Topaz

Generated by Doxygen 1.8.14

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 tz Namespace Reference

Namespaces

- transform

Functions

- void initialise ()
- void terminate ()

Variables

- constexpr char **default_properties_path** [ ] = "properties.mdl"

4.1.1 Detailed Description

Initialise and terminate tz audio module. This must be done appropriately to use any of the audio functionality.

4.1.2 Function Documentation

4.1.2.1 initialise()

```
void tz::initialise ( )  [inline]
```

Invoke this to initialise all Topaz modules and be able to use all features. Note that a Topaz window must have been constructed at least once before the graphics module fully initialises.

No audio works until this is executed. tz::initialise will invoke this automatically.

Initialises the graphics component of Topaz. An OpenGL window-context MUST exist before this is executed. Topaz Window constructor invokes this upon first invocation. If you are not using Topaz windows, you will have to invoke this yourself. If you are using Topaz windows, DO NOT TOUCH THIS.

4.1.2.2 terminate()

```
void tz::terminate ( )  [inline]
```

Invoke this to close all Topaz modules and free corresponding memory to prevent droplets Once this function is executed, most features will cease to work properly and will likely invoke undefined behaviour.

Memory droplets will remain until this is executed. Audio will not work after this is invoked. tz::terminate will invoke this automatically.

Terminates and destroys all graphics components of Topaz. tz::terminate will invoke this automatically. Only use this function if you do not wish to use tz::terminate to terminate all features, but instead cherry-pick components like graphics.

4.2 tz::graphics::model Namespace Reference

Classes

- class IndexedModel
- class OBJIndex
- class OBJModel

4.2.1 Detailed Description

Currently used only for Wavefront OBJ Models. To load an OBJ model in Topaz, invoke OBJModel::toIndexedModel to receive an instance of IndexedModel. A Topaz Mesh constructor can take an IndexedModel as a parameter.

4.3 tz::graphics::shader Namespace Reference

Functions

- Shader **pass_through** (std::string position_attribute_name="position_modelspace_attribute", std↩ ::string texture_coordinate_attribute_name="texture_coordinate_attribute", std::string texture_sampler_↩ name="texture_sampler_uniform")

4.3.1 Detailed Description

Factory functions for Shaders that need no special source code; simply pass-through shaders and the ability to render 3D geometry, with some plain old textures. If your application is so simple that only the simplest matrix transformations are needed with no fancy effects, use this.

4.4 tz::transform Namespace Reference

Functions

- Matrix4x4 translate (const Vector3F &position)
- Matrix4x4 rotate_x (float euler_x)
- Matrix4x4 rotate_y (float euler_y)
- Matrix4x4 rotate_z (float euler_z)
- Matrix4x4 rotate (const Vector3F &euler_rotation)
- Matrix4x4 scale (const Vector3F &scale)
- Matrix4x4 model (const Vector3F &position, const Vector3F &euler_rotation, const Vector3F &scale)
- Matrix4x4 view (const Vector3F &camera_position, const Vector3F &camera_euler_rotation)
- Matrix4x4 orthographic_projection (float right, float left, float top, float bottom, float near, float far)
- Matrix4x4 perspective_projection (float fov, float width, float height, float nearclip, float farclip)

4.4.1 Detailed Description

Topaz transformation matrices are always in row-major format.

4.4.2 Function Documentation

4.4.2.1 model()

```
Matrix4x4 tz::transform::model (
            const Vector3F & position,
            const Vector3F & euler_rotation,
            const Vector3F & scale )
```

Construct a row-major model matrix using the functions above.

4.4.2.2 orthographic_projection()

```
Matrix4x4 tz::transform::orthographic_projection (
            float right,
            float left,
            float top,
            float bottom,
            float near,
            float far )
```

Construct a row-major projection matrix to create an orthographic projection.

4.4.2.3 perspective_projection()

```
Matrix4x4 tz::transform::perspective_projection (
             float fov,
             float width,
             float height,
             float near_clip,
             float far_clip )
```

Construct a row-major projection matrix to create a perspective projection. Use this to simulate 3D with a "camera".

4.4.2.4 rotate()

```
Matrix4x4 tz::transform::rotate (
             const Vector3F & euler_rotation )
```

Construct a four-dimensional row-major rotational matrix using XYZ rotations (Pitch, Yaw, Roll in Euler-angles).

4.4.2.5 rotate_x()

```
Matrix4x4 tz::transform::rotate_x (
             float euler_x )
```

Construct a four-dimensional row-major rotational matrix in the x-axis (Pitch).

4.4.2.6 rotate_y()

```
Matrix4x4 tz::transform::rotate_y (
             float euler_y )
```

Construct a four-dimensional row-major rotational matrix in the y-axis (Yaw).

4.4.2.7 rotate_z()

```
Matrix4x4 tz::transform::rotate_z (
             float euler_z )
```

Construct a four-dimensional row-major rotational matrix in the x-axis (Roll).

4.4.2.8 scale()

```
Matrix4x4 tz::transform::scale (
             const Vector3F & scale )
```

Construct a four-dimensional row-major scaling matrix.

4.4.2.9 translate()

```
Matrix4x4 tz::transform::translate (
            const Vector3F & position )
```

Construct a four-dimensional row-major translation matrix.

4.4.2.10 view()

```
Matrix4x4 tz::transform::view (
            const Vector3F & camera_position,
            const Vector3F & camera_euler_rotation )
```

Construct a row-major view matrix using the functions above. Works similarly to gluLookAt(...)

4.5 tz::util::log Namespace Reference

Functions

- void **silent** ()
- template<typename FirstArg , typename... Args>
  void **silent** (FirstArg arg, Args... args)
- template<typename FirstArg = void, typename... Args>
  void **message** (FirstArg arg, Args... args)
- template<typename FirstArg = void, typename... Args>
  void **warning** (FirstArg arg, Args... args)
- template<typename FirstArg = void, typename... Args>
  void **error** (FirstArg arg, Args... args)

4.5.1 Detailed Description

Log to the console variadically. Like printf, but without the formatting and with type-safety.

4.6 tz::util::string Namespace Reference

Functions

- std::string **to_lower** (std::string data)
- std::string **to_upper** (std::string data)
- bool **begins_with** (const std::string &what, const std::string &with_what)
- bool **ends_with** (const std::string &what, const std::string &with_what)
- bool **contains** (const std::string &what, char withwhat)
- std::vector< std::string > **split_string** (const std::string &s, const std::string &delim)
- std::vector< std::string > **split_string** (const std::string &s, char delim)
- std::string **replace_all_char** (const std::string &str, char toreplace, const std::string &replacewith)
- std::string **replace_all** (std::string str, const std::string &to_replace, const std::string &replace_with)
- std::string **substring** (const std::string &str, std::size_t begin, std::size_t end)
- std::string **format** (const std::vector< std::string > &split)
- std::vector< std::string > **deformat** (const std::string &str)
- template<typename T >
  Vector3< T > **vectorise_list_3** (const std::vector< std::string > &list)
- template<typename T >
  std::vector< std::string > **devectorise_list_3** (Vector3< T > v)

4.6.1 Detailed Description

Perform processing on std::strings with these utility functions.

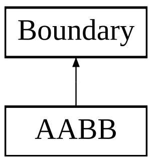
Chapter 5

Class Documentation

5.1 AABB Class Reference

```
#include <boundary.hpp>
```

Inheritance diagram for AABB:



Public Member Functions

- **AABB** (Vector3F minimum, Vector3F maximum)
- **AABB** (const AABB &copy)=default
- **AABB** (AABB &&move)=default
- AABB & **operator=** (const AABB &rhs)=default
- const Vector3F & **get_minimum** () const
- const Vector3F & **get_maximum** () const
- bool **intersects** (const AABB &rhs) const
- bool **intersects** (const Vector3F &point) const
- virtual bool intersects (Boundary ∗other_boundary) const override

5.1.1 Detailed Description

Axis-Aligned Bounding-Box. Very lightweight but is a very limited and minimalistic box-shaped boundary for an object. Use if performance > precision.

5.1.2 Member Function Documentation

5.1.2.1 intersects()

```
bool AABB::intersects (
            Boundary * other_boundary ) const  [override], [virtual]
```

Pure virtual. Override this if you want to make your own Boundaries.

Implements Boundary.

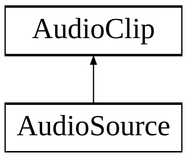
The documentation for this class was generated from the following files:

- src/physics/boundary.hpp
- src/physics/boundary.cpp

5.2 AudioClip Class Reference

```
#include <audio.hpp>
```

Inheritance diagram for AudioClip:



Public Member Functions

- AudioClip (std::string filename)
- AudioClip (const AudioClip &copy)
- AudioClip (AudioClip &&move)
- virtual ∼AudioClip ()
- AudioClip & **operator=** (const AudioClip &rhs)=delete
- void play ()
- int **get_channel** () const
- Uint32 **get_audio_length** () const
- const std::string & **get_file_name** () const

5.2.1 Detailed Description

Playable audio file. Use this for short audio segments like sound effects.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AudioClip() [1/3]

```
AudioClip::AudioClip (
            std::string filename )
```

Load AudioClip from existing file (must be wavefront audio .wav)

5.2.2.2 AudioClip() [2/3]

```
AudioClip::AudioClip (
            const AudioClip & copy )
```

Construct AudioClip using the filename of copy.

5.2.2.3 AudioClip() [3/3]

```
AudioClip::AudioClip (
            AudioClip && move )
```

Construct AudioClip using the same chunk as move. Also copies move's filename.

5.2.2.4 ∼AudioClip()

```
AudioClip::∼AudioClip ( )  [virtual]
```

Deallocate memory from the SDL_Mixer functionality.

5.2.3 Member Function Documentation

5.2.3.1 play()

```
void AudioClip::play ( )
```

Plays the audio. The audio will play until either the destructor is called or the audio is finished; whichever takes place first. Note: Invoking tz::audio::play_async on an instance of AudioClip will extend the lifetime of the instance such that the audio clip is guaranteed to be fully played.

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

- src/audio/audio.hpp
- src/audio/audio.cpp

5.3 AudioMusic Class Reference

```
#include <audio.hpp>
```

Public Member Functions

- **AudioMusic** (std::string filename)
- **AudioMusic** (const AudioMusic &copy)
- **AudioMusic** (AudioMusic &&move)
- AudioMusic & **operator=** (const AudioMusic &rhs)=delete
- const std::string & **get_file_name** () const
- bool **is_paused** () const
- void play (bool priority=true) const
- void set_paused (bool pause=true)

5.3.1 Detailed Description

Playable audio file. Use this for longer audio segments such as background music.

5.3.2 Member Function Documentation

5.3.2.1 play()

```
void AudioMusic::play (
            bool priority = true ) const
```

Play should be invoked only once and not to un-pause music.

5.3.2.2 set_paused()

```
void AudioMusic::set_paused (
            bool pause = true )
```

Pause/Resume the music.

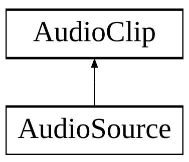
The documentation for this class was generated from the following files:

- src/audio/audio.hpp
- src/audio/audio.cpp

5.4 AudioSource Class Reference

```
#include <audio.hpp>
```

Inheritance diagram for AudioSource:



Public Member Functions

- **AudioSource** (std::string filename)
- **AudioSource** (const AudioSource &copy)=default
- **AudioSource** (AudioSource &&move)=default
- AudioSource & **operator=** (const AudioSource &rhs)=delete
- void update (const Vector3F &source_position, const Camera &relative_to) const

5.4.1 Detailed Description

Playable audio file, but from a position in 3D space. Same properties as AudioClip.

5.4.2 Member Function Documentation

5.4.2.1 update()

```
void AudioSource::update (
            const Vector3F & source_position,
            const Camera & relative_to ) const
```

Should be invoked whenever the camera rotates or moves or the AudioSource position is changed.

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

- src/audio/audio.hpp
- src/audio/audio.cpp

5.5 Bitmap< Pixel > Class Template Reference

```
#include <graphics.hpp>
```

Public Member Functions

- **Bitmap** (std::vector< Pixel > pixels=std::vector< Pixel >(), int width=0, int height=0)

Public Attributes

- std::vector< Pixel > **pixels**
- int **width**
- int **height**

5.5.1 Detailed Description

template<class Pixel = PixelRGBA>
class Bitmap< Pixel >

Bitmap representing Pixel data in any format. Topaz uses PixelRGBA as the template parameter, but you may provide any valid class with a public Vector4<T> called 'data'.

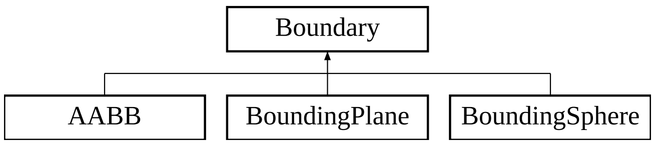
The documentation for this class was generated from the following file:

- src/graphics/graphics.hpp

5.6 Boundary Class Reference

```
#include <boundary.hpp>
```

Inheritance diagram for Boundary:



Public Member Functions

- **Boundary** (const Boundary &copy)=default
- **Boundary** (Boundary &&move)=default
- Boundary & **operator=** (const Boundary &rhs)=default
- virtual bool intersects (Boundary ∗other_boundary) const =0

5.6.1 Detailed Description

Abstract. Not available for non-polymorphic use. Inherit from this to create custom boundaries. Represents a simple boundary in space.

5.6.2 Member Function Documentation

5.6.2.1 intersects()

```
virtual bool Boundary::intersects (
            Boundary ∗ other_boundary ) const  [pure virtual]
```

Pure virtual. Override this if you want to make your own Boundaries.

Implemented in BoundingPlane, AABB, and BoundingSphere.

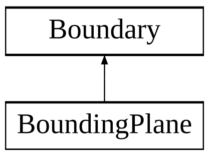
The documentation for this class was generated from the following file:

- src/physics/boundary.hpp

5.7 BoundingPlane Class Reference

```
#include <boundary.hpp>
```

Inheritance diagram for BoundingPlane:



Public Member Functions

- **BoundingPlane** (Vector3F normal, float distance)
- **BoundingPlane** (const BoundingPlane &copy)=default
- **BoundingPlane** (BoundingPlane &&move)=default
- BoundingPlane & **operator=** (const BoundingPlane &rhs)=default
- const Vector3F & **get_normal** () const
- float **get_distance** () const
- BoundingPlane **normalised** () const
- bool **intersects** (const BoundingSphere &other) const
- virtual bool intersects (Boundary ∗other_boundary) const override

5.7.1 Detailed Description

Used to bound planes. Useful for objects such as walls or floors.

5.7.2 Member Function Documentation

5.7.2.1 intersects()

```
virtual bool BoundingPlane::intersects (
            Boundary ∗ other_boundary ) const  [override], [virtual]
```

Pure virtual. Override this if you want to make your own Boundaries.

Implements Boundary.

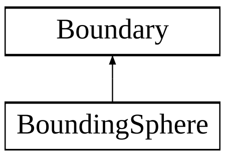
The documentation for this class was generated from the following files:

- src/physics/boundary.hpp
- src/physics/boundary.cpp

5.8 BoundingSphere Class Reference

```
#include <boundary.hpp>
```

Inheritance diagram for BoundingSphere:



Public Member Functions

- **BoundingSphere** (Vector3F centre, float radius)
- **BoundingSphere** (const BoundingSphere &copy)=default
- **BoundingSphere** (BoundingSphere &&move)=default
- BoundingSphere & **operator=** (const BoundingSphere &rhs)=default
- const Vector3F & **get_centre** () const
- float **get_radius** () const
- bool **intersects** (const BoundingSphere &rhs) const
- virtual bool intersects (Boundary ∗other_boundary) const override

5.8.1 Detailed Description

Used to bound physical spherical shapes in 3D space.

5.8.2 Member Function Documentation

5.8.2.1 intersects()

```
bool BoundingSphere::intersects (
            Boundary ∗ other_boundary ) const  [override], [virtual]
```

Pure virtual. Override this if you want to make your own Boundaries.

Implements Boundary.

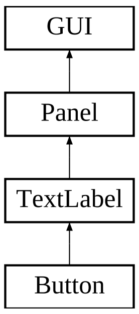
The documentation for this class was generated from the following files:

- src/physics/boundary.hpp
- src/physics/boundary.cpp

5.9 Button Class Reference

```
#include <gui_widget.hpp>
```

Inheritance diagram for Button:



Public Member Functions

- **Button** (float x, float y, Vector4F colour, std::optional< Vector4F > background_colour, std::optional< Vector3F > text_border_colour, Font font, const std::string &text, Shader &shader, MouseListener &mouse↩_listener)
- **Button** (const Button &rhs)=default
- **Button** (Button &&move)=default
- Button & **operator=** (const Button &rhs)=default
- virtual void **update** () override
- virtual bool **focused** () const override
- virtual bool **is_mouse_sensitive** () const override
- Command ∗ get_on_mouse_over () const
- Command ∗ get_on_mouse_click () const
- void set_on_mouse_over (Command ∗cmd)
- void set_on_mouse_click (Command ∗cmd)
- bool **moused_over** () const
- bool **clicked_on** () const

Protected Attributes

- MouseListener & **mouse_listener**
- bool **just_clicked**
- bool **just_moused_over**

Additional Inherited Members

5.9.1 Detailed Description

Just like a TextLabel, but also is mouse-sensitive and pressable to execute a command.

5.9.2 Member Function Documentation

5.9.2.1 get_on_mouse_click()

```
Command * Button::get_on_mouse_click ( ) const
```

Read-only access to the command executed when the Button is pressed.

5.9.2.2 get_on_mouse_over()

```
Command * Button::get_on_mouse_over ( ) const
```

Read-only access to the command executed when the Button is moused-over.

5.9.2.3 set_on_mouse_click()

```
void Button::set_on_mouse_click (
             Command * cmd )
```

Change what happens when the button is left-clicked. Inputting 'nullptr' will mean that nothing happens.

5.9.2.4 set_on_mouse_over()

```
void Button::set_on_mouse_over (
             Command * cmd )
```

Change what happens when the button is moused-over. Inputting 'nullptr' will mean that nothing happens.

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

- src/graphics/gui_widget.hpp
- src/graphics/gui_widget.cpp

5.10 Camera Class Reference

```
#include <camera.hpp>
```

Public Member Functions

- **Camera** (Vector3F position=Vector3F(), Vector3F rotation=Vector3F(), float fov=tz::graphics::default_fov, float near_clip=tz::graphics::default_near_clip, float far_clip=tz::graphics::default_far_clip, bool perspective=true)
- **Camera** (const Camera &copy)=default
- **Camera** (Camera &&move)=default
- Camera & **operator=** (const Camera &rhs)=default
- Vector3F forward () const
- Vector3F **backward** () const
- Vector3F **up** () const
- Vector3F **down** () const
- Vector3F **left** () const
- Vector3F **right** () const
- bool is_axis_bound () const
- void **set_axis_bound** (bool axis_bound)
- bool **has_perspective_projection** () const
- void **set_has_perspective_projection** (bool perspective)
- Matrix4x4 **projection** (float width, float height) const

Public Attributes

- Vector3F **position**
- Vector3F **rotation**
- float **fov**
- float **near_clip**
- float **far_clip**

5.10.1 Detailed Description

Used to navigate the 3D scene such that all objects in the scene do not have to be moved in order to achieve motion.

5.10.2 Member Function Documentation

5.10.2.1 forward()

```
Vector3F Camera::forward ( ) const
```

Orientation methods.

5.10.2.2 is_axis_bound()

```
bool Camera::is_axis_bound ( ) const
```

Axis-Bound Camera causes orientation methods to stick to their normal axes. Also prevents rotation in the x and z axis.

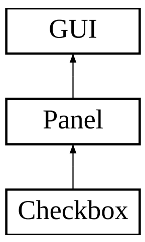
The documentation for this class was generated from the following files:

- src/camera.hpp
- src/camera.cpp

5.11 Checkbox Class Reference

```
#include <gui_widget.hpp>
```

Inheritance diagram for Checkbox:



Public Member Functions

- **Checkbox** (float x, float y, float width, float height, Vector4F colour_on, Vector4F colour_off, Shader &shader, MouseListener &mouse_listener, bool ticked=false)
- **Checkbox** (const Checkbox &copy)=default
- **Checkbox** (Checkbox &&move)=default
- Checkbox & **operator=** (const Checkbox &rhs)=default
- virtual void **update** () override
- virtual bool **focused** () const override
- virtual bool **is_mouse_sensitive** () const override
- bool **moused_over** () const
- bool **clicked_on** () const
- const Vector4F & get_colour_on () const
- const Vector4F & get_colour_off () const
- Vector4F get_colour () const

Public Attributes

- bool **value**

Protected Member Functions

- bool **is_choice** () const

Protected Attributes

- Vector4F **colour_on**
- Vector4F **colour_off**
- MouseListener & **mouse_listener**
- bool **just_clicked**
- bool **just_moused_over**
- CheckboxChoice ∗ choice_parent

Friends

- class **CheckboxChoice**

5.11.1 Detailed Description

Graphical representation of a mutable boolean. Use this to enable user-input for toggling booleans.

5.11.2 Member Function Documentation

5.11.2.1 get_colour()

```
Vector4F Checkbox::get_colour ( ) const
```

Retrieve the colour of the checkbox. The colour depends on whether the value is true or false.

5.11.2.2 get_colour_off()

```
const Vector4F & Checkbox::get_colour_off ( ) const
```

Get colour of the checkbox when the value is false.

5.11.2.3 get_colour_on()

```
const Vector4F & Checkbox::get_colour_on ( ) const
```

Get colour of the checkbox when the value is true.

5.11.3 Member Data Documentation

5.11.3.1 choice_parent

```
CheckboxChoice* Checkbox::choice_parent [protected]
```

choice_parent is handled purely by the friend-class CheckboxChoice.

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

- src/graphics/gui_widget.hpp
- src/graphics/gui_widget.cpp

5.12 CheckboxChoice Class Reference

```
#include <gui_widget.hpp>
```

Public Member Functions

- CheckboxChoice (std::initializer_list< Checkbox ∗> boxes, Checkbox ∗initial_choice=nullptr)
- CheckboxChoice (std::initializer_list< std::reference_wrapper< Checkbox >> boxes, Checkbox ∗initial_↩ choice=nullptr)
- CheckboxChoice (const CheckboxChoice &copy)=delete
- CheckboxChoice (CheckboxChoice &&move)=default
- CheckboxChoice & operator= (const CheckboxChoice &rhs)=delete
- CheckboxChoice & operator= (CheckboxChoice &&rhs)=default
- bool **has_choice** () const
- Checkbox ∗ **get_choice** () const
- void **set_choice** (Checkbox ∗choice)
- const std::unordered_set< Checkbox ∗ > & **get_bool_boxes** () const

5.12.1 Detailed Description

Non-owning helper class to manage multiple Checkboxes. Use this to allow multiple checkboxes to only have one truthy at a time.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 CheckboxChoice() [1/4]

```
CheckboxChoice::CheckboxChoice (
            std::initializer_list< Checkbox *> boxes,
            Checkbox * initial_choice = nullptr )
```

Non-owning pointer initialisation.

5.12.2.2 CheckboxChoice() [2/4]

```
CheckboxChoice::CheckboxChoice (
            std::initializer_list< std::reference_wrapper< Checkbox >> boxes,
            Checkbox * initial_choice = nullptr )
```

Non-owning reference initialisation.

5.12.2.3 CheckboxChoice() [3/4]

```
CheckboxChoice::CheckboxChoice (
            const CheckboxChoice & copy )  [delete]
```

Copy constructor deleted. This is because any two CheckboxChoices may not share a single element, or "choice-fighting" will occur. Choice-fighting is the phenomenon such that multiple CheckboxChoices choose different boxes to truthify, causing the latter to always invalidate the former.

5.12.2.4 CheckboxChoice() [4/4]

```
CheckboxChoice::CheckboxChoice (
            CheckboxChoice && move )  [default]
```

Move constructor implies the rvalue-reference parameter is about to go out-of-scope, meaning that it will not be available to cause choice-fighting. For this reason, the move constructor is available.

5.12.3 Member Function Documentation

5.12.3.1 operator=() [1/2]

```
CheckboxChoice& CheckboxChoice::operator= (
            const CheckboxChoice & rhs )  [delete]
```

Copy assignment operator is also deleted for the same reason; choice-fighting.

5.12.3.2 operator=() [2/2]

```
CheckboxChoice& CheckboxChoice::operator= (
            CheckboxChoice && rhs )  [default]
```

Similarly to the move constructor, the move assignment-operator cannot induce choice-fighting so remains available for use.

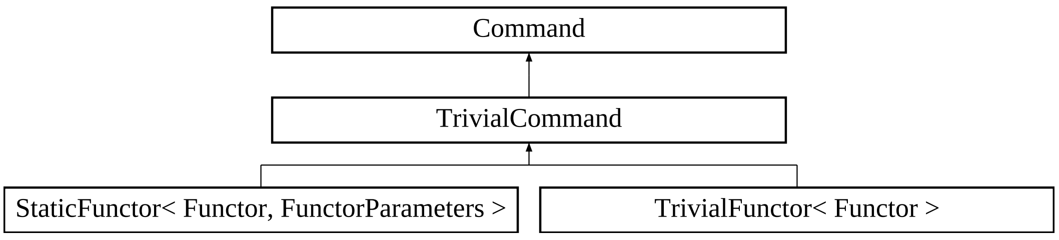
The documentation for this class was generated from the following files:

- src/graphics/gui_widget.hpp
- src/graphics/gui_widget.cpp

5.13 Command Class Reference

```
#include <command.hpp>
```

Inheritance diagram for Command:


Command
TrivialCommand
StaticFunctor< Functor, FunctorParameters >
TrivialFunctor< Functor >

Public Member Functions

- **Command** (std::string name="", std::string description="", std::string usage="", bool trivial=true)
- **Command** (const Command &copy)=default
- **Command** (Command &&move)=default
- Command & **operator=** (const Command &rhs)=default
- const std::string & **get_name** () const
- const std::string & **get_description** () const
- const std::string & **get_usage** () const
- std::size_t **get_expected_parameter_size** () const
- virtual bool **operator==** (const Command &rhs) const
- virtual void **operator()** (const std::vector< std::string > &args)=0
- bool **is_trivial** () const

Protected Attributes

- bool **trivial**

5.13.1 Detailed Description

Abstract. Not available for non-polymorphic use. Represents a hard-coded functor with string arguments. Inherit from this to create custom commands (Essential for adding functionality to Engine). For simpler functions with no parameters, use a TrivialCommand instead (see lower down this source file).

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

- src/command.hpp
- src/command.cpp

5.14 CommandExecutor Class Reference

```
#include <command.hpp>
```

Public Member Functions

- **CommandExecutor** (const CommandExecutor &copy)=default
- **CommandExecutor** (CommandExecutor &&move)=default
- CommandExecutor & **operator=** (const CommandExecutor &rhs)=default
- std::unordered_set< Command ∗ > **get_commands** () const
- void **register_command** (Command ∗command)
- template<typename Functor >
  TrivialFunctor< Functor > ∗ **emplace_trivial_command** (Functor &&functor)
- template<typename Functor , typename... FunctorParameters>
  StaticFunctor< Functor, FunctorParameters... > ∗ **emplace_static_command** (Functor &&functor, FunctorParameters &&... parameters)
- void **deregister_command** (Command ∗command)
- void **deregister_command** (const std::string &command_name)
- void **operator()** (const std::string &name, const std::vector< std::string > &args=std::vector< std::string >())

5.14.1 Detailed Description

System used to hold (but not typically own) Commands. Engine uses these to handle command input.

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

- src/command.hpp
- src/command.cpp
- src/command.inl

5.15 CubeMap Class Reference

```
#include <texture.hpp>
```

Public Member Functions

- **CubeMap** (std::string right_texture, std::string left_texture, std::string top_texture, std::string bottom_texture, std::string back_texture, std::string front_texture)
- **CubeMap** (std::string texture_directory="./", std::string skybox_name="skybox", std::string skybox_image↩_file_extension=".png")
- **CubeMap** (const CubeMap &copy)
- **CubeMap** (CubeMap &&move)
- CubeMap & **operator=** (const CubeMap &rhs)=delete
- void **bind** (Shader ∗shader, unsigned int id) const

5.15.1 Detailed Description

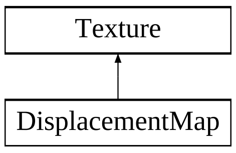
Used to construct skyboxes. Requires six textures; for each face of the skybox cube mesh.

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

- src/graphics/texture.hpp
- src/graphics/texture.cpp

5.16 DisplacementMap Class Reference

Inheritance diagram for DisplacementMap:



Public Member Functions

- **DisplacementMap** (std::string filename)
- virtual void **bind** (Shader ∗shader, unsigned int id) const override
- virtual tz::graphics::TextureType **get_texture_type** () const override

Additional Inherited Members

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

- src/graphics/texture.hpp

5.17 Engine Class Reference

```
#include <engine.hpp>
```

Public Member Functions

- Engine (Window ∗window, std::string properties_path=tz::default_properties_path, unsigned int tps=30)
- **Engine** (const Engine &copy)=default
- **Engine** (Engine &&move)=default
- Engine & **operator=** (const Engine &rhs)=default
- void update (std::size_t shader_index=0)
- const TimeProfiler & **get_time_profiler** () const
- const MDLFile & get_properties () const
- const MDLFile & get_resources () const
- const Window & get_window () const
- const std::vector< std::unique_ptr< Mesh > > & get_meshes () const
- const std::vector< std::unique_ptr< Texture > > & **get_textures** () const
- const std::vector< std::unique_ptr< NormalMap > > & **get_normal_maps** () const
- const std::vector< std::unique_ptr< ParallaxMap > > & **get_parallax_maps** () const
- const std::vector< std::unique_ptr< DisplacementMap > > & **get_displacement_maps** () const
- Shader & get_shader (std::size_t index)
- unsigned int get_fps () const
- unsigned int get_tps () const
- const CommandExecutor & get_update_command_executor () const
- const CommandExecutor & get_tick_command_executor () const
- void add_update_command (Command ∗cmd)
- template<typename Functor >
  TrivialFunctor< Functor > ∗ **emplace_trivial_update_command** (Functor &&functor)
- template<typename Functor , typename... FunctorParameters>
  StaticFunctor< Functor, FunctorParameters... > ∗ **emplace_static_update_command** (Functor &&functor, FunctorParameters &&... parameters)
- void remove_update_command (Command ∗cmd)
- void add_tick_command (Command ∗cmd)
- template<typename Functor >
  TrivialFunctor< Functor > ∗ **emplace_trivial_tick_command** (Functor &&functor)
- template<typename Functor , typename... FunctorParameters>
  StaticFunctor< Functor, FunctorParameters... > ∗ **emplace_static_tick_command** (Functor &&functor, FunctorParameters &&... parameters)
- void remove_tick_command (Command ∗cmd)
- bool is_update_due () const

Public Attributes

- Camera camera
- Scene **scene**
- Shader **default_shader**
- Shader **default_gui_shader**
- const Texture **default_texture**
- const NormalMap **default_normal_map**
- const ParallaxMap **default_parallax_map**
- const DisplacementMap **default_displacement_map**

5.17.1 Detailed Description

Hulking class holding pretty much everything you'll need to use Topaz. Due to its verbosity, it is only recommended to use this class for hobbyist/non-commercial purposes. Using this essentially provides stabilisers and handholding to using Topaz.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 Engine()

```
Engine::Engine (
             Window * window,
             std::string properties_path = tz::default_properties_path,
             unsigned int tps = 30 )
```

Constructs the Engine. Should be invoked after tz::initialise and Window construction. window = Address of the Topaz window to render into. properties_path = The absolute path to the Topaz properties file (normally called properties.mdl) tps = Number of tick updates per second (this affects runtime of physics etc, not rendering). Default is 30, although you can use less or more, depending on how precise you need physics etc to run at.

5.17.3 Member Function Documentation

5.17.3.1 add_tick_command()

```
void Engine::add_tick_command (
             Command * cmd )
```

Add a custom command to the tick command executor.

5.17.3.2 add_update_command()

```
void Engine::add_update_command (
             Command * cmd )
```

Add a custom command to the update comand executor.

5.17.3.3 get_fps()

```
unsigned int Engine::get_fps ( ) const
```

Get instantaneous fps

5.17.3.4 get_meshes()

```
const std::vector< std::unique_ptr< Mesh > > & Engine::get_meshes ( ) const
```

Access lists of all assets.

5.17.3.5 get_properties()

```
const MDLFile & Engine::get_properties ( ) const
```

Read/Edit the properties file.

5.17.3.6 get_resources()

```
const MDLFile & Engine::get_resources ( ) const
```

Read/Edit the resources file.

5.17.3.7 get_shader()

```
Shader & Engine::get_shader (
            std::size_t index )
```

Get shader by index. If index = 0 or is out of range of extra-shaders, will return the default shader. Otherwise, it shall return the extra shader at that index.

5.17.3.8 get_tick_command_executor()

```
const CommandExecutor & Engine::get_tick_command_executor ( ) const
```

Read the tick command executor (for physics updates etc)

5.17.3.9 get_tps()

```
unsigned int Engine::get_tps ( ) const
```

Get the specified number of ticks per second back when the instance was constructed.

5.17.3.10 get_update_command_executor()

```
const CommandExecutor & Engine::get_update_command_executor ( ) const
```

Read the update command executor (for rendering etc)

5.17.3.11 get_window()

```
const Window & Engine::get_window ( ) const
```

Access the window that the Engine instance currently is hooked to.

5.17.3.12 is_update_due()

```
bool Engine::is_update_due ( ) const
```

Returns true if a physics update will occur next update. Use-cases for this mainly include when you need to synchronise your own functionality with the physics updates (which you should really use add_tick_command(Command∗) for.)

5.17.3.13 remove_tick_command()

```
void Engine::remove_tick_command (
             Command ∗ cmd )
```

Remove a command from the tick command executor.

5.17.3.14 remove_update_command()

```
void Engine::remove_update_command (
             Command ∗ cmd )
```

Remove a command from the update command executor.

5.17.3.15 update()

```
void Engine::update (
             std::size_t shader_index = 0 )
```

Invoke this in your main application loop. For clarification of 'shader_index', see documentation for Engine::get_shader(std::size_t).

5.17.4 Member Data Documentation

5.17.4.1 camera

```
Camera Engine::camera
```

Editing fields of these public members is well-defined. But as far as Topaz is concerned, re-assigning them is unspecified behaviour.

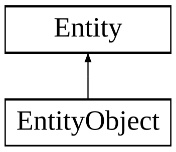
The documentation for this class was generated from the following files:

- src/engine.hpp
- src/engine.cpp
- src/engine.inl

5.18 Entity Class Reference

```
#include <entity.hpp>
```

Inheritance diagram for Entity:



Public Member Functions

- **Entity** (float mass=tz::physics::default_mass, Vector3F position=Vector3F(), Vector3F velocity=Vector3F(), std::unordered_map< std::string, Force > forces=std::unordered_map< std::string, Force >())
- **Entity** (const Entity &copy)=default
- **Entity** (Entity &&move)=default
- Entity & **operator=** (const Entity &rhs)=default
- Vector3F **get_acceleration** () const
- const std::unordered_map< std::string, Force > & **get_forces** () const
- void apply_force (std::string force_name, Force f)
- void remove_force (std::string force_name)
- virtual void **update_motion** (unsigned int fps)
- bool **operator==** (const Entity &rhs) const

Public Attributes

- float **mass**
- Vector3F **position**
- Vector3F **velocity**

Protected Attributes

- std::unordered_map< std::string, Force > **forces**

5.18.1 Detailed Description

Something which follows the rules of Newtonian Motion. Attach this to something you want to be able to experience motion and forces, such as gravity or thrust.

5.18.2 Member Function Documentation

5.18.2.1 apply_force()

```
void Entity::apply_force (
             std::string force_name,
             Force f )
```

Apply a force on this object, provided a name. Complexity: O(n) (1) (1), where n = number of existing forces.

5.18.2.2 remove_force()

```
void Entity::remove_force (
             std::string force_name )
```

Remove the force on this object with the specified name. Complexity: O(n) (1) (1), where n = number of existing forces

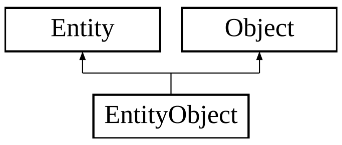
The documentation for this class was generated from the following files:

- src/entity.hpp
- src/entity.cpp

5.19 EntityObject Class Reference

```
#include <entity.hpp>
```

Inheritance diagram for EntityObject:



Public Member Functions

- EntityObject (const Mesh ∗mesh, Material material, float mass=tz::physics::default_mass, Vector3F position=Vector3F(), Vector3F rotation=Vector3F(), Vector3F scale=Vector3F(1, 1, 1), Vector3F velocity=Vector3F(), std::unordered_map< std::string, Force > forces=std::unordered_map< std::string, Force >())
- **EntityObject** (const Object &static_object, float mass)
- **EntityObject** (const EntityObject &copy)=default
- **EntityObject** (EntityObject &&move)=default
- EntityObject & **operator=** (const EntityObject &rhs)=default
- virtual void **update_motion** (unsigned int fps) override
- bool **operator==** (const EntityObject &rhs) const

Public Attributes

- Vector3F **position**

Additional Inherited Members

5.19.1 Detailed Description

Essentially an Entity which has a renderable component in the form of an Object3D. See Object3D documentation for additional details.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 EntityObject()

```
EntityObject::EntityObject (
            const Mesh * mesh,
            Material material,
            float mass = tz::physics::default_mass,
            Vector3F position = Vector3F(),
            Vector3F rotation = Vector3F(),
            Vector3F scale = Vector3F(1, 1, 1),
            Vector3F velocity = Vector3F(),
            std::unordered_map< std::string, Force > forces = std::unordered_map<std:↩
:string, Force>() )
```

Manual construction of a new EntityObject.

Parameters

mesh	- Pointer to read-only Mesh. This must outlive the EntityObject or UB will be invoked.
textures	- Map of each texture-type to the corresponding pointer to Texture. All must outlive the EntityObject or UB will be invoked.
mass	- Desired mass of the EntityObject.
position	- Position of the EntityObject, in world-space.
rotation	- Rotation of the EntityObject, in euler-angles.
scale	- Scale of the EntityObject.
velocity	
forces	

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

- src/entity.hpp
- src/entity.cpp

5.20 Font Class Reference

```
#include <graphics.hpp>
```

Public Member Functions

- **Font** (const std::string &font_path, int pixel_height)
- **Font** (const Font &copy)
- **Font** (Font &&move)
- Font & **operator=** (Font &&rhs)
- int **get_pixel_height** () const
- const std::string & **get_path** () const

Friends

- class **Texture**

5.20.1 Detailed Description

Used to render text. Texture has a constructor taking a Font as a parameter. Use this to achieve font-rendering.

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

- src/graphics/graphics.hpp
- src/graphics/graphics.cpp

5.21 Force Class Reference

```
#include <physics.hpp>
```

Public Member Functions

- **Force** (Vector3F size=Vector3F())
- **Force** (const Force &copy)=default
- **Force** (Force &&move)=default
- Force & **operator=** (const Force &rhs)=default
- Force **operator+** (const Force &other) const
- Force **operator-** (const Force &other) const
- Force **operator*** (float rhs) const
- Force **operator/** (float rhs) const
- Force & **operator+=** (const Force &other)
- Force & **operator-=** (const Force &other)
- bool **operator==** (const Force &other) const

Public Attributes

- Vector3F **size**

5.21.1 Detailed Description

Represent a physical force in three-dimensions.

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

- src/physics/physics.hpp
- src/physics/physics.cpp

5.22 FrameBuffer Class Reference

```
#include <texture.hpp>
```

Public Member Functions

- **FrameBuffer** (int width, int height)
- template< class Buffer , typename... Args>
  Buffer & emplace (GLenum attachment, Args &&... args)
- template< typename... Args>
  Texture & emplace_texture (GLenum attachment, Args &&... args)
- template< typename... Args>
  RenderBuffer & emplace_renderbuffer (GLenum attachment, Args &&... args)
- const std::unordered_map< GLenum, std::variant< Texture, RenderBuffer > > & get_attachments () const
- std::unordered_map< GLenum, std::reference_wrapper< const Texture > > **get_texture_attachments** () const
- bool valid () const
- bool has_colour (unsigned int attachment_index=0) const
- bool has_depth () const
- bool has_stencil () const
- void set_output_attachment (GLenum attachment) const
- void clear (GLbitfield mask=(GL_COLOR_BUFFER_BIT), float r=0.0f, float g=0.0f, float b=0.0f, float a=1.0f) const
- void set_render_target () const

5.22.1 Detailed Description

Something to draw to that isn't a window. FrameBuffer attachments can either be a Texture or a RenderBuffer.

5.22.2 Member Function Documentation

5.22.2.1 clear()

```
void FrameBuffer::clear (
             GLbitfield mask = (GL_COLOR_BUFFER_BIT),
             float r = 0.0f,
             float g = 0.0f,
             float b = 0.0f,
             float a = 1.0f ) const
```

Perform an OpenGL clear operation on the framebuffer.

5.22.2.2 emplace()

```
template<class Buffer , typename... Args>
Buffer & FrameBuffer::emplace (
             GLenum attachment,
             Args &&... args )
```

Build an instance of either Texture or RenderBuffer in-place into the framebuffer.

5.22.2.3 emplace_renderbuffer()

```
template<typename... Args>
RenderBuffer & FrameBuffer::emplace_renderbuffer (
             GLenum attachment,
             Args &&... args )
```

Build an instance of RenderBuffer in-place into the framebuffer.

5.22.2.4 emplace_texture()

```
template<typename... Args>
Texture & FrameBuffer::emplace_texture (
             GLenum attachment,
             Args &&... args )
```

Build an instance of Texture in-place into the framebuffer.

5.22.2.5 get_attachments()

```
const std::unordered_map< GLenum, std::variant< Texture, RenderBuffer > > & FrameBuffer↩
::get_attachments ( ) const
```

Read-only access to all attachments to this framebuffer.

5.22.2.6 has_colour()

```
bool FrameBuffer::has_colour (
             unsigned int attachment_index = 0 ) const
```

Returns true if this framebuffer has a Texture or RenderBuffer with the attachment GL_COLOR_ATTACHMENTx, where x == attachment_index.

5.22.2.7 has_depth()

```
bool FrameBuffer::has_depth ( ) const
```

Returns true if this framebuffer has a Texture or RenderBuffer with the attachment GL_DEPTH_ATTACHMENT.

5.22.2.8 has_stencil()

```
bool FrameBuffer::has_stencil ( ) const
```

Returns true if this framebuffer has a Texture or RenderBuffer with the attachment GL_STENCIL_ATTACHMENT.

5.22.2.9 set_output_attachment()

```
void FrameBuffer::set_output_attachment (
             GLenum attachment ) const
```

Make the output value of the fragment-shader write to an existing buffer with the corresponding attachment type (e.g specifying GL_COLOR_ATTACHMENT0 will write to the Texture/RenderBuffer applied to GL_COLOR_ATT↩ ACHMENT0).

5.22.2.10 set_render_target()

```
void FrameBuffer::set_render_target ( ) const
```

Bind and sets the viewpoint to this framebuffer. This means that any render calls will apply to this framebuffer.

5.22.2.11 valid()

```
bool FrameBuffer::valid ( ) const
```

Returns true if OpenGL sees the framebuffer as complete.

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

- src/graphics/texture.hpp
- src/graphics/texture.cpp
- src/graphics/texture.inl

5.23 Functor< FunctorT > Class Template Reference

```
#include <utility.hpp>
```

Public Member Functions

- **Functor** (FunctorT functor)
- template<typename... FunctorParameters>
  void **operator()** (FunctorParameters... parameters)

5.23.1 Detailed Description

template<typename FunctorT>
class Functor< FunctorT >

Wrapper for a function with variadic arguments. Unlike TrivialFunctor, cannot be inserted into a CommandExecutor.

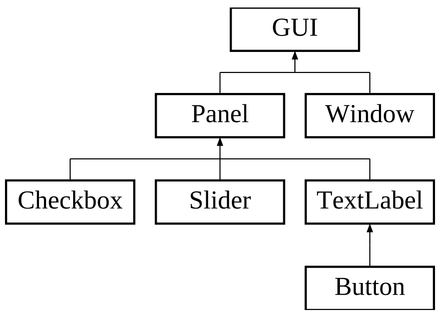
The documentation for this class was generated from the following files:

- src/utility.hpp
- src/utility.inl

5.24 GUI Class Reference

```
#include <gui.hpp>
```

Inheritance diagram for GUI:



Public Member Functions

- **GUI** (float x, float y, float width, float height, std::optional< std::reference_wrapper< Shader >> shader)
- **GUI** (const GUI &copy)=default
- **GUI** (GUI &&move)=default
- GUI & **operator=** (const GUI &rhs)=default
- virtual void **update** ()
- virtual void **destroy** ()
- virtual bool **focused** () const =0
- virtual bool **is_window** () const =0
- virtual bool **is_mouse_sensitive** () const
- virtual float **get_window_pos_x** () const
- virtual float **get_window_pos_y** () const
- float **get_x** () const
- float **get_y** () const

- virtual float **get_width** () const
- virtual float **get_height** () const
- void **set_x** (float x)
- void **set_y** (float y)
- void **set_width** (float width)
- void **set_height** (float height)
- Window ∗ **find_window_parent** () const
- bool **has_window_parent** () const
- const std::optional< std::reference_wrapper< Shader > > & **get_shader** () const
- bool **has_shader** () const
- GUI ∗ **get_parent** () const
- void **set_parent** (GUI ∗parent)
- const std::deque< GUI ∗ > & **get_children** () const
- void **add_child** (GUI ∗child)
- void **remove_child** (GUI ∗child)
- bool **is_hidden** () const
- virtual void **set_hidden** (bool hidden)
- void **set_using_proportional_positioning** (bool use_proportional_positioning)
- bool **is_using_proportional_positioning** () const
- GUI ∗ **covered_by** () const
- bool **covered** () const

Protected Attributes

- float **x**
- float **y**
- float **width**
- float **height**
- std::optional< std::reference_wrapper< Shader > > **shader**
- GUI ∗ **parent**
- std::deque< GUI ∗ > **children**
- bool **hidden**
- bool **use_proportional_positioning**

5.24.1 Detailed Description

Represents a Topaz GUI element. Abstract. Not available for non-polymorphic use. Inherit from this to create custom Topaz GUI yourself.

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

- src/graphics/gui.hpp
- src/graphics/gui.cpp

5.25 tz::graphics::model::IndexedModel Class Reference

Public Member Functions

- void **calculate_normals** ()
- void **calculate_tangents** ()

Public Attributes

- std::vector< Vector3F > **positions**
- std::vector< Vector2F > **texcoords**
- std::vector< Vector3F > **normals**
- std::vector< Vector3F > **tangents**
- std::vector< unsigned int > **indices**

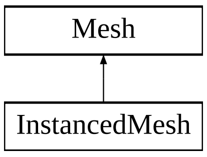
The documentation for this class was generated from the following files:

- src/graphics/graphics.hpp
- src/graphics/graphics.cpp

5.26 InstancedMesh Class Reference

```
#include <mesh.hpp>
```

Inheritance diagram for InstancedMesh:



Public Member Functions

- **InstancedMesh** (std::string filename, std::vector< Vector3F > positions, std::vector< Vector3F > rotations, std::vector< Vector3F > scales)
- **InstancedMesh** (const InstancedMesh &copy)=default
- **InstancedMesh** (InstancedMesh &&move)=default
- InstancedMesh & **operator=** (const InstancedMesh &rhs)=default
- const std::vector< Vector3F > & **get_instance_positions** () const
- const std::vector< Vector3F > & **get_instance_rotations** () const
- const std::vector< Vector3F > & **get_instance_scales** () const
- std::size_t **get_instance_quantity** () const
- virtual void **render** (bool patches, GLenum mode=GL_TRIANGLES) const override

Additional Inherited Members

5.26.1 Detailed Description

Like a normal mesh, but supports OpenGL instancing. Use this if you want to render the same mesh very many times at once with little attribute changes. This class is abstracted away by tz::graphics::batch in object.hpp.

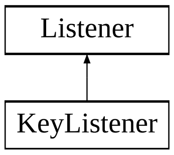
The documentation for this class was generated from the following files:

- src/graphics/mesh.hpp
- src/graphics/mesh.cpp

5.27 KeyListener Class Reference

```
#include <listener.hpp>
```

Inheritance diagram for KeyListener:



Public Member Functions

- **KeyListener** (Window &window)
- **KeyListener** (const KeyListener &copy)=default
- **KeyListener** (KeyListener &&move)=default
- KeyListener & **operator=** (const KeyListener &rhs)=default
- virtual void **handle_events** (SDL_Event &evt) override
- bool **is_key_pressed** (const std::string &keyname) const
- bool **is_key_released** (const std::string &keyname) const
- bool **catch_key_pressed** (const std::string &keyname)
- bool **catch_key_released** (const std::string &keyname)

Additional Inherited Members

5.27.1 Detailed Description

How Topaz handles keyboard input. Register this to a Topaz Window to use properly.

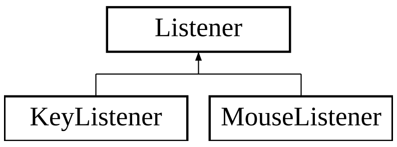
The documentation for this class was generated from the following files:

- src/listener.hpp
- src/listener.cpp

5.28 Listener Class Reference

```
#include <listener.hpp>
```

Inheritance diagram for Listener:



Public Member Functions

- **Listener** (const Listener &copy)=default
- **Listener** (Listener &&move)=default
- Listener & **operator=** (const Listener &rhs)=default
- virtual void **handle_events** (SDL_Event &evt)=0
- unsigned int **get_id** () const

Static Public Member Functions

- static unsigned int **get_num_listeners** ()

5.28.1 Detailed Description

Wrapper for an SDL_Event listener. Abstract. Not available for non-polymorphic use. Inherit from this to create custom listeners that a Topaz Window can register.

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

- src/listener.hpp
- src/listener.cpp

5.29 tz::data::Manager Class Reference

```
#include <data.hpp>
```

Public Member Functions

- **Manager** (std::string datafilename)
- **Manager** (const Manager &copy)=default
- **Manager** (Manager &&move)=default
- Manager & **operator=** (const Manager &rhs)=default
- std::string **resource_link** (const std::string &resource_name) const
- std::string **resource_name** (const std::string &resource_link) const
- std::unordered_map< std::string, std::string > **retrieve_models** (const char ∗sequence_name=tz::data↩ ::default_models_sequence_name) const
- std::unordered_map< std::string, std::string > **retrieve_textures** (const char ∗sequence_name=tz::data↩ ::default_textures_sequence_name) const
- std::unordered_map< std::string, std::string > **retrieve_normal_maps** (const char ∗sequence_name=tz↩ ::data::default_normal_maps_sequence_name) const
- std::unordered_map< std::string, std::string > **retrieve_parallax_maps** (const char ∗sequence_name=tz↩ ::data::default_parallax_maps_sequence_name) const
- std::unordered_map< std::string, std::string > **retrieve_displacement_maps** (const char ∗sequence_↩ name=tz::data::default_displacement_maps_sequence_name) const
- unsigned int **retrieve_all_data** (std::vector< std::unique_ptr< Mesh >> &all_meshes, std::vector< std::unique_ptr< Texture >> &all_textures, std::vector< std::unique_ptr< NormalMap >> &all_↩ normalmaps, std::vector< std::unique_ptr< ParallaxMap >> &all_parallaxmaps, std::vector< std::unique↩ _ptr< DisplacementMap >> &all_displacementmaps) const

5.29.1 Detailed Description

Essentially an MDL Asset Manager. This is how Topaz handles MDL data files which hold assets such as textures and models.

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

- src/data.hpp
- src/data.cpp

5.30 Material Class Reference

```
#include <material.hpp>
```

Public Member Functions

- **Material** (Texture ∗texture, NormalMap ∗normal_map=nullptr, ParallaxMap ∗parallax_map=nullptr, DisplacementMap ∗displacement_map=nullptr)
- bool has_texture () const
- const Texture ∗ **get_texture** () const
- bool **has_normal_map** () const
- const NormalMap ∗ **get_normal_map** () const
- bool **has_parallax_map** () const
- const ParallaxMap ∗ **get_parallax_map** () const
- bool **has_displacement_map** () const
- const DisplacementMap ∗ **get_displacement_map** () const
- void **set_texture** (Texture ∗texture)
- void **set_normal_map** (NormalMap ∗normal_map)
- void **set_parallax_map** (ParallaxMap ∗parallax_map)
- void **set_displacement_map** (DisplacementMap ∗displacement_map)
- virtual void **bind** (Shader &shader) const
- bool **operator==** (const Material &rhs) const

5.30.1 Detailed Description

Non-owning container for a tuple of Texture, NormalMap, ParallaxMap and DisplacementMap. Objects contain one of these for simplity's sake.

5.30.2 Member Function Documentation

5.30.2.1 has_texture()

```
bool Material::has_texture ( ) const
```

Returns true if the texture component is not null. Note: This will return false if the default-texture was manually passed into the constructor, but true if nullptr was passed and the default-texture was inferred.

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

- src/graphics/material.hpp
- src/graphics/material.cpp

5.31 Matrix2x2 Class Reference

```
#include <matrix.hpp>
```

Public Member Functions

- **Matrix2x2** (Vector2F x=Vector2F(1.0f, 0.0f), Vector2F y=Vector2F(0.0f, 1.0f))
- **Matrix2x2** (const Matrix2x2 &copy)=default
- **Matrix2x2** (Matrix2x2 &&move)=default
- Matrix2x2 & **operator=** (const Matrix2x2 &rhs)=default
- float **determinant** () const

Public Attributes

- Vector2F **x**
- Vector2F **y**

5.31.1 Detailed Description

Represents a two-dimensional square matrix.

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

- src/matrix.hpp
- src/matrix.cpp

5.32 Matrix3x3 Class Reference

```
#include <matrix.hpp>
```

Public Member Functions

- **Matrix3x3** (Vector3F x=Vector3F(1.0f, 0.0f, 0.0f), Vector3F y=Vector3F(0.0f, 1.0f, 0.0f), Vector3F z=Vector3↩ F(0.0f, 0.0f, 1.0f))
- **Matrix3x3** (const Matrix3x3 &copy)=default
- **Matrix3x3** (Matrix3x3 &&move)=default
- Matrix3x3 & **operator=** (const Matrix3x3 &rhs)=default
- float **determinant** () const

Public Attributes

- Vector3F **x**
- Vector3F **y**
- Vector3F **z**

5.32.1 Detailed Description

Represents a three-dimensional square matrix.

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

- src/matrix.hpp
- src/matrix.cpp

5.33 Matrix4x4 Class Reference

```
#include <matrix.hpp>
```

Public Member Functions

- **Matrix4x4** (Vector4F x=Vector4F(1.0f, 0.0f, 0.0f, 0.0f), Vector4F y=Vector4F(0.0f, 1.0f, 0.0f, 0.0f), Vector4F z=Vector4F(0.0f, 0.0f, 1.0f, 0.0f), Vector4F w=Vector4F(0.0f, 0.0f, 0.0f, 1.0f))
- **Matrix4x4** (const Matrix4x4 &copy)=default
- **Matrix4x4** (Matrix4x4 &&move)=default
- Matrix4x4 & **operator=** (const Matrix4x4 &rhs)=default
- Matrix4x4 **transposed** () const
- std::array< float, 16 > **fill_data** () const
- Matrix3x3 **sub_matrix** (float iter_i, float iter_j) const
- Vector4F **operator*** (const Vector4F &other) const
- Matrix4x4 **operator*** (const Matrix4x4 &other) const
- float **determinant** () const
- Matrix4x4 **inverse** () const

Static Public Member Functions

- static Matrix4x4 **identity** ()

Public Attributes

- Vector4F **x**
- Vector4F **y**
- Vector4F **z**
- Vector4F **w**

5.33.1 Detailed Description

Represents a four-dimensional square matrix.

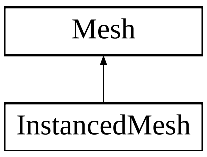
The documentation for this class was generated from the following files:

- src/matrix.hpp
- src/matrix.cpp

5.34 Mesh Class Reference

```
#include <mesh.hpp>
```

Inheritance diagram for Mesh:



Public Member Functions

- **Mesh** (std::string filename)
- **Mesh** (const Vertex ∗vertices, std::size_t number_of_vertices, const unsigned int ∗indices, std::size_↩
t number_of_indices)
- **Mesh** (const std::vector< Vertex > &vertices, const std::vector< unsigned int > &indices)
- **Mesh** (const Mesh &copy)=default
- **Mesh** (Mesh &&move)=default
- Mesh & **operator=** (const Mesh &rhs)=default
- tz::graphics::model::IndexedModel **get_indexed_model** () const
- const std::vector< Vector3F > & **get_positions** () const
- const std::vector< Vector2F > & **get_texcoords** () const
- const std::vector< Vector3F > & **get_normals** () const
- const std::vector< Vector3F > & **get_tangents** () const
- std::string **get_file_name** () const
- virtual void **render** (bool patches, GLenum mode=GL_TRIANGLES) const
- bool **operator==** (const Mesh &rhs) const

Protected Attributes

- const std::string **filename**
- tz::graphics::model::IndexedModel **model**
- GLuint **vertex_array_object**
- std::array< GLuint, static_cast< std::size_t >tz::graphics::BufferTypes::NUM_BUFFERS)> **vbo_buffers**
- unsigned int **render_count**

5.34.1 Detailed Description

Lowest-level renderable class that Topaz offers. All renderable Topaz classes such as Object3D contain these. Holds 3D vertex data, from a Wavefront OBJ model, for example. Use this if you have an existing shader you can use to draw manually.

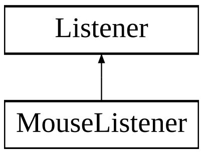
The documentation for this class was generated from the following files:

- src/graphics/mesh.hpp
- src/graphics/mesh.cpp

5.35 MouseListener Class Reference

```
#include <listener.hpp>
```

Inheritance diagram for MouseListener:



Public Member Functions

- **MouseListener** (Window &window)
- **MouseListener** (const MouseListener &copy)=default
- **MouseListener** (MouseListener &&move)=default
- MouseListener & **operator=** (const MouseListener &rhs)=default
- virtual void **handle_events** (SDL_Event &evt) override
- void **reload_mouse_delta** ()
- bool **is_left_clicked** () const
- bool **is_right_clicked** () const
- const Vector2F & **get_mouse_pos** () const
- Vector2F **get_mouse_delta_pos** () const
- const Vector2F & **get_left_click_location** () const
- const Vector2F & **get_right_click_location** () const

Additional Inherited Members

5.35.1 Detailed Description

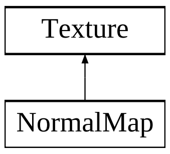
How Topaz handles mouse input. Register this to a Topaz Window to use properly.

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

- src/listener.hpp
- src/listener.cpp

5.36 NormalMap Class Reference

Inheritance diagram for NormalMap:



Public Member Functions

- **NormalMap** (std::string filename)
- virtual void **bind** (Shader ∗shader, unsigned int id) const override
- virtual tz::graphics::TextureType **get_texture_type** () const override

Additional Inherited Members

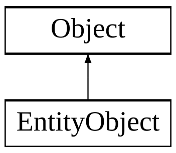
The documentation for this class was generated from the following file:

- src/graphics/texture.hpp

5.37 Object Class Reference

```
#include <object.hpp>
```

Inheritance diagram for Object:



Public Member Functions

- **Object** (std::variant< const Mesh ∗, std::shared_ptr< const Mesh >> mesh, Material material, Vector3F position, Vector3F rotation, Vector3F scale)
- **Object** (const Object &copy)=default
- **Object** (Object &&move)=default
- Object & **operator=** (const Object &rhs)=default
- const Mesh & **get_mesh** () const
- const Material & **get_material** () const
- virtual void render (const Camera &cam, Shader ∗shader, float width, float height) const
- bool **operator==** (const Object &rhs) const

Public Attributes

- Vector3F **position**
- Vector3F **rotation**
- Vector3F **scale**

Protected Attributes

- std::variant< const Mesh ∗, std::shared_ptr< const Mesh > > **mesh**
- Material **material**

5.37.1 Detailed Description

Collaboration of a mesh, texture, normal-map, parallax-map and displacement-map Use this to represent a 3D object, including its vertex data, texture, material etc.

5.37.2 Member Function Documentation

5.37.2.1 render()

```
void Object::render (
             const Camera & cam,
             Shader * shader,
             float width,
             float height ) const  [virtual]
```

Complexity: O(n) (1) (n), where n =∼ size of mesh data (will not return until GPU finishes processing)

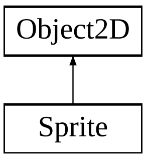
The documentation for this class was generated from the following files:

- src/graphics/object.hpp
- src/graphics/object.cpp

5.38 Object2D Class Reference

```
#include <object.hpp>
```

Inheritance diagram for Object2D:



Public Member Functions

- **Object2D** (Vector2F position, float rotation, Vector2F scale, Vector4F colour=Vector4F(0.0f, 0.0f, 0.0f, 1.0f))
- Object2D & **operator=** (const Object2D &copy)
- virtual void **render** (const Camera &cam, Shader ∗shader, float width, float height) const

Public Attributes

- Vector2F **position**
- Vector2F **scale**
- float **rotation**
- Vector4F **colour**

Protected Attributes

- Mesh **quad**

5.38.1 Detailed Description

Simple plane mesh with a colour to represent a simple-sprite.

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

- src/graphics/object.hpp
- src/graphics/object.cpp

5.39 tz::graphics::model::OBJIndex Class Reference

Public Member Functions

- bool **operator<** (const OBJIndex &rhs) const

Public Attributes

- unsigned int **vertex_index**
- unsigned int **uv_index**
- unsigned int **normal_index**

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

- src/graphics/graphics.hpp

5.40 tz::graphics::model::OBJModel Class Reference

Public Member Functions

- **OBJModel** (const std::string &file_name)
- IndexedModel **to_indexed_model** ()

Public Attributes

- std::vector< OBJIndex > **obj_indices**
- std::vector< Vector3F > **vertices**
- std::vector< Vector2F > **uvs**
- std::vector< Vector3F > **normals**
- bool **has_uvs**
- bool **has_normals**

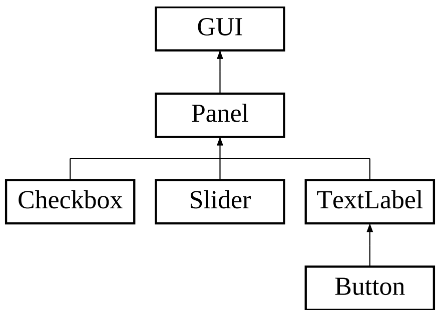
The documentation for this class was generated from the following files:

- src/graphics/graphics.hpp
- src/graphics/graphics.cpp

5.41 Panel Class Reference

```
#include <gui_display.hpp>
```

Inheritance diagram for Panel:



Public Member Functions

- **Panel** (float x, float y, float width, float height, Vector4F colour, Shader &shader)
- **Panel** (const Panel &copy)=default
- **Panel** (Panel &&move)=default
- Panel & **operator=** (const Panel &rhs)=default
- const Vector4F & **get_colour** () const
- void **set_colour** (Vector4F colour)
- const Texture ∗ **get_texture** () const
- void **set_texture** (Texture ∗texture)
- void **disable_texture** ()
- bool **has_texture** () const
- virtual void **update** () override
- virtual void **destroy** () override
- virtual bool **focused** () const override
- virtual bool **is_window** () const override
- virtual bool **is_mouse_sensitive** () const override

Protected Member Functions

- void **render_panel** (Vector4F colour, bool update=true)

Protected Attributes

- Texture ∗ **texture**
- Vector4F **colour**
- Mesh **quad**

5.41.1 Detailed Description

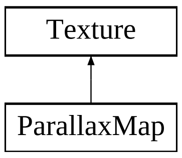
A 2D plane rendered on the screen. Can contain any other gui element in its own region. Can have a texture bound, but does not by default.

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

- src/graphics/gui_display.hpp
- src/graphics/gui_display.cpp

5.42 ParallaxMap Class Reference

Inheritance diagram for ParallaxMap:



Public Member Functions

- **ParallaxMap** (std::string filename)
- virtual void **bind** (Shader ∗shader, unsigned int id) const override
- virtual tz::graphics::TextureType **get_texture_type** () const override

Additional Inherited Members

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

- src/graphics/texture.hpp

5.43 PixelRGBA Class Reference

```
#include <graphics.hpp>
```

Public Member Functions

- constexpr **PixelRGBA** (unsigned char red=0, unsigned char green=0, unsigned char blue=0, unsigned char alpha=0)

Public Attributes

- Vector4< unsigned char > **data**

5.43.1 Detailed Description

Representation of Pixel Data in RGBA format.

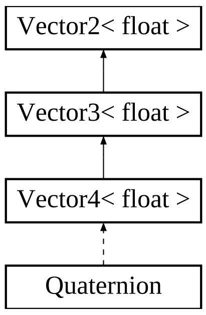
The documentation for this class was generated from the following file:

- src/graphics/graphics.hpp

5.44 Quaternion Class Reference

```
#include <quaternion.hpp>
```

Inheritance diagram for Quaternion:



Public Member Functions

- Quaternion (Vector3F rotation_axis=Vector3F(), float angle=0.0f)
- Quaternion (Vector3F euler_rotation)
- Quaternion (Matrix4x4 rotational_matrix)
- float **get_angle** () const
- Vector3F **get_rotation_axis** () const
- Matrix4x4 to_matrix () const
- Quaternion **normalised** () const
- Quaternion inverse () const
- Quaternion operator- () const
- Matrix4x4 operator() () const
- operator Matrix4x4 () const
- Quaternion operator∗ (const Quaternion &rhs) const
- Quaternion **operator**∗ (float scalar) const
- Quaternion **operator**/ (float scalar) const
- Vector4F operator∗ (const Vector3F &vector) const

Additional Inherited Members

5.44.1 Detailed Description

Represents 3D rotations. Implicitly convertible to a Matrix4x4 representing a rotational-matrix.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 Quaternion() [1/3]

```
Quaternion::Quaternion (
            Vector3F rotation_axis = Vector3F(),
            float angle = 0.0f )
```

Construct Quaternion from a rotation axis and an angle about the axis.

5.44.2.2 Quaternion() [2/3]

```
Quaternion::Quaternion (
            Vector3F euler_rotation )
```

Construct Quaternion from three rotations in euler angles about the three spatial dimensions.

5.44.2.3 Quaternion() [3/3]

```
Quaternion::Quaternion (
            Matrix4x4 rotational_matrix )
```

Construct Quaternion from an existing rotational matrix.

5.44.3 Member Function Documentation

5.44.3.1 inverse()

```
Quaternion Quaternion::inverse ( ) const
```

Just the same as the conjugate of the normalised.

5.44.3.2 operator Matrix4x4()

```
Quaternion::operator Matrix4x4 ( ) const
```

Enable implicit/explicit conversion to the Matrix4x4 component (via Quaternion::to_matrix())

5.44.3.3 operator()()

```
Matrix4x4 Quaternion::operator() ( ) const
```

Quaternion functor returns the Matrix (via Quaternion::to_matrix())

5.44.3.4 operator∗() [1/2]

```
Quaternion Quaternion::operator∗ (
             const Quaternion & rhs ) const
```

Combine quaternion rotations.

5.44.3.5 operator∗() [2/2]

```
Vector4F Quaternion::operator∗ (
             const Vector3F & vector ) const
```

Rotate a vector by a quaternion. Does not work for homogeneous coordinates; convert to a rotational matrix if you need to do that.

5.44.3.6 operator-()

```
Quaternion Quaternion::operator- ( ) const
```

Unary operator- is the same as getting the conjugate of the Quaternion (reversing the polarity AKA getting opposite direction of the original rotation axis)

5.44.3.7 to_matrix()

```
Matrix4x4 Quaternion::to_matrix ( ) const
```

Convert the Quaternion to a row-major rotational matrix.

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

- src/quaternion.hpp
- src/quaternion.cpp

5.45 Random< Engine, EngineResultType > Class Template Reference

```
#include <utility.hpp>
```

Public Member Functions

- **Random** (EngineResultType seed=std::random_device()())
- **Random** (const Random< Engine, EngineResultType > &copy)
- **Random** (Random< Engine, EngineResultType > &&move)=default
- Random< Engine, EngineResultType > & **operator=** (const Random< Engine, EngineResultType > &rhs)=default
- const EngineResultType & **get_seed** () const
- const Engine & **get_engine** () const
- int **next_int** (int min=0, int max=std::numeric_limits< int >::max())
- float **next_float** (float min=0, float max=std::numeric_limits< float >::max())
- template<typename Number = int>
  Number **operator()** (Number min=Number(), Number max=std::numeric_limits< Number >::max())

5.45.1 Detailed Description

template<typename Engine = std::default_random_engine, typename EngineResultType = std::default_random_engine::result↩_type>
class Random< Engine, EngineResultType >

Generate a random number using any of the C++ standard library random engines. Using default template arguments yields implementation-defined behaviour, but normally is a linear-congruentional engine.

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

- src/utility.hpp
- src/utility.inl

5.46 RenderBuffer Class Reference

```
#include <texture.hpp>
```

Public Member Functions

- **RenderBuffer** (int width, int height, GLenum internal_format=GL_RGBA)
- RenderBuffer (const RenderBuffer &copy)=delete
- **RenderBuffer** (RenderBuffer &&move)
- RenderBuffer & operator= (const RenderBuffer &rhs)=delete

Friends

- class **FrameBuffer**

5.46.1 Detailed Description

Simple wrapper for an OpenGL RenderBuffer. It's just a POD as they're write-only.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 RenderBuffer()

```
RenderBuffer::RenderBuffer (
            const RenderBuffer & copy )  [delete]
```

OpenGL RenderBuffers are write-only, so cannot possibly read the data in which to copy or move.

5.46.3 Member Function Documentation

5.46.3.1 operator=()

```
RenderBuffer& RenderBuffer::operator= (
            const RenderBuffer & rhs )  [delete]
```

RenderBuffer::operator= shall act like a pointer-copy; both share the same handle. However, when one dies the other becomes invalid, so this will be deleted too.

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

- src/graphics/texture.hpp
- src/graphics/texture.cpp

5.47 Scene Class Reference

```
#include <scene.hpp>
```

Public Member Functions

- Scene ()
- Scene (std::string filename, std::string resources_path, const std::vector< std::unique_ptr< Mesh >> &all↩
_meshes, const std::vector< std::unique_ptr< Texture >> &all_textures, const std::vector< std::unique↩
_ptr< NormalMap >> &all_normal_maps, const std::vector< std::unique_ptr< ParallaxMap >> &all_↩
parallax_maps, const std::vector< std::unique_ptr< DisplacementMap >> &all_displacement_maps, bool batch=true)
- **Scene** (const Scene &copy)=default
- **Scene** (Scene &&move)=default
- Scene & **operator=** (const Scene &rhs)=default
- bool has_file_name () const
- const std::string & get_file_name () const
- void **add_object** (Object obj)
- void **add_entity** (Entity ent)
- void **add_entity_object** (EntityObject eo)
- template<class Element , typename... Args>
  Element & emplace (Args &&... args)
- template<typename... Args>
  Object & emplace_object (Args &&... args)
- template<typename... Args>
  Entity & emplace_entity (Args &&... args)
- template<typename... Args>
  EntityObject & emplace_entity_object (Args &&... args)
- void remove_object (const Object &obj)
- void remove_entity (const Entity &ent)
- void remove_entity_object (const EntityObject &eo)
- const std::vector< Object > & get_objects () const
- const std::vector< Entity > & get_entities () const
- const std::vector< EntityObject > & get_entity_objects () const
- std::size_t get_size () const
- void export_scene (const std::string &scene_link) const
- void save () const
- void render (const Camera &cam, Shader ∗shader, unsigned int width, unsigned int height)
- void update (unsigned int tps)

Public Attributes

- Vector3F **spawn_point**
- Vector3F **spawn_orientation**

5.47.1 Detailed Description

Contains Objects, EntityObjects and pretty much any Topaz renderable you can think of. Use Scenes to store everything you want to draw.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 Scene() [1/2]

```
Scene::Scene ( )
```

Construct an empty scene.

5.47.2.2 Scene() [2/2]

```
Scene::Scene (
             std::string filename,
             std::string resources_path,
             const std::vector< std::unique_ptr< Mesh >> & all_meshes,
             const std::vector< std::unique_ptr< Texture >> & all_textures,
             const std::vector< std::unique_ptr< NormalMap >> & all_normal_maps,
             const std::vector< std::unique_ptr< ParallaxMap >> & all_parallax_maps,
             const std::vector< std::unique_ptr< DisplacementMap >> & all_displacement_maps,
             bool batch = true )
```

Load a scene from an existing MDL file. Takes in all asset vectors. Should probably be replaced with just a const asset manager reference of some kind to read the data without all this verbosity.

5.47.3 Member Function Documentation

5.47.3.1 emplace()

```
template<class Element , typename... Args>
Element & Scene::emplace (
             Args &&... args )
```

Construct an Object3D, Entity or EntityObject3D in-place and add it to the scene.

5.47.3.2 emplace_entity()

```
template<typename... Args>
Entity & Scene::emplace_entity (
             Args &&... args )
```

Construct an Entity in-place and add it to the scene.

5.47.3.3 emplace_entity_object()

```
template<typename... Args>
EntityObject & Scene::emplace_entity_object (
             Args &&... args )
```

Construct an EntityObject3D in-place and add it to the scene.

5.47.3.4 emplace_object()

```
template<typename... Args>
Object & Scene::emplace_object (
             Args &&... args )
```

Construct an Object3D in-place and add it to the scene.

5.47.3.5 export_scene()

```
void Scene::export_scene (
             const std::string & scene_link ) const
```

Export scene data to a MDL file called scene_link Complexity: O(n + m) (1) (n + m), where n = number of objects and m = number of entity_objects.

5.47.3.6 get_entities()

```
const std::vector< Entity > & Scene::get_entities ( ) const
```

Access all Entity elements in the scene (Read-only).

5.47.3.7 get_entity_objects()

```
const std::vector< EntityObject > & Scene::get_entity_objects ( ) const
```

Access all EntityObject3D elements in the scene (Read-only).

5.47.3.8 get_file_name()

```
const std::string & Scene::get_file_name ( ) const
```

Throws a std::bad_optional_access exception if this->has_file_name() returns false.

5.47.3.9 get_objects()

```
const std::vector< Object > & Scene::get_objects ( ) const
```

Access all Object3D elements in the scene (Read-only).

5.47.3.10 get_size()

```
std::size_t Scene::get_size ( ) const
```

Returns total number of Object3Ds, Entities and EntityObject3Ds in the scene.

5.47.3.11 has_file_name()

```
bool Scene::has_file_name ( ) const
```

Returns true if the scene was loaded from an external file.

5.47.3.12 remove_entity()

```
void Scene::remove_entity (
             const Entity & ent )
```

Remove an existing Entity from the scene.

5.47.3.13 remove_entity_object()

```
void Scene::remove_entity_object (
             const EntityObject & eo )
```

Remove an existing EntityObject3D from the scene.

5.47.3.14 remove_object()

```
void Scene::remove_object (
             const Object & obj )
```

Remove an existing Object3D from the scene.

5.47.3.15 render()

```
void Scene::render (
             const Camera & cam,
             Shader * shader,
             unsigned int width,
             unsigned int height )
```

Render all elements in the scene from the perspective of the camera, attaching a shader and updating uniforms in the process. This method should be invoked as often as possible, to smooth gameplay. Complexity: O(n + p) (1) (n + p), where n = number of objects, p = number of entity_objects.

5.47.3.16 save()

```
void Scene::save ( ) const
```

Export scene data to a MDL file with the same name as the file which loaded this scene, overwriting it. Complexity: See Scene::export_scene.

5.47.3.17 update()

```
void Scene::update (
             unsigned int tps )
```

Update all elements in the scene that obey some form of law of physics. Pass tps as the expected ticks-per-second, not the instantaneous tick per second. This function should be called per 'tick'. Complexity: O(n + m) (1) (n + m), where n = number of entity_objects, m = number of entities.

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

- src/graphics/scene.hpp
- src/graphics/scene.cpp
- src/graphics/scene.inl

5.48 Shader Class Reference

```
#include <shader.hpp>
```

Public Member Functions

- Shader (std::string vertex_source, std::string tessellation_control_source, std::string tessellation_↩ evaluation_source, std::string geometry_source, std::string fragment_source, bool compile=true, bool link=true, bool validate=true)
- Shader (std::string filename, bool compile=true, bool link=true, bool validate=true)
- **Shader** (const Shader &copy)
- **Shader** (Shader &&move)
- Shader & operator= (const Shader &rhs)=delete
- void **compile** (std::string vertex_source, std::string tessellation_control_source, std::string tessellation_↩ evaluation_source, std::string geometry_source, std::string fragment_source)
- void **link** ()
- void **validate** ()
- bool **is_compiled** () const
- bool **is_linked** () const
- bool **is_validated** () const
- bool ready () const
- template<class T >
  void **add_uniform** (Uniform< T > &&uniform)
- template<class T >
  void **emplace_uniform** (std::string uniform_location, T value)
- void **remove_uniform** (std::string_view uniform_location)
- bool **has_uniform** (std::string_view uniform_location) const
- UniformImplicit ∗ **get_uniform** (std::string_view uniform_location) const

- template<class T >
  void **set_uniform** (std::string_view uniform_location, T value)
- template<class T >
  T **get_uniform_value** (std::string_view uniform_location) const
- std::size_t **number_active_uniforms** () const
- const std::string & **get_attribute_location** (std::size_t attribute_id) const
- void **register_attribute** (std::size_t attribute_id, std::string attribute_location)
- bool **has_vertex_shader** () const
- bool **has_tessellation_control_shader** () const
- bool **has_tessellation_evaluation_shader** () const
- bool **has_geometry_shader** () const
- bool **has_fragment_shader** () const
- GLuint **get_program_handle** () const
- void **bind** () const
- void **update** () const

5.48.1 Detailed Description

Use this to load, compile, link, run, edit, analyse and even receive transform-feedback from an OpenGL shader written in GLSL.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 Shader() [1/2]

```
Shader::Shader (
            std::string vertex_source,
            std::string tessellation_control_source,
            std::string tessellation_evaluation_source,
            std::string geometry_source,
            std::string fragment_source,
            bool compile = true,
            bool link = true,
            bool validate = true )
```

Constructs a shader from the given shader-sources. Invalid shader sources will emit errors via tz::util::log::error. Valid shader sources will yield desired shaders. Empty shader sources will yield a lack of corresponding shaders (e.g if geometry_source is "", there shall be no geometry shader).

5.48.2.2 Shader() [2/2]

```
Shader::Shader (
            std::string filename,
            bool compile = true,
            bool link = true,
            bool validate = true )
```

Constructs a shader from a given filename, comprised of: Vertex shader from the path '.vertex.glsl', Tessellation-Control shader from the path '.tessellation_control.glsl', Tessellation-Evaluation shader from the path '.tessellation_evaluation.glsl', Geometry shader from the path '.geometry.glsl', and Fragment shader from the path '.fragment.glsl'.

5.48.3 Member Function Documentation

5.48.3.1 operator=()

```
Shader& Shader::operator= (
            const Shader & rhs )  [delete]
```

Until transform feedback and other compilation options are implemented, the idea of copy-constructing Shaders is meaningless.

5.48.3.2 ready()

```
bool Shader::ready ( ) const
```

Returns true if the Shader is compiled, linked and validated and therefore ready to be bound.

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

- src/graphics/shader.hpp
- src/graphics/shader.cpp
- src/graphics/shader.inl

5.49 Skybox Class Reference

```
#include <object.hpp>
```

Public Member Functions

- **Skybox** (std::string cube_mesh_link, CubeMap &cm)
- **Skybox** (const Skybox &copy)=default
- **Skybox** (Skybox &&move)=default
- Skybox & **operator=** (const Skybox &rhs)=default
- void **render** (const Camera &cam, Shader &shad, const std::vector< std::unique_ptr< Mesh >> &all_↩ meshes, float width, float height)

5.49.1 Detailed Description

Wraps an OpenGL cubemap via a set of six textures. Use this to render skyboxes in a 3D world easily. Bring your own skybox shader though (Default one provided with Topaz is called 'skybox').

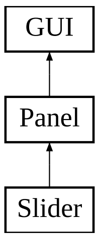
The documentation for this class was generated from the following files:

- src/graphics/object.hpp
- src/graphics/object.cpp

5.50 Slider Class Reference

```
#include <gui_widget.hpp>
```

Inheritance diagram for Slider:



Public Member Functions

- **Slider** (float x, float y, float width, float height, Vector4F slider_colour, Vector4F background_colour, Vector2F slider_size, Shader &shader, MouseListener &mouse_listener, float position=0.0f)
- **Slider** (const Slider &copy)=default
- **Slider** (Slider &&move)=default
- Slider & **operator=** (const Slider &rhs)=default
- virtual void **update** () override
- virtual bool **focused** () const override
- virtual bool **is_mouse_sensitive** () const override
- bool **moused_over** () const
- bool **clicked_on** () const
- const Vector4F & **get_slider_colour** () const
- const Vector2F & **get_slider_size** () const

Public Attributes

- double **position**

Additional Inherited Members

5.50.1 Detailed Description

Graphical representation of a mutable double. Use this to enable user-input for editing continuous data.

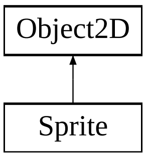
The documentation for this class was generated from the following files:

- src/graphics/gui_widget.hpp
- src/graphics/gui_widget.cpp

5.51 Sprite Class Reference

```
#include <sprite.hpp>
```

Inheritance diagram for Sprite:



Public Member Functions

- **Sprite** (Vector2F position, float rotation, Vector2F scale, const Texture ∗texture)
- **Sprite** (Vector2F position, float rotation, Vector2F scale, Vector4F colour)
- Sprite & **operator=** (const Sprite &copy)
- virtual void **render** (const Camera &cam, Shader ∗shader, float width, float height) const override

Additional Inherited Members

5.51.1 Detailed Description

Like a normal Object2D, but has a texture bound.

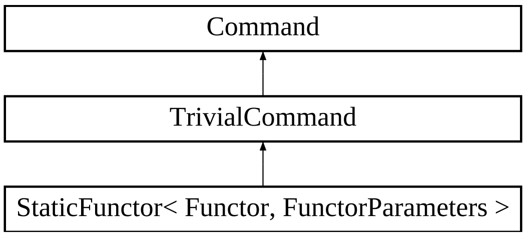
The documentation for this class was generated from the following files:

- src/graphics/sprite.hpp
- src/graphics/sprite.cpp

5.52 StaticFunctor< Functor, FunctorParameters > Class Template Reference

```
#include <command.hpp>
```

Inheritance diagram for StaticFunctor< Functor, FunctorParameters >:



Public Member Functions

- **StaticFunctor** (Functor &&, FunctorParameters &&... parameters)
- virtual void **operator()** () override

Protected Attributes

- Functor **functor**
- std::tuple< FunctorParameters... > **parameters**

5.52.1 Detailed Description

template<typename Functor, typename... FunctorParameters>
class StaticFunctor< Functor, FunctorParameters >

Wrapper for a functor which takes arguments, but the arguments are forwarded upon construction. If the functor is trivially-invokable (no arguments), then consider using TrivialFunctor. If the functor is not trivial-invokable and the arguments must vary, consider using the Functor class.

Template Parameters

Functor	- Functor type, normally an anonymous class (lambda)
FunctorParameters	- Parameters to be forwarded to the StaticFunctor.

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

- src/command.hpp
- src/command.inl

5.53 stbi_io_callbacks Struct Reference

Public Attributes

- int(∗ **read**)(void ∗user, char ∗data, int size)
- void(∗ **skip**)(void ∗user, int n)
- int(∗ **eof**)(void ∗user)

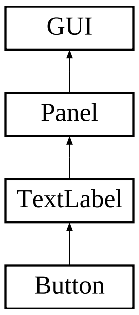
The documentation for this struct was generated from the following file:

- src/graphics/stb_image.h

5.54 TextLabel Class Reference

```
#include <gui_display.hpp>
```

Inheritance diagram for TextLabel:



Public Member Functions

- **TextLabel** (float x, float y, Vector4F colour, std::optional< Vector4F > background_colour, std::optional< Vector3F > text_border_colour, Font font, const std::string &text, Shader &shader)
- **TextLabel** (const TextLabel &copy)=default
- **TextLabel** (TextLabel &&move)=default
- TextLabel & **operator=** (const TextLabel &rhs)=default
- virtual void **update** () override
- bool **has_background_colour** () const
- bool **has_text_border_colour** () const
- const Font & **get_font** () const
- void **set_font** (Font font)
- const std::string & **get_text** () const
- void **set_text** (const std::string &new_text)
- const Texture & **get_texture** () const
- void **set_texture** (Texture texture)

Additional Inherited Members

5.54.1 Detailed Description

Very similar to a Panel, but has additional font-rendering applied. Use this to write text to the screen.

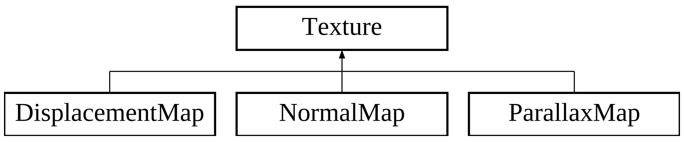
The documentation for this class was generated from the following files:

- src/graphics/gui_display.hpp
- src/graphics/gui_display.cpp

5.55 Texture Class Reference

`#include <texture.hpp>`

Inheritance diagram for Texture:



Public Member Functions

- Texture ()
- Texture (int width, int height)
- Texture (std::string filename, bool mipmapping=true, bool gamma_corrected=true, bool store_bitmap=false)
- template<class Pixel >
  Texture (Bitmap< Pixel > pixel_data)
- Texture (const Font &font, const std::string &text, SDL_Color foreground_colour, bool store_bitmap=false)
- **Texture** (const Texture &copy)
- **Texture** (Texture &&move)
- Texture & **operator=** (Texture &&rhs)
- virtual void **bind** (Shader ∗shader, unsigned int id) const
- bool **has_file_name** () const
- const std::string & **get_file_name** () const
- int **get_width** () const
- int **get_height** () const
- bool **has_bitmap** () const
- tz::graphics::MipmapType **get_mipmap_type** () const
- bool **has_mipmap** () const
- Bitmap< PixelRGBA > **get_bitmap** () const
- virtual tz::graphics::TextureType **get_texture_type** () const
- bool **operator==** (const Texture &rhs) const

Static Public Member Functions

- template<class T >
  static T ∗ **get_from_link** (const std::string &texture_link, const std::vector< std::unique_ptr< T >> &all_↩
  textures)

Protected Member Functions

- unsigned char ∗ **load_texture** ()
- void **delete_texture** (unsigned char ∗imgdata)
- void **bind_with_string** (Shader ∗shader, unsigned int id, const std::string &sampler_uniform_name) const

Protected Attributes

- std::optional< std::string > **filename**
- GLuint **texture_handle**
- int **width**
- int **height**
- int **components**
- bool **gamma_corrected**
- std::optional< Bitmap< PixelRGBA > > **bitmap**

Friends

- class **FrameBuffer**

5.55.1 Detailed Description

Holds pixel and colour data and can interact with OpenGL buffers. Bind Textures so that Topaz Meshes do not render monochromoatically.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 Texture() [1/5]

```
Texture::Texture ( )
```

Creates an uninitialised texture. This allocates a texture-handle but nothing else, so is not ready for rendering.

5.55.2.2 Texture() [2/5]

```
Texture::Texture (
            int width,
            int height )
```

Creates a completely empty texture, but would be ready to be written to, if bound to a framebuffer.

5.55.2.3 Texture() [3/5]

```
Texture::Texture (
            std::string filename,
            bool mipmapping = true,
            bool gamma_corrected = true,
            bool store_bitmap = false )
```

Loads a texture from a file.

5.55.2.4 Texture() [4/5]

```
template<class Pixel >
Texture::Texture (
            Bitmap< Pixel > pixel_data )
```

Loads a texture from existing Pixel Data

5.55.2.5 Texture() [5/5]

```
Texture::Texture (
            const Font & font,
            const std::string & text,
            SDL_Color foreground_colour,
            bool store_bitmap = false )
```

Loads a texture from a font, given text.

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

- src/graphics/texture.hpp
- src/graphics/texture.cpp
- src/graphics/texture.inl

5.56 TimeProfiler Class Reference

```
#include <time.hpp>
```

Public Member Functions

- **TimeProfiler** (const TimeProfiler &copy)=default
- **TimeProfiler** (TimeProfiler &&move)=default
- TimeProfiler & **operator=** (const TimeProfiler &rhs)=default
- void begin_frame ()
- void end_frame ()
- void reset ()
- float get_delta_average () const
- float get_last_delta () const
- unsigned int get_fps () const

5.56.1 Detailed Description

Specialised Timer that can be used to calculate FPS during runtime.

5.56.2 Member Function Documentation

5.56.2.1 begin_frame()

```
void TimeProfiler::begin_frame ( )
```

Invoke this at the beginning of your frame construction in the game-loop.

5.56.2.2 end_frame()

```
void TimeProfiler::end_frame ( )
```

Invoke this at the end of your frame construction in the game-loop.

5.56.2.3 get_delta_average()

```
float TimeProfiler::get_delta_average ( ) const
```

Returns the average time taken between each frame.

5.56.2.4 get_fps()

```
unsigned int TimeProfiler::get_fps ( ) const
```

Returns the average number of frames processed per second.

5.56.2.5 get_last_delta()

```
float TimeProfiler::get_last_delta ( ) const
```

Returns the time taken for the most recent frame.

5.56.2.6 reset()

```
void TimeProfiler::reset ( )
```

Purges all existing time-deltas from the delta-vector. Invoke this to reset the fps-counter.

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

- src/time.hpp
- src/time.cpp

5.57 Timer Class Reference

```
#include <time.hpp>
```

Public Member Functions

- **Timer** (const Timer &copy)=default
- **Timer** (Timer &&move)=default
- Timer & **operator=** (const Timer &rhs)=default
- void update ()
- void **reload** ()
- float get_range () const
- bool millis_passed (float millis) const

5.57.1 Detailed Description

Use this to schedule, record time or pretty much do anything that requires timing.

5.57.2 Member Function Documentation

5.57.2.1 get_range()

```
float Timer::get_range ( ) const
```

Returns the time taken between the last invocation of Timer::update and Timer::reload.

5.57.2.2 millis_passed()

```
bool Timer::millis_passed (
             float millis ) const
```

Returns whether Timer::get_range returns a time >= the

Parameters

millis.	

5.57.2.3 update()

```
void Timer::update ( )
```

Invoke every frame, so that Timer::get_range and Timer::millis_passed return accurate values.

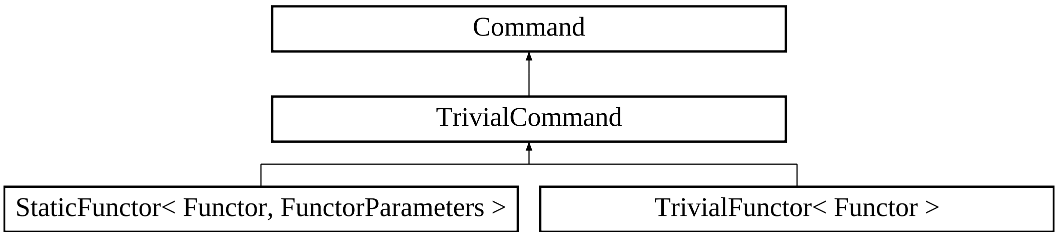
The documentation for this class was generated from the following files:

- src/time.hpp
- src/time.cpp

5.58 TrivialCommand Class Reference

```
#include <command.hpp>
```

Inheritance diagram for TrivialCommand:


Command
TrivialCommand
StaticFunctor< Functor, FunctorParameters >
TrivialFunctor< Functor >

Public Member Functions

- **TrivialCommand** (std::string name="", std::string description="")
- **TrivialCommand** (const TrivialCommand &copy)=default
- **TrivialCommand** (TrivialCommand &&move)=default
- TrivialCommand & **operator=** (const TrivialCommand &rhs)=default
- virtual void **operator()** ()=0

Additional Inherited Members

5.58.1 Detailed Description

Exactly the same as Command. However, does not support 'usage' nor command arguments. This is used as a wrapper for an invokable to be used in Engine. This is an abstract class. To utilise your own TrivialCommands, create classes which inherit and override virtual void operator()() to provide your desired functionality.

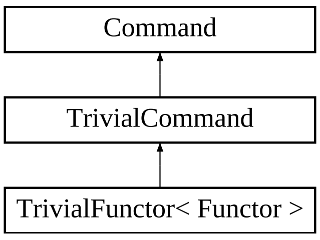
The documentation for this class was generated from the following files:

- src/command.hpp
- src/command.cpp

5.59 TrivialFunctor< Functor > Class Template Reference

```
#include <command.hpp>
```

Inheritance diagram for TrivialFunctor< Functor >:



Public Member Functions

- **TrivialFunctor** (Functor &&functor)
- virtual void **operator()** () override

Protected Attributes

- Functor **functor**

5.59.1 Detailed Description

template<typename Functor>
class TrivialFunctor< Functor >

Wrapper for a trivially-invokable object. Trivially-invokable objects contain a definition of 'operator() const' taking no arguments. If the functor is not trivially invokable, you must use either a StaticFunctor or the Functor class.

Template Parameters

Functor	- Functor type, normally an anonymous class (lambda)

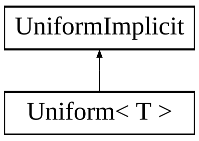
The documentation for this class was generated from the following files:

- src/command.hpp
- src/command.inl

5.60 Uniform< T > Class Template Reference

```
#include <shader.hpp>
```

Inheritance diagram for Uniform< T >:



Public Member Functions

- **Uniform** (GLuint shader_handle, std::string uniform_location, T value)
- **Uniform** (const Uniform< T > &copy)=delete
- **Uniform** (Uniform< T > &&move)=default
- GLuint **get_shader_handle** () const
- virtual std::string_view **get_uniform_location** () const final
- const T & **get_value** () const
- void **set_value** (T value)
- virtual void **push** () const final

5.60.1 Detailed Description

template<class T>
class Uniform< T >

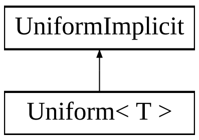
Represent an OpenGL uniform in C++. Supports the following Topaz/C++ primitives: bool, int, unsigned int, float, double, Vector2F, Vector3F, Vector4F, Matrix2x2, Matrix3x3, and Matrix4x4. If the template argument is not any of these types, a static assertation will fail in Uniform<T>::push and emit a compiler error.

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

- src/graphics/shader.hpp
- src/graphics/shader.inl

5.61 UniformImplicit Class Reference

Inheritance diagram for UniformImplicit:



Public Member Functions

- virtual GLuint **get_shader_handle** () const =0
- virtual std::string_view **get_uniform_location** () const =0
- virtual void **push** () const =0

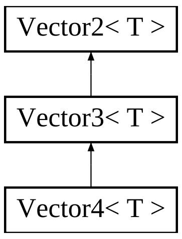
The documentation for this class was generated from the following file:

- src/graphics/shader.hpp

5.62 Vector2< T > Class Template Reference

```
#include <vector.hpp>
```

Inheritance diagram for Vector2< T >:



Public Member Functions

- **Vector2** (T x=T(), T y=T())
- constexpr **Vector2** (const std::array< T, 2 > &data)
- **Vector2** (const Vector2< T > &copy)=default
- **Vector2** (Vector2< T > &&move)=default
- Vector2< T > & **operator=** (const Vector2< T > &rhs)=default
- Vector2POD **to_raw** () const
- T **length** () const
- T **dot** (const Vector2< T > &rhs) const
- Vector2< T > **normalised** () const
- Vector2< T > **operator+** (const Vector2< T > &rhs) const
- Vector2< T > **operator-** (const Vector2< T > &rhs) const
- Vector2< T > **operator*** (T scalar) const
- Vector2< T > **operator/** (T scalar) const
- Vector2< T > & **operator+=** (const Vector2< T > &rhs)
- Vector2< T > & **operator-=** (const Vector2< T > &rhs)
- Vector2< T > & **operator*=** (T scalar)
- Vector2< T > & **operator/=** (T scalar)
- bool **operator<** (const Vector2< T > &rhs) const
- bool **operator>** (const Vector2< T > &rhs) const
- bool **operator<=** (const Vector2< T > &rhs) const
- bool **operator>=** (const Vector2< T > &rhs) const
- bool **operator==** (const Vector2< T > &rhs) const
- Vector2< T > **xy** () const
- Vector2< T > **yx** () const

Public Attributes

- T **x**
- T **y**

5.62.1 Detailed Description

template<typename T>
class Vector2< T >

Tuple of two arguments of type T. Similar to an std::tuple<T, T> but contains useful mathematical functions in addition, such as cross-products.

Template Parameters

T	- The type of which to store a pair of.

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

- src/vector.hpp
- src/vector.inl

5.63 Vector2POD Struct Reference

```
#include <vector.hpp>
```

Public Attributes

- float **x**
- float **y**

5.63.1 Detailed Description

C-style POD structs so that trivial structs can be passed to OpenGL buffers as the memory is guaranteed to be contiguous Plain-Old-Data structure of a Vector2F.

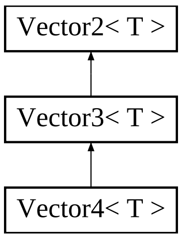
The documentation for this struct was generated from the following file:

- src/vector.hpp

5.64 Vector3< T > Class Template Reference

```
#include <vector.hpp>
```

Inheritance diagram for Vector3< T >:



Public Member Functions

- **Vector3** (T x=T(), T y=T(), T z=T())
- **Vector3** (Vector2< T > xy, T z)
- **Vector3** (T x, Vector2< T > yz)
- constexpr **Vector3** (const std::array< T, 3 > &data)
- **Vector3** (const Vector3< T > &copy)=default
- **Vector3** (Vector3< T > &&move)=default
- Vector3< T > & **operator=** (const Vector3< T > &rhs)=default
- Vector3POD **to_raw** () const
- T **length** () const
- T **dot** (const Vector3< T > &rhs) const
- Vector3< T > **cross** (const Vector3< T > &rhs) const
- Vector3< T > **normalised** () const
- Vector3< T > **operator+** (const Vector3< T > &rhs) const
- Vector3< T > **operator-** (const Vector3< T > &rhs) const

- Vector3< T > **operator*** (T scalar) const
- Vector3< T > **operator/** (T scalar) const
- Vector3< T > & **operator+=** (const Vector3< T > &rhs)
- Vector3< T > & **operator-=** (const Vector3< T > &rhs)
- Vector3< T > & **operator*=** (T scalar)
- Vector3< T > & **operator/=** (T scalar)
- bool **operator<** (const Vector3< T > &rhs) const
- bool **operator>** (const Vector3< T > &rhs) const
- bool **operator<=** (const Vector3< T > &rhs) const
- bool **operator>=** (const Vector3< T > &rhs) const
- bool **operator==** (const Vector3< T > &rhs) const
- Vector3< T > **xyz** () const
- Vector3< T > **xzy** () const
- Vector3< T > **yxz** () const
- Vector3< T > **yzx** () const
- Vector3< T > **zxy** () const
- Vector3< T > **zyx** () const

Public Attributes

- T **z**

5.64.1 Detailed Description

template<typename T>
class Vector3< T >

Tuple of three arguments of type T. Similar to an std::tuple<T, T, T> but contains useful mathematical functions in addition, such as cross-products.

Template Parameters

T	- The type of which to store a trio of.

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

- src/vector.hpp
- src/vector.inl

5.65 Vector3POD Struct Reference

```
#include <vector.hpp>
```

Public Attributes

- float **x**
- float **y**
- float **z**

5.65.1 Detailed Description

Plain-Old-Data structure of a Vector2F.

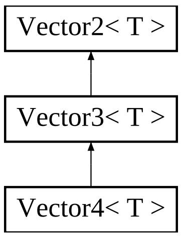
The documentation for this struct was generated from the following file:

- src/vector.hpp

5.66 Vector4< T > Class Template Reference

```
#include <vector.hpp>
```

Inheritance diagram for Vector4< T >:



Public Member Functions

- **Vector4** (T x=T(), T y=T(), T z=T(), T w=T())
- **Vector4** (Vector3< T > xyz, T w)
- **Vector4** (T x, Vector3< T > yzw)
- **Vector4** (Vector2< T > xy, Vector2< T > zw)
- constexpr **Vector4** (const std::array< T, 4 > &data)
- **Vector4** (const Vector4 &copy)=default
- **Vector4** (Vector4 &&move)=default
- Vector4< T > & **operator=** (const Vector4< T > &rhs)=default
- Vector4POD **to_raw** () const
- T **length** () const
- T **dot** (Vector4< T > rhs) const
- Vector4< T > **normalised** () const
- Vector4< T > **operator+** (const Vector4< T > &rhs) const
- Vector4< T > **operator-** (const Vector4< T > &rhs) const
- Vector4< T > **operator*** (T scalar) const
- Vector4< T > **operator/** (T scalar) const
- Vector4< T > & **operator+=** (const Vector4< T > &rhs)
- Vector4< T > & **operator-=** (const Vector4< T > &rhs)
- Vector4< T > & **operator*=** (T scalar)
- Vector4< T > & **operator/=** (T scalar)
- bool **operator<** (const Vector4< T > &rhs) const
- bool **operator>** (const Vector4< T > &rhs) const
- bool **operator<=** (const Vector4< T > &rhs) const
- bool **operator>=** (const Vector4< T > &rhs) const
- bool **operator==** (const Vector4< T > &rhs) const
- Vector4< T > **xyzw** () const

- Vector4< T > **xywz** () const
- Vector4< T > **xzyw** () const
- Vector4< T > **xzwy** () const
- Vector4< T > **xwyz** () const
- Vector4< T > **xwzy** () const
- Vector4< T > **yxzw** () const
- Vector4< T > **yxwz** () const
- Vector4< T > **yzxw** () const
- Vector4< T > **yzwx** () const
- Vector4< T > **ywxz** () const
- Vector4< T > **ywzx** () const
- Vector4< T > **zxyw** () const
- Vector4< T > **zxwy** () const
- Vector4< T > **zyxw** () const
- Vector4< T > **zywx** () const
- Vector4< T > **zwxy** () const
- Vector4< T > **zwyx** () const
- Vector4< T > **wxyz** () const
- Vector4< T > **wxzy** () const
- Vector4< T > **wyxz** () const
- Vector4< T > **wyzx** () const
- Vector4< T > **wzxy** () const
- Vector4< T > **wzyx** () const

Public Attributes

- T **w**

5.66.1 Detailed Description

template<typename T>
class Vector4< T >

Tuple of four arguments of type T. Similar to an std::tuple<T, T, T, T> but contains useful mathematical functions in addition, such as cross-products.

Template Parameters

T	- The type of which to store a quartet of.

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

- src/vector.hpp
- src/vector.inl

5.67 Vector4POD Struct Reference

```
#include <vector.hpp>
```

Public Attributes

- float **x**
- float **y**
- float **z**
- float **w**

5.67.1 Detailed Description

Plain-Old-Data structure of a Vector2F.

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

- src/vector.hpp

5.68 Vertex Class Reference

```
#include <graphics.hpp>
```

Public Member Functions

- **Vertex** (Vector3F position, Vector2F texture_coordinate, Vector3F normal)
- **Vertex** (const Vertex &copy)=default
- **Vertex** (Vertex &&move)=default
- Vertex & **operator=** (const Vertex &rhs)=default

Public Attributes

- Vector3F **position**
- Vector2F **texture_coordinate**
- Vector3F **normal**

5.68.1 Detailed Description

Holds vertex data as POD.

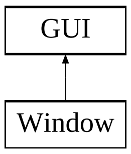
The documentation for this class was generated from the following files:

- src/graphics/graphics.hpp
- src/graphics/graphics.cpp

5.69 Window Class Reference

```
#include <gui.hpp>
```

Inheritance diagram for Window:



Public Types

- enum **SwapIntervalType** : int { **LATE_SWAP_TEARING** = -1, **IMMEDIATE_UPDATES** = 0, **VSYNC** = 1 }
- enum **FullscreenType** : Uint32 { **VIDEO_MODE** = SDL_WINDOW_FULLSCREEN, **DESKTOP_MODE** = SDL_WINDOW_FULLSCREEN_DESKTOP, **WINDOWED_MODE** = 0 }

Public Member Functions

- **Window** (int w=800, int h=600, std::string title="Untitled")
- **Window** (const Window &copy)
- **Window** (Window &&move)=delete
- Window & **operator=** (const Window &rhs)=delete
- virtual void **update** () override
- virtual void **destroy** () override
- virtual bool **focused** () const override
- virtual bool **is_window** () const override
- virtual float **get_window_pos_x** () const override
- virtual float **get_window_pos_y** () const override
- virtual void **set_hidden** (bool hidden) override
- bool **is_close_requested** () const
- SwapIntervalType **get_swap_interval_type** () const
- void **set_swap_interval_type** (SwapIntervalType type) const
- void **set_title** (const std::string &new_title)
- bool **is_fullscreen** () const
- FullscreenType **get_fullscreen** () const
- void **set_fullscreen** (FullscreenType type) const
- void **set_render_target** () const
- void **clear_focus** ()
- void **clear** (GLbitfield mask=(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT|GL_STENCIL_BUF↩ FER_BIT), float r=0.0f, float g=0.0f, float b=0.0f, float a=1.0f) const
- void **register_listener** (Listener &l)
- void **deregister_listener** (Listener &l)
- GUI ∗ **get_focused_child** () const
- void **set_focused_child** (GUI ∗child)

Additional Inherited Members

5.69.1 Detailed Description

Topaz Windows used to draw on the screen. Topaz's graphics module will not initialise fully until at least one instance of this class is instantiated.

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

- src/graphics/gui.hpp
- src/graphics/gui.cpp

Index