 0x00000004)
- #define [paPrimeOutputBuffersUsingStreamCallback](#) (([PaStreamFlags](#)) 0x00000008)
- #define [paPlatformSpecificFlags](#) (([PaStreamFlags](#)) 0xFFFF0000)
- #define [paInputUnderflow](#) (([PaStreamCallbackFlags](#)) 0x00000001)
- #define [paInputOverflow](#) (([PaStreamCallbackFlags](#)) 0x00000002)
- #define [paOutputUnderflow](#) (([PaStreamCallbackFlags](#)) 0x00000004)
- #define [paOutputOverflow](#) (([PaStreamCallbackFlags](#)) 0x00000008)
- #define [paPrimingOutput](#) (([PaStreamCallbackFlags](#)) 0x00000010)

Typdefinitionen

- typedef int [PaError](#)
- typedef enum PaErrorCode **PaErrorCode**
- typedef int [PaDeviceIndex](#)
- typedef int [PaHostApiIndex](#)
- typedef enum [PaHostApiTypeId](#) [PaHostApiTypeId](#)
- typedef struct [PaHostApiInfo](#) [PaHostApiInfo](#)
- typedef struct [PaHostErrorInfo](#) [PaHostErrorInfo](#)
- typedef double [PaTime](#)
- typedef unsigned long [PaSampleFormat](#)
- typedef struct [PaDeviceInfo](#) [PaDeviceInfo](#)
- typedef struct [PaStreamParameters](#) [PaStreamParameters](#)
- typedef void [PaStream](#)
- typedef unsigned long [PaStreamFlags](#)
- typedef struct [PaStreamCallbackTimeInfo](#) [PaStreamCallbackTimeInfo](#)
- typedef unsigned long [PaStreamCallbackFlags](#)
- typedef enum [PaStreamCallbackResult](#) [PaStreamCallbackResult](#)
- typedef int [PaStreamCallback](#) (const void *input, void *output, unsigned long frameCount, const [PaStreamCallbackTimeInfo](#) *timeInfo, [PaStreamCallbackFlags](#) statusFlags, void *userData)
- typedef void [PaStreamFinishedCallback](#) (void *userData)
- typedef struct [PaStreamInfo](#) [PaStreamInfo](#)

Aufzählungen

- enum **PaErrorCode** {
paNoError = 0, **paNotInitialized** = -10000, **paUnanticipatedHostError**, **paInvalidChannelCount**,
paInvalidSampleRate, **paInvalidDevice**, **paInvalidFlag**, **paSampleFormatNotSupported**,
paBadIODeviceCombination, **paInsufficientMemory**, **paBufferTooBig**, **paBufferTooSmall**,
paNullCallback, **paBadStreamPtr**, **paTimedOut**, **paInternalError**,
paDeviceUnavailable, **paIncompatibleHostApiSpecificStreamInfo**, **paStreamIsStopped**, **paStreamIsNotStopped**,
paInputOverflowed, **paOutputUnderflowed**, **paHostApiNotFound**, **paInvalidHostApi**,
paCanNotReadFromACallbackStream, **paCanNotWriteToACallbackStream**, **paCanNotReadFromAnOutputOnlyStream**,
paCanNotWriteToAnInputOnlyStream, **paIncompatibleStreamHostApi**, **paBadBufferPtr** }
- enum [PaHostApiTypeId](#) {
paInDevelopment = 0, **paDirectSound** = 1, **paMME** = 2, **paASIO** = 3,
paSoundManager = 4, **paCoreAudio** = 5, **paOSS** = 7, **paALSA** = 8,
paAL = 9, **paBeOS** = 10, **paWDMKS** = 11, **paJACK** = 12,
paWASAPI = 13, **paAudioScienceHPI** = 14 }
- enum [PaStreamCallbackResult](#) { [paContinue](#) = 0, [paComplete](#) = 1, [paAbort](#) = 2 }

Funktionen

- int [Pa_GetVersion](#) (void)
- const char * [Pa_GetVersionText](#) (void)
- const char * [Pa_GetErrorText](#) ([PaError](#) errorCode)
- [PaError](#) [Pa_Initialize](#) (void)
- [PaError](#) [Pa_Terminate](#) (void)
- [PaHostApiIndex](#) [Pa_GetHostApiCount](#) (void)
- [PaHostApiIndex](#) [Pa_GetDefaultHostApi](#) (void)
- const [PaHostApiInfo](#) * [Pa_GetHostApiInfo](#) ([PaHostApiIndex](#) hostApi)
- [PaHostApiIndex](#) [Pa_HostApiTypeIdToHostApiIndex](#) ([PaHostApiTypeId](#) type)

- [PaDeviceIndex](#) [Pa_HostApiDeviceIndexToDeviceIndex](#) ([PaHostApiIndex](#) hostApi, int hostApiDeviceIndex)
- const [PaHostErrorInfo](#) * [Pa_GetLastHostErrorInfo](#) (void)
- [PaDeviceIndex](#) [Pa_GetDeviceCount](#) (void)
- [PaDeviceIndex](#) [Pa_GetDefaultInputDevice](#) (void)
- [PaDeviceIndex](#) [Pa_GetDefaultOutputDevice](#) (void)
- const [PaDeviceInfo](#) * [Pa_GetDeviceInfo](#) ([PaDeviceIndex](#) device)
- [PaError](#) [Pa_IsFormatSupported](#) (const [PaStreamParameters](#) *inputParameters, const [PaStreamParameters](#) *outputParameters, double sampleRate)
- [PaError](#) [Pa_OpenStream](#) ([PaStream](#) **stream, const [PaStreamParameters](#) *inputParameters, const [PaStreamParameters](#) *outputParameters, double sampleRate, unsigned long framesPerBuffer, [PaStreamFlags](#) streamFlags, [PaStreamCallback](#) *streamCallback, void *userData)
- [PaError](#) [Pa_OpenDefaultStream](#) ([PaStream](#) **stream, int numInputChannels, int numOutputChannels, [PaSampleFormat](#) sampleFormat, double sampleRate, unsigned long framesPerBuffer, [PaStreamCallback](#) *streamCallback, void *userData)
- [PaError](#) [Pa_CloseStream](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_SetStreamFinishedCallback](#) ([PaStream](#) *stream, [PaStreamFinishedCallback](#) *streamFinishedCallback)
- [PaError](#) [Pa_StartStream](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_StopStream](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_AbortStream](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_IsStreamStopped](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_IsStreamActive](#) ([PaStream](#) *stream)
- const [PaStreamInfo](#) * [Pa_GetStreamInfo](#) ([PaStream](#) *stream)
- [PaTime](#) [Pa_GetStreamTime](#) ([PaStream](#) *stream)
- double [Pa_GetStreamCpuLoad](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_ReadStream](#) ([PaStream](#) *stream, void *buffer, unsigned long frames)
- [PaError](#) [Pa_WriteStream](#) ([PaStream](#) *stream, const void *buffer, unsigned long frames)
- signed long [Pa_GetStreamReadAvailable](#) ([PaStream](#) *stream)
- signed long [Pa_GetStreamWriteAvailable](#) ([PaStream](#) *stream)
- [PaHostApiTypeId](#) [Pa_GetStreamHostApiType](#) ([PaStream](#) *stream)
- [PaError](#) [Pa_GetSampleSize](#) ([PaSampleFormat](#) format)
- void [Pa_Sleep](#) (long msec)

6.1.1 Ausführliche Beschreibung

The portable PortAudio API.

6.1.2 Makro-Dokumentation

6.1.2.1 #define paClipOff ((PaStreamFlags) 0x00000001)

Disable default clipping of out of range samples.

Siehe auch

[PaStreamFlags](#)

6.1.2.2 #define paCustomFormat ((PaSampleFormat) 0x00010000)

Siehe auch

[PaSampleFormat](#)

6.1.2.3 #define paDitherOff ((PaStreamFlags) 0x00000002)

Disable default dithering.

Siehe auch

[PaStreamFlags](#)

6.1.2.4 #define paFloat32 ((PaSampleFormat) 0x00000001)

Siehe auch

[PaSampleFormat](#)

6.1.2.5 #define paFormatIsSupported (0)

Return code for Pa_IsFormatSupported indicating success.

6.1.2.6 #define paFramesPerBufferUnspecified (0)

Can be passed as the framesPerBuffer parameter to [Pa_OpenStream\(\)](#) or [Pa_OpenDefaultStream\(\)](#) to indicate that the stream callback will accept buffers of any size.

6.1.2.7 #define paInputOverflow ((PaStreamCallbackFlags) 0x00000002)

In a stream opened with paFramesPerBufferUnspecified, indicates that data prior to the first sample of the input buffer was discarded due to an overflow, possibly because the stream callback is using too much CPU time. Otherwise indicates that data prior to one or more samples in the input buffer was discarded.

Siehe auch

[PaStreamCallbackFlags](#)

6.1.2.8 #define paInputUnderflow ((PaStreamCallbackFlags) 0x00000001)

In a stream opened with paFramesPerBufferUnspecified, indicates that input data is all silence (zeros) because no real data is available. In a stream opened without paFramesPerBufferUnspecified, it indicates that one or more zero samples have been inserted into the input buffer to compensate for an input underflow.

Siehe auch

[PaStreamCallbackFlags](#)

6.1.2.9 #define paInt16 ((PaSampleFormat) 0x00000008)

Siehe auch

[PaSampleFormat](#)

6.1.2.10 `#define paInt24 ((PaSampleFormat) 0x00000004)`

Packed 24 bit format.

Siehe auch

[PaSampleFormat](#)

6.1.2.11 `#define paInt32 ((PaSampleFormat) 0x00000002)`

Siehe auch

[PaSampleFormat](#)

6.1.2.12 `#define paInt8 ((PaSampleFormat) 0x00000010)`

Siehe auch

[PaSampleFormat](#)

6.1.2.13 `#define paNeverDropInput ((PaStreamFlags) 0x00000004)`

Flag requests that where possible a full duplex stream will not discard overflowed input samples without calling the stream callback. This flag is only valid for full duplex callback streams and only when used in combination with the `paFramesPerBufferUnspecified` (0) `framesPerBuffer` parameter. Using this flag incorrectly results in a `paInvalidFlag` error being returned from `Pa_OpenStream` and `Pa_OpenDefaultStream`.

Siehe auch

[PaStreamFlags](#), [paFramesPerBufferUnspecified](#)

6.1.2.14 `#define paNoDevice ((PaDeviceIndex)-1)`

A special `PaDeviceIndex` value indicating that no device is available, or should be used.

Siehe auch

[PaDeviceIndex](#)

6.1.2.15 `#define paNoFlag ((PaStreamFlags) 0)`

Siehe auch

[PaStreamFlags](#)

6.1.2.16 `#define paNonInterleaved ((PaSampleFormat) 0x80000000)`

Siehe auch

[PaSampleFormat](#)

6.1.2.17 #define paOutputOverflow ((PaStreamCallbackFlags) 0x00000008)

Indicates that output data will be discarded because no room is available.

Siehe auch

[PaStreamCallbackFlags](#)

6.1.2.18 #define paOutputUnderflow ((PaStreamCallbackFlags) 0x00000004)

Indicates that output data (or a gap) was inserted, possibly because the stream callback is using too much CPU time.

Siehe auch

[PaStreamCallbackFlags](#)

6.1.2.19 #define paPlatformSpecificFlags ((PaStreamFlags)0xFFFF0000)

A mask specifying the platform specific bits.

Siehe auch

[PaStreamFlags](#)

6.1.2.20 #define paPrimeOutputBuffersUsingStreamCallback ((PaStreamFlags) 0x00000008)

Call the stream callback to fill initial output buffers, rather than the default behavior of priming the buffers with zeros (silence). This flag has no effect for input-only and blocking read/write streams.

Siehe auch

[PaStreamFlags](#)

6.1.2.21 #define paPrimingOutput ((PaStreamCallbackFlags) 0x00000010)

Some of all of the output data will be used to prime the stream, input data may be zero.

Siehe auch

[PaStreamCallbackFlags](#)

6.1.2.22 #define paUInt8 ((PaSampleFormat) 0x00000020)

Siehe auch

[PaSampleFormat](#)

6.1.2.23 #define paUseHostApiSpecificDeviceSpecification ((PaDeviceIndex)-2)

A special PaDeviceIndex value indicating that the device(s) to be used are specified in the host api specific stream info structure.

Siehe auch

[PaDeviceIndex](#)

6.1.3 Dokumentation der benutzerdefinierten Typen

6.1.3.1 typedef int PaDeviceIndex

The type used to refer to audio devices. Values of this type usually range from 0 to ([Pa_GetDeviceCount\(\)](#)-1), and may also take on the PaNoDevice and paUseHostApiSpecificDeviceSpecification values.

Siehe auch

[Pa_GetDeviceCount](#), [paNoDevice](#), [paUseHostApiSpecificDeviceSpecification](#)

6.1.3.2 typedef struct PaDeviceInfo PaDeviceInfo

A structure providing information and capabilities of PortAudio devices. Devices may support input, output or both input and output.

6.1.3.3 typedef int PaError

Error codes returned by PortAudio functions. Note that with the exception of paNoError, all PaErrorCodes are negative.

6.1.3.4 typedef int PaHostApiIndex

The type used to enumerate to host APIs at runtime. Values of this type range from 0 to ([Pa_GetHostApiCount\(\)](#)-1).

Siehe auch

[Pa_GetHostApiCount](#)

6.1.3.5 typedef struct PaHostApiInfo PaHostApiInfo

A structure containing information about a particular host API.

6.1.3.6 typedef enum PaHostApiTypeId PaHostApiTypeId

Unchanging unique identifiers for each supported host API. This type is used in the [PaHostApiInfo](#) structure. The values are guaranteed to be unique and to never change, thus allowing code to be written that conditionally uses host API specific extensions.

New type ids will be allocated when support for a host API reaches "public alpha" status, prior to that developers should use the paInDevelopment type id.

Siehe auch

[PaHostApiInfo](#)

6.1.3.7 typedef struct PaHostErrorInfo PaHostErrorInfo

Structure used to return information about a host error condition.

6.1.3.8 typedef unsigned long PaSampleFormat

A type used to specify one or more sample formats. Each value indicates a possible format for sound data passed to and from the stream callback, `Pa_ReadStream` and `Pa_WriteStream`.

The standard formats `paFloat32`, `paInt16`, `paInt32`, `paInt24`, `paInt8` and `aUInt8` are usually implemented by all implementations.

The floating point representation (`paFloat32`) uses +1.0 and -1.0 as the maximum and minimum respectively.

`paUInt8` is an unsigned 8 bit format where 128 is considered "ground"

The `paNonInterleaved` flag indicates that audio data is passed as an array of pointers to separate buffers, one buffer for each channel. Usually, when this flag is not used, audio data is passed as a single buffer with all channels interleaved.

Siehe auch

[Pa_OpenStream](#), [Pa_OpenDefaultStream](#), [PaDeviceInfo](#)
[paFloat32](#), [paInt16](#), [paInt32](#), [paInt24](#), [paInt8](#)
[paUInt8](#), [paCustomFormat](#), [paNonInterleaved](#)

6.1.3.9 typedef void PaStream

A single `PaStream` can provide multiple channels of real-time streaming audio input and output to a client application. A stream provides access to audio hardware represented by one or more `PaDevices`. Depending on the underlying Host API, it may be possible to open multiple streams using the same device, however this behavior is implementation defined. Portable applications should assume that a `PaDevice` may be simultaneously used by at most one `PaStream`.

Pointers to `PaStream` objects are passed between PortAudio functions that operate on streams.

Siehe auch

[Pa_OpenStream](#), [Pa_OpenDefaultStream](#), [Pa_OpenDefaultStream](#), [Pa_CloseStream](#), [Pa_StartStream](#), [Pa_StopStream](#), [Pa_AbortStream](#), [Pa_IsStreamActive](#), [Pa_GetStreamTime](#), [Pa_GetStreamCpuLoad](#)

6.1.3.10 typedef int PaStreamCallback(const void *input, void *output, unsigned long frameCount, const PaStreamCallbackTimeInfo *timeInfo, PaStreamCallbackFlags statusFlags, void *userData)

Functions of type `PaStreamCallback` are implemented by PortAudio clients. They consume, process or generate audio in response to requests from an active PortAudio stream.

When a stream is running, PortAudio calls the stream callback periodically. The callback function is responsible for processing buffers of audio samples passed via the input and output parameters.

The PortAudio stream callback runs at very high or real-time priority. It is required to consistently meet its time deadlines. Do not allocate memory, access the file system, call library functions or call other functions from the stream callback that may block or take an unpredictable amount of time to complete.

In order for a stream to maintain glitch-free operation the callback must consume and return audio data faster than it is recorded and/or played. PortAudio anticipates that each callback invocation may execute for a duration approaching the duration of `frameCount` audio frames at the stream sample rate. It is reasonable to expect to be able to utilise 70% or more of the available CPU time in the PortAudio callback. However, due to buffer size adaption and other factors, not all host APIs are able to guarantee audio stability under heavy CPU load with arbitrary fixed callback buffer sizes. When high callback CPU utilisation is required the most robust behavior can be achieved by using `paFramesPerBufferUnspecified` as the [Pa_OpenStream\(\)](#) `framesPerBuffer` parameter.

Parameter

<i>input</i>	and
<i>output</i>	are either arrays of interleaved samples or; if non-interleaved samples were requested using the <code>paNonInterleaved</code> sample format flag, an array of buffer pointers, one non-interleaved buffer for each channel.

The format, packing and number of channels used by the buffers are determined by parameters to [Pa_OpenStream\(\)](#).

Parameter

<i>frameCount</i>	The number of sample frames to be processed by the stream callback.
<i>timeInfo</i>	Timestamps indicating the ADC capture time of the first sample in the input buffer, the DAC output time of the first sample in the output buffer and the time the callback was invoked. See PaStreamCallbackTimeInfo and Pa_GetStreamTime()
<i>statusFlags</i>	Flags indicating whether input and/or output buffers have been inserted or will be dropped to overcome underflow or overflow conditions.
<i>userData</i>	The value of a user supplied pointer passed to Pa_OpenStream() intended for storing synthesis data etc.

Rückgabe

The stream callback should return one of the values in the [PaStreamCallbackResult](#) enumeration. To ensure that the callback continues to be called, it should return `paContinue` (0). Either `paComplete` or `paAbort` can be returned to finish stream processing, after either of these values is returned the callback will not be called again. If `paAbort` is returned the stream will finish as soon as possible. If `paComplete` is returned, the stream will continue until all buffers generated by the callback have been played. This may be useful in applications such as soundfile players where a specific duration of output is required. However, it is not necessary to utilize this mechanism as [Pa_StopStream\(\)](#), [Pa_AbortStream\(\)](#) or [Pa_CloseStream\(\)](#) can also be used to stop the stream. The callback must always fill the entire output buffer irrespective of its return value.

Siehe auch

[Pa_OpenStream](#), [Pa_OpenDefaultStream](#)

Zu beachten

With the exception of [Pa_GetStreamCpuLoad\(\)](#) it is not permissible to call PortAudio API functions from within the stream callback.

6.1.3.11 typedef unsigned long PaStreamCallbackFlags

Flag bit constants for the `statusFlags` to `PaStreamCallback`.

Siehe auch

[paInputUnderflow](#), [paInputOverflow](#), [paOutputUnderflow](#), [paOutputOverflow](#), [paPrimingOutput](#)

6.1.3.12 typedef enum PaStreamCallbackResult PaStreamCallbackResult

Allowable return values for the `PaStreamCallback`.

Siehe auch

[PaStreamCallback](#)

6.1.3.13 typedef struct PaStreamCallbackTimeInfo PaStreamCallbackTimeInfo

Timing information for the buffers passed to the stream callback.

Time values are expressed in seconds and are synchronised with the time base used by [Pa_GetStreamTime\(\)](#) for the associated stream.

Siehe auch

[PaStreamCallback](#), [Pa_GetStreamTime](#)

6.1.3.14 typedef void PaStreamFinishedCallback(void *userData)

Functions of type PaStreamFinishedCallback are implemented by PortAudio clients. They can be registered with a stream using the Pa_SetStreamFinishedCallback function. Once registered they are called when the stream becomes inactive (ie once a call to [Pa_StopStream\(\)](#) will not block). A stream will become inactive after the stream callback returns non-zero, or when Pa_StopStream or Pa_AbortStream is called. For a stream providing audio output, if the stream callback returns paComplete, or Pa_StopStream is called, the stream finished callback will not be called until all generated sample data has been played.

Parameter

<i>userData</i>	The userData parameter supplied to Pa_OpenStream()
-----------------	--

Siehe auch

[Pa_SetStreamFinishedCallback](#)

6.1.3.15 typedef unsigned long PaStreamFlags

Flags used to control the behavior of a stream. They are passed as parameters to Pa_OpenStream or Pa_OpenDefaultStream. Multiple flags may be ORed together.

Siehe auch

[Pa_OpenStream](#), [Pa_OpenDefaultStream](#)
[paNoFlag](#), [paClipOff](#), [paDitherOff](#), [paNeverDropInput](#), [paPrimeOutputBuffersUsingStreamCallback](#), [pa-PlatformSpecificFlags](#)

6.1.3.16 typedef struct PaStreamInfo PaStreamInfo

A structure containing unchanging information about an open stream.

Siehe auch

[Pa_GetStreamInfo](#)

6.1.3.17 typedef struct PaStreamParameters PaStreamParameters

Parameters for one direction (input or output) of a stream.

6.1.3.18 typedef double PaTime

The type used to represent monotonic time in seconds. PaTime is used for the fields of the [PaStreamCallbackTimeInfo](#) argument to the PaStreamCallback and as the result of [Pa_GetStreamTime\(\)](#).

PaTime values have unspecified origin.

Siehe auch

[PaStreamCallback](#), [PaStreamCallbackTimeInfo](#), [Pa_GetStreamTime](#)

6.1.4 Dokumentation der Aufzählungstypen

6.1.4.1 enum PaHostApiTypeId

Unchanging unique identifiers for each supported host API. This type is used in the [PaHostApiInfo](#) structure. The values are guaranteed to be unique and to never change, thus allowing code to be written that conditionally uses host API specific extensions.

New type ids will be allocated when support for a host API reaches "public alpha" status, prior to that developers should use the paInDevelopment type id.

Siehe auch

[PaHostApiInfo](#)

6.1.4.2 enum PaStreamCallbackResult

Allowable return values for the PaStreamCallback.

Siehe auch

[PaStreamCallback](#)

Aufzählungswerte

paContinue Signal that the stream should continue invoking the callback and processing audio.

paComplete Signal that the stream should stop invoking the callback and finish once all output samples have played.

paAbort Signal that the stream should stop invoking the callback and finish as soon as possible.

6.1.5 Dokumentation der Funktionen

6.1.5.1 PaError Pa_AbortStream (PaStream * *stream*)

Terminates audio processing immediately without waiting for pending buffers to complete.

6.1.5.2 PaError Pa_CloseStream (PaStream * *stream*)

Closes an audio stream. If the audio stream is active it discards any pending buffers as if [Pa_AbortStream\(\)](#) had been called.

6.1.5.3 PaHostApiIndex Pa_GetDefaultHostApi (void)

Retrieve the index of the default host API. The default host API will be the lowest common denominator host API on the current platform and is unlikely to provide the best performance.

Rückgabe

A non-negative value ranging from 0 to ([Pa_GetHostApiCount\(\)](#)-1) indicating the default host API index or, a [PaErrorCode](#) (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.4 [PaDeviceIndex](#) [Pa_GetDefaultInputDevice](#) (void)

Retrieve the index of the default input device. The result can be used in the `inputDevice` parameter to [Pa_OpenStream\(\)](#).

Rückgabe

The default input device index for the default host API, or `paNoDevice` if no default input device is available or an error was encountered.

6.1.5.5 [PaDeviceIndex](#) [Pa_GetDefaultOutputDevice](#) (void)

Retrieve the index of the default output device. The result can be used in the `outputDevice` parameter to [Pa_OpenStream\(\)](#).

Rückgabe

The default output device index for the default host API, or `paNoDevice` if no default output device is available or an error was encountered.

Zu beachten

On the PC, the user can specify a default device by setting an environment variable. For example, to use device #1.

```
set PA_RECOMMENDED_OUTPUT_DEVICE=1
```

The user should first determine the available device ids by using the supplied application "pa_devs".

6.1.5.6 [PaDeviceIndex](#) [Pa_GetDeviceCount](#) (void)

Retrieve the number of available devices. The number of available devices may be zero.

Rückgabe

A non-negative value indicating the number of available devices or, a [PaErrorCode](#) (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.7 [const PaDeviceInfo*](#) [Pa_GetDeviceInfo](#) ([PaDeviceIndex](#) *device*)

Retrieve a pointer to a [PaDeviceInfo](#) structure containing information about the specified device.

Rückgabe

A pointer to an immutable [PaDeviceInfo](#) structure. If the device parameter is out of range the function returns NULL.

Parameter

<i>device</i>	A valid device index in the range 0 to (Pa_GetDeviceCount() -1)
---------------	--

Zu beachten

PortAudio manages the memory referenced by the returned pointer, the client must not manipulate or free the memory. The pointer is only guaranteed to be valid between calls to [Pa_Initialize\(\)](#) and [Pa_Terminate\(\)](#).

Siehe auch

[PaDeviceInfo](#), [PaDeviceIndex](#)

6.1.5.8 `const char* Pa_GetErrorText (PaError errorCode)`

Translate the supplied PortAudio error code into a human readable message.

6.1.5.9 `PaHostApiIndex Pa_GetHostApiCount (void)`

Retrieve the number of available host APIs. Even if a host API is available it may have no devices available.

Rückgabe

A non-negative value indicating the number of available host APIs or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

[PaHostApiIndex](#)

6.1.5.10 `const PaHostApiInfo* Pa_GetHostApiInfo (PaHostApiIndex hostApi)`

Retrieve a pointer to a structure containing information about a specific host Api.

Parameter

<i>hostApi</i>	A valid host API index ranging from 0 to (Pa_GetHostApiCount() -1)
----------------	---

Rückgabe

A pointer to an immutable [PaHostApiInfo](#) structure describing a specific host API. If the `hostApi` parameter is out of range or an error is encountered, the function returns `NULL`.

The returned structure is owned by the PortAudio implementation and must not be manipulated or freed. The pointer is only guaranteed to be valid between calls to [Pa_Initialize\(\)](#) and [Pa_Terminate\(\)](#).

6.1.5.11 `const PaHostErrorInfo* Pa_GetLastHostErrorInfo (void)`

Return information about the last host error encountered. The error information returned by [Pa_GetLastHostErrorInfo\(\)](#) will never be modified asynchronously by errors occurring in other PortAudio owned threads (such as the thread that manages the stream callback.)

This function is provided as a last resort, primarily to enhance debugging by providing clients with access to all available error information.

Rückgabe

A pointer to an immutable structure constraining information about the host error. The values in this structure will only be valid if a PortAudio function has previously returned the `paUnanticipatedHostError` error code.

6.1.5.12 **PaError** Pa_GetSampleSize (**PaSampleFormat** *format*)

Retrieve the size of a given sample format in bytes.

Rückgabe

The size in bytes of a single sample in the specified format, or `paSampleFormatNotSupported` if the format is not supported.

6.1.5.13 **double** Pa_GetStreamCpuLoad (**PaStream** * *stream*)

Retrieve CPU usage information for the specified stream. The "CPU Load" is a fraction of total CPU time consumed by a callback stream's audio processing routines including, but not limited to the client supplied stream callback. This function does not work with blocking read/write streams.

This function may be called from the stream callback function or the application.

Rückgabe

A floating point value, typically between 0.0 and 1.0, where 1.0 indicates that the stream callback is consuming the maximum number of CPU cycles possible to maintain real-time operation. A value of 0.5 would imply that PortAudio and the stream callback was consuming roughly 50% of the available CPU time. The return value may exceed 1.0. A value of 0.0 will always be returned for a blocking read/write stream, or if an error occurs.

6.1.5.14 **PaHostApiTypeId** Pa_GetStreamHostApiType (**PaStream** * *stream*)

Retrieve the host type handling an open stream.

Rückgabe

Returns a non-negative value representing the host API type handling an open stream or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.15 **const PaStreamInfo*** Pa_GetStreamInfo (**PaStream** * *stream*)

Retrieve a pointer to a [PaStreamInfo](#) structure containing information about the specified stream.

Rückgabe

A pointer to an immutable [PaStreamInfo](#) structure. If the stream parameter invalid, or an error is encountered, the function returns NULL.

Parameter

<i>stream</i>	A pointer to an open stream previously created with <code>Pa_OpenStream</code> .
---------------	--

Zu beachten

PortAudio manages the memory referenced by the returned pointer, the client must not manipulate or free the memory. The pointer is only guaranteed to be valid until the specified stream is closed.

Siehe auch

[PaStreamInfo](#)

6.1.5.16 signed long Pa_GetStreamReadAvailable (PaStream * stream)

Retrieve the number of frames that can be read from the stream without waiting.

Rückgabe

Returns a non-negative value representing the maximum number of frames that can be read from the stream without blocking or busy waiting or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.17 PaTime Pa_GetStreamTime (PaStream * stream)

Returns the current time in seconds for a stream according to the same clock used to generate callback [PaStreamCallbackTimeInfo](#) timestamps. The time values are monotonically increasing and have unspecified origin.

Pa_GetStreamTime returns valid time values for the entire life of the stream, from when the stream is opened until it is closed. Starting and stopping the stream does not affect the passage of time returned by Pa_GetStreamTime.

This time may be used for synchronizing other events to the audio stream, for example synchronizing audio to MIDI.

Rückgabe

The stream's current time in seconds, or 0 if an error occurred.

Siehe auch

[PaTime](#), [PaStreamCallback](#), [PaStreamCallbackTimeInfo](#)

6.1.5.18 signed long Pa_GetStreamWriteAvailable (PaStream * stream)

Retrieve the number of frames that can be written to the stream without waiting.

Rückgabe

Returns a non-negative value representing the maximum number of frames that can be written to the stream without blocking or busy waiting or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.19 int Pa_GetVersion (void)

Retrieve the release number of the currently running PortAudio build, eg 1900.

6.1.5.20 const char* Pa_GetVersionText (void)

Retrieve a textual description of the current PortAudio build, eg "PortAudio V19-devel 13 October 2002".

6.1.5.21 PaDeviceIndex Pa_HostApiDeviceIndexToDeviceIndex (PaHostApiIndex hostApi, int hostApiDeviceIndex)

Convert a host-API-specific device index to standard PortAudio device index. This function may be used in conjunction with the deviceCount field of [PaHostApiInfo](#) to enumerate all devices for the specified host API.

Parameter

<i>hostApi</i>	A valid host API index ranging from 0 to (Pa_GetHostApiCount())-1)
<i>hostApiDevice-Index</i>	A valid per-host device index in the range 0 to (Pa_GetHostApiInfo(hostApi)->deviceCount -1)

Rückgabe

A non-negative `PaDeviceIndex` ranging from 0 to ([Pa_GetDeviceCount\(\)](#))-1) or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

A `paInvalidHostApi` error code indicates that the host API index specified by the `hostApi` parameter is out of range.

A `paInvalidDevice` error code indicates that the `hostApiDeviceIndex` parameter is out of range.

Siehe auch

[PaHostApiInfo](#)

6.1.5.22 `PaHostApiIndex Pa_HostApiTypeToHostApiIndex (PaHostApiType type)`

Convert a static host API unique identifier, into a runtime host API index.

Parameter

<i>type</i>	A unique host API identifier belonging to the <code>PaHostApiType</code> enumeration.
-------------	---

Rückgabe

A valid `PaHostApiIndex` ranging from 0 to ([Pa_GetHostApiCount\(\)](#))-1) or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

The `paHostApiNotFound` error code indicates that the host API specified by the `type` parameter is not available.

Siehe auch

[PaHostApiType](#)

6.1.5.23 `PaError Pa_Initialize (void)`

Library initialization function - call this before using PortAudio. This function initializes internal data structures and prepares underlying host APIs for use. With the exception of [Pa_GetVersion\(\)](#), [Pa_GetVersionText\(\)](#), and [Pa_GetErrorText\(\)](#), this function MUST be called before using any other PortAudio API functions.

If [Pa_Initialize\(\)](#) is called multiple times, each successful call must be matched with a corresponding call to [Pa_Terminate\(\)](#). Pairs of calls to [Pa_Initialize\(\)](#)/[Pa_Terminate\(\)](#) may overlap, and are not required to be fully nested.

Note that if [Pa_Initialize\(\)](#) returns an error code, [Pa_Terminate\(\)](#) should NOT be called.

Rückgabe

`paNoError` if successful, otherwise an error code indicating the cause of failure.

Siehe auch

[Pa_Terminate](#)

6.1.5.24 `PaError Pa_IsFormatSupported (const PaStreamParameters * inputParameters, const PaStreamParameters * outputParameters, double sampleRate)`

Determine whether it would be possible to open a stream with the specified parameters.

Parameter

<i>inputParameters</i>	A structure that describes the input parameters used to open a stream. The suggested-Latency field is ignored. See PaStreamParameters for a description of these parameters. inputParameters must be NULL for output-only streams.
<i>outputParameters</i>	A structure that describes the output parameters used to open a stream. The suggested-Latency field is ignored. See PaStreamParameters for a description of these parameters. outputParameters must be NULL for input-only streams.
<i>sampleRate</i>	The required sampleRate. For full-duplex streams it is the sample rate for both input and output

Rückgabe

Returns 0 if the format is supported, and an error code indicating why the format is not supported otherwise. The constant `paFormatIsSupported` is provided to compare with the return value for success.

Siehe auch

[paFormatIsSupported](#), [PaStreamParameters](#)

6.1.5.25 **PaError Pa_IsStreamActive (PaStream * stream)**

Determine whether the stream is active. A stream is active after a successful call to [Pa_StartStream\(\)](#), until it becomes inactive either as a result of a call to [Pa_StopStream\(\)](#) or [Pa_AbortStream\(\)](#), or as a result of a return value other than `paContinue` from the stream callback. In the latter case, the stream is considered inactive after the last buffer has finished playing.

Rückgabe

Returns one (1) when the stream is active (ie playing or recording audio), zero (0) when not playing or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

[Pa_StopStream](#), [Pa_AbortStream](#), [Pa_IsStreamStopped](#)

6.1.5.26 **PaError Pa_IsStreamStopped (PaStream * stream)**

Determine whether the stream is stopped. A stream is considered to be stopped prior to a successful call to `Pa_StartStream` and after a successful call to `Pa_StopStream` or `Pa_AbortStream`. If a stream callback returns a value other than `paContinue` the stream is NOT considered to be stopped.

Rückgabe

Returns one (1) when the stream is stopped, zero (0) when the stream is running or, a `PaErrorCode` (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

[Pa_StopStream](#), [Pa_AbortStream](#), [Pa_IsStreamActive](#)

6.1.5.27 **PaError Pa_OpenDefaultStream (PaStream ** stream, int numInputChannels, int numOutputChannels, PaSampleFormat sampleFormat, double sampleRate, unsigned long framesPerBuffer, PaStreamCallback * streamCallback, void * userData)**

A simplified version of [Pa_OpenStream\(\)](#) that opens the default input and/or output devices.

Parameter

<i>stream</i>	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
<i>numInput-Channels</i>	The number of channels of sound that will be supplied to the stream callback or returned by Pa_ReadStream. It can range from 1 to the value of maxInputChannels in the PaDeviceInfo record for the default input device. If 0 the stream is opened as an output-only stream.
<i>numOutput-Channels</i>	The number of channels of sound to be delivered to the stream callback or passed to Pa_WriteStream. It can range from 1 to the value of maxOutputChannels in the PaDeviceInfo record for the default output device. If 0 the stream is opened as an output-only stream.
<i>sampleFormat</i>	The sample format of both the input and output buffers provided to the callback or passed to and from Pa_ReadStream and Pa_WriteStream. sampleFormat may be any of the formats described by the PaSampleFormat enumeration.
<i>sampleRate</i>	Same as Pa_OpenStream parameter of the same name.
<i>framesPerBuffer</i>	Same as Pa_OpenStream parameter of the same name.
<i>streamCallback</i>	Same as Pa_OpenStream parameter of the same name.
<i>userData</i>	Same as Pa_OpenStream parameter of the same name.

Rückgabe

As for Pa_OpenStream

Siehe auch

[Pa_OpenStream](#), [PaStreamCallback](#)

6.1.5.28 PaError Pa_OpenStream (PaStream ** stream, const PaStreamParameters * inputParameters, const PaStreamParameters * outputParameters, double sampleRate, unsigned long framesPerBuffer, PaStreamFlags streamFlags, PaStreamCallback * streamCallback, void * userData)

Opens a stream for either input, output or both.

Parameter

<i>stream</i>	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
<i>inputParameters</i>	A structure that describes the input parameters used by the opened stream. See PaStreamParameters for a description of these parameters. inputParameters must be NULL for output-only streams.
<i>output-Parameters</i>	A structure that describes the output parameters used by the opened stream. See PaStreamParameters for a description of these parameters. outputParameters must be NULL for input-only streams.
<i>sampleRate</i>	The desired sampleRate. For full-duplex streams it is the sample rate for both input and output
<i>framesPerBuffer</i>	The number of frames passed to the stream callback function, or the preferred block granularity for a blocking read/write stream. The special value paFramesPerBufferUnspecified (0) may be used to request that the stream callback will receive an optimal (and possibly varying) number of frames based on host requirements and the requested latency settings. Note: With some host APIs, the use of non-zero framesPerBuffer for a callback stream may introduce an additional layer of buffering which could introduce additional latency. PortAudio guarantees that the additional latency will be kept to the theoretical minimum however, it is strongly recommended that a non-zero framesPerBuffer value only be used when your algorithm requires a fixed number of frames per stream callback.

<i>streamFlags</i>	Flags which modify the behavior of the streaming process. This parameter may contain a combination of flags ORed together. Some flags may only be relevant to certain buffer formats.
<i>streamCallback</i>	A pointer to a client supplied function that is responsible for processing and filling input and output buffers. If this parameter is NULL the stream will be opened in 'blocking read/write' mode. In blocking mode, the client can receive sample data using <code>Pa_ReadStream</code> and write sample data using <code>Pa_WriteStream</code> , the number of samples that may be read or written without blocking is returned by <code>Pa_GetStreamReadAvailable</code> and <code>Pa_GetStreamWriteAvailable</code> respectively.
<i>userData</i>	A client supplied pointer which is passed to the stream callback function. It could for example, contain a pointer to instance data necessary for processing the audio buffers. This parameter is ignored if <code>streamCallback</code> is NULL.

Rückgabe

Upon success `Pa_OpenStream()` returns `paNoError` and places a pointer to a valid `PaStream` in the stream argument. The stream is inactive (stopped). If a call to `Pa_OpenStream()` fails, a non-zero error code is returned (see `PaError` for possible error codes) and the value of stream is invalid.

Siehe auch

[PaStreamParameters](#), [PaStreamCallback](#), [Pa_ReadStream](#), [Pa_WriteStream](#), [Pa_GetStreamReadAvailable](#), [Pa_GetStreamWriteAvailable](#)

6.1.5.29 `PaError Pa_ReadStream (PaStream * stream, void * buffer, unsigned long frames)`

Read samples from an input stream. The function doesn't return until the entire buffer has been filled - this may involve waiting for the operating system to supply the data.

Parameter

<i>stream</i>	A pointer to an open stream previously created with <code>Pa_OpenStream</code> .
<i>buffer</i>	A pointer to a buffer of sample frames. The buffer contains samples in the format specified by the <code>inputParameters->sampleFormat</code> field used to open the stream, and the number of channels specified by <code>inputParameters->numChannels</code> . If non-interleaved samples were requested using the <code>paNonInterleaved</code> sample format flag, buffer is a pointer to the first element of an array of buffer pointers, one non-interleaved buffer for each channel.
<i>frames</i>	The number of frames to be read into buffer. This parameter is not constrained to a specific range, however high performance applications will want to match this parameter to the <code>framesPerBuffer</code> parameter used when opening the stream.

Rückgabe

On success `paNoError` will be returned, or `paInputOverflowed` if input data was discarded by PortAudio after the previous call and before this call.

6.1.5.30 `PaError Pa_SetStreamFinishedCallback (PaStream * stream, PaStreamFinishedCallback * streamFinishedCallback)`

Register a stream finished callback function which will be called when the stream becomes inactive. See the description of `PaStreamFinishedCallback` for further details about when the callback will be called.

Parameter

<i>stream</i>	a pointer to a PaStream that is in the stopped state - if the stream is not stopped, the stream's finished callback will remain unchanged and an error code will be returned.
<i>streamFinished-Callback</i>	a pointer to a function with the same signature as PaStreamFinishedCallback, that will be called when the stream becomes inactive. Passing NULL for this parameter will un-register a previously registered stream finished callback function.

Rückgabe

on success returns paNoError, otherwise an error code indicating the cause of the error.

Siehe auch

[PaStreamFinishedCallback](#)

6.1.5.31 void Pa_Sleep (long msec)

Put the caller to sleep for at least 'msec' milliseconds. This function is provided only as a convenience for authors of portable code (such as the tests and examples in the PortAudio distribution.)

The function may sleep longer than requested so don't rely on this for accurate musical timing.

6.1.5.32 PaError Pa_StartStream (PaStream * stream)

Commences audio processing.

6.1.5.33 PaError Pa_StopStream (PaStream * stream)

Terminates audio processing. It waits until all pending audio buffers have been played before it returns.

6.1.5.34 PaError Pa_Terminate (void)

Library termination function - call this when finished using PortAudio. This function deallocates all resources allocated by PortAudio since it was initialized by a call to [Pa_Initialize\(\)](#). In cases where [Pa_Initialise\(\)](#) has been called multiple times, each call must be matched with a corresponding call to [Pa_Terminate\(\)](#). The final matching call to [Pa_Terminate\(\)](#) will automatically close any PortAudio streams that are still open.

[Pa_Terminate\(\)](#) MUST be called before exiting a program which uses PortAudio. Failure to do so may result in serious resource leaks, such as audio devices not being available until the next reboot.

Rückgabe

paNoError if successful, otherwise an error code indicating the cause of failure.

Siehe auch

[Pa_Initialize](#)

6.1.5.35 PaError Pa_WriteStream (PaStream * stream, const void * buffer, unsigned long frames)

Write samples to an output stream. This function doesn't return until the entire buffer has been consumed - this may involve waiting for the operating system to consume the data.

Parameter

<i>stream</i>	A pointer to an open stream previously created with Pa_OpenStream.
<i>buffer</i>	A pointer to a buffer of sample frames. The buffer contains samples in the format specified by the outputParameters->sampleFormat field used to open the stream, and the number of channels specified by outputParameters->numChannels. If non-interleaved samples were requested using the paNonInterleaved sample format flag, buffer is a pointer to the first element of an array of buffer pointers, one non-interleaved buffer for each channel.
<i>frames</i>	The number of frames to be written from buffer. This parameter is not constrained to a specific range, however high performance applications will want to match this parameter to the framesPerBuffer parameter used when opening the stream.

Rückgabe

On success PaNoError will be returned, or paOutputUnderflowed if additional output data was inserted after the previous call and before this call.

Index

- ~Audio
 - Audio, [10](#)
- ~AudioControl
 - AudioControl, [12](#)
- ~Input
 - Input, [27](#)
- ~PowerUp
 - PowerUp, [45](#)
- addEntry
 - Menu, [30](#)
- Audio, [9](#)
 - ~Audio, [10](#)
 - Audio, [10](#)
 - getSample, [10](#)
 - getSampLenumber, [10](#)
 - getSource, [10](#)
 - getVolume, [11](#)
 - setVolume, [11](#)
- AudioControl, [11](#)
 - ~AudioControl, [12](#)
 - AudioControl, [12](#)
 - AudioControl, [12](#)
 - playInitialize, [13](#)
 - update, [13](#)
- AudioControl::playStruct, [43](#)
- audioCooldownStruct, [15](#)
- audioCooldownstruct, [15](#)
- audioDistanceStruct, [16](#)
- audioStruct, [17](#)
- changeSelection
 - Menu, [30](#)
- channelCount
 - PaStreamParameters, [39](#)
- clear
 - Menu, [31](#)
- collisionStruct, [18](#)
- compareGameObjects, [19](#)
- compareScores, [19](#)
- currentTime
 - PaStreamCallbackTimeInfo, [37](#)
- decreaseAlcoholLevel
 - Player, [41](#)
- defaultHighInputLatency
 - PaDeviceInfo, [34](#)
- defaultInputDevice
 - PaHostApilInfo, [35](#)
- defaultLowInputLatency
 - PaDeviceInfo, [34](#)
- defaultOutputDevice
 - PaHostApilInfo, [35](#)
- device
 - PaStreamParameters, [39](#)
- deviceCount
 - PaHostApilInfo, [35](#)
- displayInit
 - Menu, [31](#)
- displayUpdate
 - Menu, [31](#)
- Enemy, [20](#)
 - Enemy, [21](#)
 - getDeath, [21](#)
 - getDeathCooldown, [21](#)
 - getFireCooldown, [21](#)
 - getHealth, [21](#)
 - getInflictedDamage, [21](#)
 - setDeath, [22](#)
 - setHealth, [22](#)
- errorCode
 - PaHostErrorInfo, [36](#)
- errorText
 - PaHostErrorInfo, [36](#)
- eventFilter
 - Input, [28](#)
- flipHorizontal
 - MovingObject, [34](#)
- Game, [22](#)
 - Game, [23](#)
 - setState, [23](#)
 - start, [24](#)
 - step, [24](#)
 - timerEvent, [25](#)
- GameObject, [25](#)
 - GameObject, [26](#)
 - GameObject, [26](#)
- getAlcoholLevel
 - Player, [41](#)
- getAlcoholLevelBonus
 - PowerUp, [45](#)
- getAmmunationBonus
 - PowerUp, [45](#)
- getAmmunatiuon
 - Player, [41](#)
- getAndDeleteLastKey
 - Input, [28](#)

- getDeath
 - Enemy, [21](#)
- getDeathCooldown
 - Enemy, [21](#)
- getEntry
 - Menu, [31](#)
- getFireCooldown
 - Enemy, [21](#)
 - Player, [41](#)
- getHealth
 - Enemy, [21](#)
 - Player, [42](#)
- getHealthBonus
 - PowerUp, [45](#)
- getImmunityCooldown
 - Player, [42](#)
- getImmunityCooldownBonus
 - PowerUp, [46](#)
- getInflictedDamage
 - Enemy, [21](#)
 - Player, [42](#)
 - Shoot, [48](#)
- getKeyactions
 - Input, [28](#)
- getOrigin
 - Shoot, [48](#)
- getPowerUPType
 - PowerUp, [46](#)
- getSample
 - Audio, [10](#)
- getSamplenumber
 - Audio, [10](#)
- getSelection
 - Menu, [31](#)
- getSource
 - Audio, [10](#)
- getTitle
 - Menu, [32](#)
- getType
 - Menu, [32](#)
- getVolume
 - Audio, [11](#)
- hostApi
 - PaDeviceInfo, [35](#)
- hostApiSpecificStreamInfo
 - PaStreamParameters, [39](#)
- hostApiType
 - PaHostErrorInfo, [36](#)
- inJump
 - Player, [42](#)
- increaseAlcoholLevel
 - Player, [42](#)
- Input, [26](#)
 - ~Input, [27](#)
 - eventFilter, [28](#)
 - getAndDeleteLastKey, [28](#)
 - getKeyactions, [28](#)
 - Input, [27](#)
- inputBufferAdcTime
 - PaStreamCallbackTimeInfo, [37](#)
- inputLatency
 - PaStreamInfo, [38](#)
- Menu, [29](#)
 - addEntry, [30](#)
 - changeSelection, [30](#)
 - clear, [31](#)
 - displayInit, [31](#)
 - displayUpdate, [31](#)
 - getEntry, [31](#)
 - getSelection, [31](#)
 - getTitle, [32](#)
 - getType, [32](#)
 - Menu, [30](#)
- Menu::menuEntry, [32](#)
- MovingObject, [33](#)
 - flipHorizontal, [34](#)
 - updatePosition, [34](#)
- name
 - PaHostApiInfo, [35](#)
- outputBufferDacTime
 - PaStreamCallbackTimeInfo, [37](#)
- outputLatency
 - PaStreamInfo, [38](#)
- paAbort
 - portaudio.h, [61](#)
- paComplete
 - portaudio.h, [61](#)
- paContinue
 - portaudio.h, [61](#)
- Pa_AbortStream
 - portaudio.h, [61](#)
- Pa_CloseStream
 - portaudio.h, [61](#)
- Pa_GetDefaultHostApi
 - portaudio.h, [61](#)
- Pa_GetDefaultInputDevice
 - portaudio.h, [62](#)
- Pa_GetDefaultOutputDevice
 - portaudio.h, [62](#)
- Pa_GetDeviceCount
 - portaudio.h, [62](#)
- Pa_GetDeviceInfo
 - portaudio.h, [62](#)
- Pa_GetErrorText
 - portaudio.h, [63](#)
- Pa_GetHostApiCount
 - portaudio.h, [63](#)
- Pa_GetHostApiInfo
 - portaudio.h, [63](#)
- Pa_GetLastHostErrorInfo
 - portaudio.h, [63](#)
- Pa_GetSampleSize

- portaudio.h, [63](#)
- Pa_GetStreamCpuLoad
 - portaudio.h, [64](#)
- Pa_GetStreamHostApiType
 - portaudio.h, [64](#)
- Pa_GetStreamInfo
 - portaudio.h, [64](#)
- Pa_GetStreamReadAvailable
 - portaudio.h, [64](#)
- Pa_GetStreamTime
 - portaudio.h, [65](#)
- Pa_GetStreamWriteAvailable
 - portaudio.h, [65](#)
- Pa_GetVersion
 - portaudio.h, [65](#)
- Pa_GetVersionText
 - portaudio.h, [65](#)
- Pa_HostApiDeviceIndexToDeviceIndex
 - portaudio.h, [65](#)
- Pa_HostApiTypeIdToHostApiIndex
 - portaudio.h, [66](#)
- Pa_Initialize
 - portaudio.h, [66](#)
- Pa_IsFormatSupported
 - portaudio.h, [66](#)
- Pa_IsStreamActive
 - portaudio.h, [67](#)
- Pa_IsStreamStopped
 - portaudio.h, [67](#)
- Pa_OpenDefaultStream
 - portaudio.h, [67](#)
- Pa_OpenStream
 - portaudio.h, [68](#)
- Pa_ReadStream
 - portaudio.h, [69](#)
- Pa_SetStreamFinishedCallback
 - portaudio.h, [69](#)
- Pa_Sleep
 - portaudio.h, [70](#)
- Pa_StartStream
 - portaudio.h, [70](#)
- Pa_StopStream
 - portaudio.h, [70](#)
- Pa_Terminate
 - portaudio.h, [70](#)
- Pa_WriteStream
 - portaudio.h, [70](#)
- paClipOff
 - portaudio.h, [53](#)
- paCustomFormat
 - portaudio.h, [53](#)
- PaDeviceIndex
 - portaudio.h, [57](#)
- PaDeviceInfo, [34](#)
 - defaultHighInputLatency, [34](#)
 - defaultLowInputLatency, [34](#)
 - hostApi, [35](#)
 - portaudio.h, [57](#)
- paDitherOff
 - portaudio.h, [53](#)
- PaError
 - portaudio.h, [57](#)
- paFloat32
 - portaudio.h, [54](#)
- paFormatIsSupported
 - portaudio.h, [54](#)
- paFramesPerBufferUnspecified
 - portaudio.h, [54](#)
- PaHostApiIndex
 - portaudio.h, [57](#)
- PaHostApiInfo, [35](#)
 - defaultInputDevice, [35](#)
 - defaultOutputDevice, [35](#)
 - deviceCount, [35](#)
 - name, [35](#)
 - portaudio.h, [57](#)
 - structVersion, [36](#)
 - type, [36](#)
- PaHostApiTypeId
 - portaudio.h, [57](#), [61](#)
- PaHostErrorInfo, [36](#)
 - errorCode, [36](#)
 - errorText, [36](#)
 - hostApiType, [36](#)
 - portaudio.h, [57](#)
- paInputOverflow
 - portaudio.h, [54](#)
- paInputUnderflow
 - portaudio.h, [54](#)
- paInt16
 - portaudio.h, [54](#)
- paInt24
 - portaudio.h, [54](#)
- paInt32
 - portaudio.h, [55](#)
- paInt8
 - portaudio.h, [55](#)
- paNeverDropInput
 - portaudio.h, [55](#)
- paNoDevice
 - portaudio.h, [55](#)
- paNoFlag
 - portaudio.h, [55](#)
- paNonInterleaved
 - portaudio.h, [55](#)
- paOutputOverflow
 - portaudio.h, [55](#)
- paOutputUnderflow
 - portaudio.h, [56](#)
- paPlatformSpecificFlags
 - portaudio.h, [56](#)
- paPrimeOutputBuffersUsingStreamCallback
 - portaudio.h, [56](#)
- paPrimingOutput
 - portaudio.h, [56](#)
- PaSampleFormat

- portaudio.h, [57](#)
- PaStream
 - portaudio.h, [58](#)
- PaStreamCallback
 - portaudio.h, [58](#)
- PaStreamCallbackFlags
 - portaudio.h, [59](#)
- PaStreamCallbackResult
 - portaudio.h, [59](#), [61](#)
- PaStreamCallbackTimeInfo, [37](#)
 - currentTime, [37](#)
 - inputBufferAdcTime, [37](#)
 - outputBufferDacTime, [37](#)
 - portaudio.h, [59](#)
- PaStreamFinishedCallback
 - portaudio.h, [60](#)
- PaStreamFlags
 - portaudio.h, [60](#)
- PaStreamInfo, [37](#)
 - inputLatency, [38](#)
 - outputLatency, [38](#)
 - portaudio.h, [60](#)
 - sampleRate, [38](#)
 - structVersion, [38](#)
- PaStreamParameters, [38](#)
 - channelCount, [39](#)
 - device, [39](#)
 - hostApiSpecificStreamInfo, [39](#)
 - portaudio.h, [60](#)
 - sampleFormat, [39](#)
 - suggestedLatency, [39](#)
- PaTime
 - portaudio.h, [60](#)
- paUInt8
 - portaudio.h, [56](#)
- paUseHostApiSpecificDeviceSpecification
 - portaudio.h, [56](#)
- playInitialize
 - AudioControl, [13](#)
- Player, [40](#)
 - decreaseAlcoholLevel, [41](#)
 - getAlcoholLevel, [41](#)
 - getAmmunatiuon, [41](#)
 - getFireCooldown, [41](#)
 - getHealth, [42](#)
 - getImmunityCooldown, [42](#)
 - getInflictedDamage, [42](#)
 - inJump, [42](#)
 - increaseAlcoholLevel, [42](#)
 - Player, [41](#)
 - receiveDamage, [42](#)
 - setHealth, [43](#)
 - setImmunityCooldown, [43](#)
 - update, [43](#)
- portaudio.h
 - paAbort, [61](#)
 - paComplete, [61](#)
 - paContinue, [61](#)
- portaudio.h
 - Pa_AbortStream, [61](#)
 - Pa_CloseStream, [61](#)
 - Pa_GetDefaultHostApi, [61](#)
 - Pa_GetDefaultInputDevice, [62](#)
 - Pa_GetDefaultOutputDevice, [62](#)
 - Pa_GetDeviceCount, [62](#)
 - Pa_GetDeviceInfo, [62](#)
 - Pa_GetErrorText, [63](#)
 - Pa_GetHostApiCount, [63](#)
 - Pa_GetHostApiInfo, [63](#)
 - Pa_GetLastHostErrorInfo, [63](#)
 - Pa_GetSampleSize, [63](#)
 - Pa_GetStreamCpuLoad, [64](#)
 - Pa_GetStreamHostApiType, [64](#)
 - Pa_GetStreamInfo, [64](#)
 - Pa_GetStreamReadAvailable, [64](#)
 - Pa_GetStreamTime, [65](#)
 - Pa_GetStreamWriteAvailable, [65](#)
 - Pa_GetVersion, [65](#)
 - Pa_GetVersionText, [65](#)
 - Pa_HostApiDeviceIndexToDeviceIndex, [65](#)
 - Pa_HostApiTypeIdToHostApiIndex, [66](#)
 - Pa_Initialize, [66](#)
 - Pa_IsFormatSupported, [66](#)
 - Pa_IsStreamActive, [67](#)
 - Pa_IsStreamStopped, [67](#)
 - Pa_OpenDefaultStream, [67](#)
 - Pa_OpenStream, [68](#)
 - Pa_ReadStream, [69](#)
 - Pa_SetStreamFinishedCallback, [69](#)
 - Pa_Sleep, [70](#)
 - Pa_StartStream, [70](#)
 - Pa_StopStream, [70](#)
 - Pa_Terminate, [70](#)
 - Pa_WriteStream, [70](#)
 - paClipOff, [53](#)
 - paCustomFormat, [53](#)
 - PaDeviceIndex, [57](#)
 - PaDeviceInfo, [57](#)
 - paDitherOff, [53](#)
 - PaError, [57](#)
 - paFloat32, [54](#)
 - paFormatIsSupported, [54](#)
 - paFramesPerBufferUnspecified, [54](#)
 - PaHostApiIndex, [57](#)
 - PaHostApiInfo, [57](#)
 - PaHostApiTypeId, [57](#), [61](#)
 - PaHostErrorInfo, [57](#)
 - paInputOverflow, [54](#)
 - paInputUnderflow, [54](#)
 - paInt16, [54](#)
 - paInt24, [54](#)
 - paInt32, [55](#)
 - paInt8, [55](#)
 - paNeverDropInput, [55](#)
 - paNoDevice, [55](#)
 - paNoFlag, [55](#)

- paNonInterleaved, [55](#)
- paOutputOverflow, [55](#)
- paOutputUnderflow, [56](#)
- paPlatformSpecificFlags, [56](#)
- paPrimeOutputBuffersUsingStreamCallback, [56](#)
- paPrimingOutput, [56](#)
- PaSampleFormat, [57](#)
- PaStream, [58](#)
- PaStreamCallback, [58](#)
- PaStreamCallbackFlags, [59](#)
- PaStreamCallbackResult, [59](#), [61](#)
- PaStreamCallbackTimeInfo, [59](#)
- PaStreamFinishedCallback, [60](#)
- PaStreamFlags, [60](#)
- PaStreamInfo, [60](#)
- PaStreamParameters, [60](#)
- PaTime, [60](#)
- paUInt8, [56](#)
- paUseHostApiSpecificDeviceSpecification, [56](#)
- PowerUp, [44](#)
 - ~PowerUp, [45](#)
 - getAlcoholLevelBonus, [45](#)
 - getAmmunationBonus, [45](#)
 - getHealthBonus, [45](#)
 - getImmunityCooldownBonus, [46](#)
 - getPowerUPType, [46](#)
 - PowerUp, [45](#)
 - PowerUp, [45](#)
- receiveDamage
 - Player, [42](#)
- sampleFormat
 - PaStreamParameters, [39](#)
- sampleRate
 - PaStreamInfo, [38](#)
- scoreStruct, [46](#)
- setDeath
 - Enemy, [22](#)
- setHealth
 - Enemy, [22](#)
 - Player, [43](#)
- setImmunityCooldown
 - Player, [43](#)
- setState
 - Game, [23](#)
- setVolume
 - Audio, [11](#)
- Shoot, [47](#)
 - getInflictedDamage, [48](#)
 - getOrigin, [48](#)
 - Shoot, [47](#)
- start
 - Game, [24](#)
- stateStruct, [48](#)
- step
 - Game, [24](#)
- structVersion
 - PaHostApilInfo, [36](#)
 - PaStreamInfo, [38](#)
 - suggestedLatency
 - PaStreamParameters, [39](#)
 - timerEvent
 - Game, [25](#)
 - type
 - PaHostApilInfo, [36](#)
 - update
 - AudioControl, [13](#)
 - Player, [43](#)
 - updatePosition
 - MovingObject, [34](#)
- Wiesn-Run/src/portaudio.h, [51](#)