# Fracture Engine

0.0.1

Generated by Doxygen 1.9.8

# Chapter 1

# Todo List

**File Component.h**

: The system currently does not support ECS style component system. This will be implemented in the future.

**File EntryPoint.h**

: Add support for other platforms.

**Member Fracture::IndexBuffer::Create (uint32_t *indices, uint32_t size)**

: Currently only supports uint32_t indices. We need to add support for other types of indices.

**Member Fracture::LayerStack::PopLayer (Layer *layer)**

: Should we return the pointer to the layer after detaching it?

**Member Fracture::LayerStack::PopOverlay (Layer *layer)**

: Should we return the pointer to the layer after detaching it?

**Member Fracture::LayerStack::~LayerStack ()**

: Should we return the pointer to the layer instead of deleting it for the application to handle?

**Member Fracture::OpenGLRendererAPI::OpenGLRendererAPI ()**

: Add more states to be set up.

**Member Fracture::OpenGLVertexBuffer::SetData (const void *data, uint32_t size) override**

: Currently the draw call is of type OPENGL_STATIC_DRAW. This needs to be changed to be customizable.

**Member Fracture::Renderer::BeginScene (OrthographicCamera &camera)**

: Currently only supports orthographic camera. Add support for any camera.

**Struct Fracture::Renderer::SceneData**

: Move this to a scene class.

**Member Fracture::Renderer::Submit (const Ref< VertexArray > &vertexArray, const Ref< Shader > &shader, const glm::mat4 &transform)**

: Add support for materails.

: Add support for batch rendering.

: Add support for instanced rendering.

**Member Fracture::RendererAPI::GetAPI ()**

: Provide a way for the api to be set by the user and not be hardcoded.

**Class Fracture::Shader**

: Add a way to automatically find the uniforms that need to be set in the shader.

**Class Fracture::ShaderLibrary**

: Add a way to make instances of existing shaders.

: Add some default shaders.

**Member Fracture::Window::Create (const WindowProperties &properties=WindowProperties())**

: Currently this only creates a WindowsWindow. In the future we will have to check the platform and create the appropriate window.

**File GraphicsContext.h**

: Add VulkanContext

**File Log.h**

: Add logging to file

**File Renderer.h**

: Add VulkanRendererAPI

# Chapter 2

# Topic Index

## 2.1 Topics

Here is a list of all topics with brief descriptions:

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 4

# Hierarchical Index

## 4.1 Class Hierarchy

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

# Chapter 5

# Class Index

## 5.1 Class List

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

# Chapter 6

# File Index

## 6.1 File List

Here is a list of all files with brief descriptions:

# Chapter 7

# Topic Documentation

## 7.1 Keyboard buttons

Keyboard key IDs.

### Macros

- #define FR_KEY_SPACE 32
- #define FR_KEY_APOSTROPHE 39 /* ' */
- #define FR_KEY_COMMA 44 /* , */
- #define FR_KEY_MINUS 45 /* - */
- #define FR_KEY_PERIOD 46 /* . */
- #define FR_KEY_SLASH 47 /* / */
- #define FR_KEY_0 48
- #define FR_KEY_1 49
- #define FR_KEY_2 50
- #define FR_KEY_3 51
- #define FR_KEY_4 52
- #define FR_KEY_5 53
- #define FR_KEY_6 54
- #define FR_KEY_7 55
- #define FR_KEY_8 56
- #define FR_KEY_9 57
- #define FR_KEY_SEMICOLON 59 /* ; */
- #define FR_KEY_EQUAL 61 /* = */
- #define FR_KEY_A 65
- #define FR_KEY_B 66
- #define FR_KEY_C 67
- #define FR_KEY_D 68
- #define FR_KEY_E 69
- #define FR_KEY_F 70
- #define FR_KEY_G 71
- #define FR_KEY_H 72
- #define FR_KEY_I 73
- #define FR_KEY_J 74
- #define FR_KEY_K 75
- #define FR_KEY_L 76

- #define FR_KEY_M 77
- #define FR_KEY_N 78
- #define FR_KEY_O 79
- #define FR_KEY_P 80
- #define FR_KEY_Q 81
- #define FR_KEY_R 82
- #define FR_KEY_S 83
- #define FR_KEY_T 84
- #define FR_KEY_U 85
- #define FR_KEY_V 86
- #define FR_KEY_W 87
- #define FR_KEY_X 88
- #define FR_KEY_Y 89
- #define FR_KEY_Z 90
- #define FR_KEY_LEFT_BRACKET 91 /* [ */
- #define FR_KEY_BACKSLASH 92 /* \ */
- #define FR_KEY_RIGHT_BRACKET 93 /* ] */
- #define FR_KEY_GRAVE_ACCENT 96 /* ` */
- #define FR_KEY_WORLD_1 161 /* non-US #1 */
- #define FR_KEY_WORLD_2 162 /* non-US #2 */
- #define FR_KEY_ESCAPE 256
- #define FR_KEY_ENTER 257
- #define FR_KEY_TAB 258
- #define FR_KEY_BACKSPACE 259
- #define FR_KEY_INSERT 260
- #define FR_KEY_DELETE 261
- #define FR_KEY_RIGHT 262
- #define FR_KEY_LEFT 263
- #define FR_KEY_DOWN 264
- #define FR_KEY_UP 265
- #define FR_KEY_PAGE_UP 266
- #define FR_KEY_PAGE_DOWN 267
- #define FR_KEY_HOME 268
- #define FR_KEY_END 269
- #define FR_KEY_CAPS_LOCK 280
- #define FR_KEY_SCROLL_LOCK 281
- #define FR_KEY_NUM_LOCK 282
- #define FR_KEY_PRINT_SCREEN 283
- #define FR_KEY_PAUSE 284
- #define FR_KEY_F1 290
- #define FR_KEY_F2 291
- #define FR_KEY_F3 292
- #define FR_KEY_F4 293
- #define FR_KEY_F5 294
- #define FR_KEY_F6 295
- #define FR_KEY_F7 296
- #define FR_KEY_F8 297
- #define FR_KEY_F9 298
- #define FR_KEY_F10 299
- #define FR_KEY_F11 300
- #define FR_KEY_F12 301
- #define FR_KEY_F13 302
- #define FR_KEY_F14 303
- #define FR_KEY_F15 304
- #define FR_KEY_F16 305

- #define FR_KEY_F17 306
- #define FR_KEY_F18 307
- #define FR_KEY_F19 308
- #define FR_KEY_F20 309
- #define FR_KEY_F21 310
- #define FR_KEY_F22 311
- #define FR_KEY_F23 312
- #define FR_KEY_F24 313
- #define FR_KEY_F25 314
- #define FR_KEY_KP_0 320
- #define FR_KEY_KP_1 321
- #define FR_KEY_KP_2 322
- #define FR_KEY_KP_3 323
- #define FR_KEY_KP_4 324
- #define FR_KEY_KP_5 325
- #define FR_KEY_KP_6 326
- #define FR_KEY_KP_7 327
- #define FR_KEY_KP_8 328
- #define FR_KEY_KP_9 329
- #define FR_KEY_KP_DECIMAL 330
- #define FR_KEY_KP_DIVIDE 331
- #define FR_KEY_KP_MULTIPLY 332
- #define FR_KEY_KP_SUBTRACT 333
- #define FR_KEY_KP_ADD 334
- #define FR_KEY_KP_ENTER 335
- #define FR_KEY_KP_EQUAL 336
- #define FR_KEY_LEFT_SHIFT 340
- #define FR_KEY_LEFT_CONTROL 341
- #define FR_KEY_LEFT_ALT 342
- #define FR_KEY_RIGHT_SHIFT 344
- #define FR_KEY_RIGHT_CONTROL 345
- #define FR_KEY_RIGHT_ALT 346
- #define FR_KEY_MENU 348
- #define FR_KEY_LEFT_SUPER 343
- #define FR_KEY_LEFT_WINDOWS 343
- #define FR_KEY_RIGHT_SUPER 347
- #define FR_KEY_RIGHT_WINDOWS 347
- #define FR_MOUSE_BUTTON_1 0
- #define FR_MOUSE_BUTTON_2 1
- #define FR_MOUSE_BUTTON_3 2
- #define FR_MOUSE_BUTTON_4 3
- #define FR_MOUSE_BUTTON_5 4
- #define FR_MOUSE_BUTTON_6 5
- #define FR_MOUSE_BUTTON_7 6
- #define FR_MOUSE_BUTTON_8 7
- #define FR_MOUSE_BUTTON_LAST FR_MOUSE_BUTTON_8
- #define FR_MOUSE_BUTTON_LEFT FR_MOUSE_BUTTON_1
- #define FR_MOUSE_BUTTON_RIGHT FR_MOUSE_BUTTON_2
- #define FR_MOUSE_BUTTON_MIDDLE FR_MOUSE_BUTTON_3

### 7.1.1 Detailed Description

Keyboard key IDs.

Mouse button IDs.

### 7.1.2 Macro Definition Documentation

#### 7.1.2.1 FR_KEY_0

```
#define FR_KEY_0 48
```

#### 7.1.2.2 FR_KEY_1

```
#define FR_KEY_1 49
```

#### 7.1.2.3 FR_KEY_2

```
#define FR_KEY_2 50
```

#### 7.1.2.4 FR_KEY_3

```
#define FR_KEY_3 51
```

#### 7.1.2.5 FR_KEY_4

```
#define FR_KEY_4 52
```

#### 7.1.2.6 FR_KEY_5

```
#define FR_KEY_5 53
```

#### 7.1.2.7 FR_KEY_6

```
#define FR_KEY_6 54
```

#### 7.1.2.8 FR_KEY_7

```
#define FR_KEY_7 55
```

#### 7.1.2.9 FR_KEY_8

```
#define FR_KEY_8 56
```

#### 7.1.2.10 FR_KEY_9

```
#define FR_KEY_9 57
```

#### 7.1.2.11 FR_KEY_A

```
#define FR_KEY_A 65
```

#### 7.1.2.12 FR_KEY_APOSTROPHE

```
#define FR_KEY_APOSTROPHE 39 /* ' */
```

#### 7.1.2.13 FR_KEY_B

```
#define FR_KEY_B 66
```

#### 7.1.2.14 FR_KEY_BACKSLASH

```
#define FR_KEY_BACKSLASH 92 /* \ */
```

#### 7.1.2.15 FR_KEY_BACKSPACE

```
#define FR_KEY_BACKSPACE 259
```

#### 7.1.2.16 FR_KEY_C

```
#define FR_KEY_C 67
```

#### 7.1.2.17 FR_KEY_CAPS_LOCK

```
#define FR_KEY_CAPS_LOCK 280
```

#### 7.1.2.18 FR_KEY_COMMA

```
#define FR_KEY_COMMA 44 /* , */
```

#### 7.1.2.19 FR_KEY_D

```
#define FR_KEY_D 68
```

#### 7.1.2.20 FR_KEY_DELETE

```
#define FR_KEY_DELETE 261
```

#### 7.1.2.21 FR_KEY_DOWN

```
#define FR_KEY_DOWN 264
```

#### 7.1.2.22 FR_KEY_E

```
#define FR_KEY_E 69
```

#### 7.1.2.23 FR_KEY_END

```
#define FR_KEY_END 269
```

#### 7.1.2.24 FR_KEY_ENTER

```
#define FR_KEY_ENTER 257
```

#### 7.1.2.25 FR_KEY_EQUAL

```
#define FR_KEY_EQUAL 61 /* = */
```

#### 7.1.2.26 FR_KEY_ESCAPE

```
#define FR_KEY_ESCAPE 256
```

#### 7.1.2.27 FR_KEY_F

```
#define FR_KEY_F 70
```

#### 7.1.2.28 FR_KEY_F1

```
#define FR_KEY_F1 290
```

#### 7.1.2.29 FR_KEY_F10

```
#define FR_KEY_F10 299
```

#### 7.1.2.30 FR_KEY_F11

```
#define FR_KEY_F11 300
```

#### 7.1.2.31 FR_KEY_F12

```
#define FR_KEY_F12 301
```

#### 7.1.2.32 FR_KEY_F13

```
#define FR_KEY_F13 302
```

#### 7.1.2.33 FR_KEY_F14

```
#define FR_KEY_F14 303
```

#### 7.1.2.34 FR_KEY_F15

```
#define FR_KEY_F15 304
```

#### 7.1.2.35 FR_KEY_F16

```
#define FR_KEY_F16 305
```

#### 7.1.2.36 FR_KEY_F17

```
#define FR_KEY_F17 306
```

#### 7.1.2.37 FR_KEY_F18

```
#define FR_KEY_F18 307
```

#### 7.1.2.38 FR_KEY_F19

```
#define FR_KEY_F19 308
```

#### 7.1.2.39 FR_KEY_F2

```
#define FR_KEY_F2 291
```

#### 7.1.2.40 FR_KEY_F20

```
#define FR_KEY_F20 309
```

#### 7.1.2.41 FR_KEY_F21

```
#define FR_KEY_F21 310
```

#### 7.1.2.42 FR_KEY_F22

```
#define FR_KEY_F22 311
```

#### 7.1.2.43 FR_KEY_F23

```
#define FR_KEY_F23 312
```

#### 7.1.2.44 FR_KEY_F24

```
#define FR_KEY_F24 313
```

#### 7.1.2.45 FR_KEY_F25

```
#define FR_KEY_F25 314
```

#### 7.1.2.46 FR_KEY_F3

```
#define FR_KEY_F3 292
```

#### 7.1.2.47 FR_KEY_F4

```
#define FR_KEY_F4 293
```

#### 7.1.2.48 FR_KEY_F5

```
#define FR_KEY_F5 294
```

#### 7.1.2.49 FR_KEY_F6

```
#define FR_KEY_F6 295
```

#### 7.1.2.50 FR_KEY_F7

```
#define FR_KEY_F7 296
```

#### 7.1.2.51 FR_KEY_F8

```
#define FR_KEY_F8 297
```

#### 7.1.2.52 FR_KEY_F9

```
#define FR_KEY_F9 298
```

#### 7.1.2.53 FR_KEY_G

```
#define FR_KEY_G 71
```

#### 7.1.2.54 FR_KEY_GRAVE_ACCENT

```
#define FR_KEY_GRAVE_ACCENT 96 /* ` */
```

#### 7.1.2.55 FR_KEY_H

```
#define FR_KEY_H 72
```

#### 7.1.2.56 FR_KEY_HOME

```
#define FR_KEY_HOME 268
```

#### 7.1.2.57 FR_KEY_I

```
#define FR_KEY_I 73
```

#### 7.1.2.58 FR_KEY_INSERT

```
#define FR_KEY_INSERT 260
```

#### 7.1.2.59 FR_KEY_J

```
#define FR_KEY_J 74
```

#### 7.1.2.60 FR_KEY_K

```
#define FR_KEY_K 75
```

#### 7.1.2.61 FR_KEY_KP_0

```
#define FR_KEY_KP_0 320
```

#### 7.1.2.62 FR_KEY_KP_1

```
#define FR_KEY_KP_1 321
```

#### 7.1.2.63 FR_KEY_KP_2

```
#define FR_KEY_KP_2 322
```

#### 7.1.2.64 FR_KEY_KP_3

```
#define FR_KEY_KP_3 323
```

#### 7.1.2.65 FR_KEY_KP_4

```
#define FR_KEY_KP_4 324
```

#### 7.1.2.66 FR_KEY_KP_5

```
#define FR_KEY_KP_5 325
```

#### 7.1.2.67 FR_KEY_KP_6

```
#define FR_KEY_KP_6 326
```

#### 7.1.2.68 FR_KEY_KP_7

```
#define FR_KEY_KP_7 327
```

#### 7.1.2.69 FR_KEY_KP_8

```
#define FR_KEY_KP_8 328
```

#### 7.1.2.70 FR_KEY_KP_9

```
#define FR_KEY_KP_9 329
```

#### 7.1.2.71 FR_KEY_KP_ADD

```
#define FR_KEY_KP_ADD 334
```

#### 7.1.2.72 FR_KEY_KP_DECIMAL

```
#define FR_KEY_KP_DECIMAL 330
```

#### 7.1.2.73 FR_KEY_KP_DIVIDE

```
#define FR_KEY_KP_DIVIDE 331
```

#### 7.1.2.74 FR_KEY_KP_ENTER

```
#define FR_KEY_KP_ENTER 335
```

#### 7.1.2.75 FR_KEY_KP_EQUAL

```
#define FR_KEY_KP_EQUAL 336
```

#### 7.1.2.76 FR_KEY_KP_MULTIPLY

```
#define FR_KEY_KP_MULTIPLY 332
```

#### 7.1.2.77 FR_KEY_KP_SUBTRACT

```
#define FR_KEY_KP_SUBTRACT 333
```

#### 7.1.2.78 FR_KEY_L

```
#define FR_KEY_L 76
```

#### 7.1.2.79 FR_KEY_LEFT

```
#define FR_KEY_LEFT 263
```

#### 7.1.2.80 FR_KEY_LEFT_ALT

```
#define FR_KEY_LEFT_ALT 342
```

#### 7.1.2.81 FR_KEY_LEFT_BRACKET

```
#define FR_KEY_LEFT_BRACKET 91 /* [ */
```

#### 7.1.2.82 FR_KEY_LEFT_CONTROL

```
#define FR_KEY_LEFT_CONTROL 341
```

#### 7.1.2.83 FR_KEY_LEFT_SHIFT

```
#define FR_KEY_LEFT_SHIFT 340
```

#### 7.1.2.84 FR_KEY_LEFT_SUPER

```
#define FR_KEY_LEFT_SUPER 343
```

#### 7.1.2.85 FR_KEY_LEFT_WINDOWS

```
#define FR_KEY_LEFT_WINDOWS 343
```

#### 7.1.2.86 FR_KEY_M

```
#define FR_KEY_M 77
```

#### 7.1.2.87 FR_KEY_MENU

```
#define FR_KEY_MENU 348
```

#### 7.1.2.88 FR_KEY_MINUS

```
#define FR_KEY_MINUS 45 /* - */
```

#### 7.1.2.89 FR_KEY_N

```
#define FR_KEY_N 78
```

#### 7.1.2.90 FR_KEY_NUM_LOCK

```
#define FR_KEY_NUM_LOCK 282
```

#### 7.1.2.91 FR_KEY_O

```
#define FR_KEY_O 79
```

#### 7.1.2.92 FR_KEY_P

```
#define FR_KEY_P 80
```

#### 7.1.2.93 FR_KEY_PAGE_DOWN

```
#define FR_KEY_PAGE_DOWN 267
```

#### 7.1.2.94 FR_KEY_PAGE_UP

```
#define FR_KEY_PAGE_UP 266
```

#### 7.1.2.95 FR_KEY_PAUSE

```
#define FR_KEY_PAUSE 284
```

#### 7.1.2.96 FR_KEY_PERIOD

```
#define FR_KEY_PERIOD 46 /* .  */
```

#### 7.1.2.97 FR_KEY_PRINT_SCREEN

```
#define FR_KEY_PRINT_SCREEN 283
```

#### 7.1.2.98 FR_KEY_Q

```
#define FR_KEY_Q 81
```

#### 7.1.2.99 FR_KEY_R

```
#define FR_KEY_R 82
```

#### 7.1.2.100 FR_KEY_RIGHT

```
#define FR_KEY_RIGHT 262
```

#### 7.1.2.101 FR_KEY_RIGHT_ALT

```
#define FR_KEY_RIGHT_ALT 346
```

#### 7.1.2.102 FR_KEY_RIGHT_BRACKET

```
#define FR_KEY_RIGHT_BRACKET 93 /* ] */
```

#### 7.1.2.103 FR_KEY_RIGHT_CONTROL

```
#define FR_KEY_RIGHT_CONTROL 345
```

#### 7.1.2.104 FR_KEY_RIGHT_SHIFT

```
#define FR_KEY_RIGHT_SHIFT 344
```

#### 7.1.2.105 FR_KEY_RIGHT_SUPER

```
#define FR_KEY_RIGHT_SUPER 347
```

#### 7.1.2.106 FR_KEY_RIGHT_WINDOWS

```
#define FR_KEY_RIGHT_WINDOWS 347
```

#### 7.1.2.107 FR_KEY_S

```
#define FR_KEY_S 83
```

#### 7.1.2.108 FR_KEY_SCROLL_LOCK

```
#define FR_KEY_SCROLL_LOCK 281
```

#### 7.1.2.109 FR_KEY_SEMICOLON

```
#define FR_KEY_SEMICOLON 59 /* ; */
```

#### 7.1.2.110 FR_KEY_SLASH

```
#define FR_KEY_SLASH 47 /* / */
```

#### 7.1.2.111 FR_KEY_SPACE

```
#define FR_KEY_SPACE 32
```

#### 7.1.2.112 FR_KEY_T

```
#define FR_KEY_T 84
```

#### 7.1.2.113 FR_KEY_TAB

```
#define FR_KEY_TAB 258
```

#### 7.1.2.114 FR_KEY_U

```
#define FR_KEY_U 85
```

#### 7.1.2.115 FR_KEY_UP

```
#define FR_KEY_UP 265
```

#### 7.1.2.116 FR_KEY_V

```
#define FR_KEY_V 86
```

#### 7.1.2.117 FR_KEY_W

```
#define FR_KEY_W 87
```

#### 7.1.2.118 FR_KEY_WORLD_1

```
#define FR_KEY_WORLD_1 161 /* non-US #1 */
```

#### 7.1.2.119 FR_KEY_WORLD_2

```
#define FR_KEY_WORLD_2 162 /* non-US #2 */
```

#### 7.1.2.120 FR_KEY_X

```
#define FR_KEY_X 88
```

#### 7.1.2.121 FR_KEY_Y

```
#define FR_KEY_Y 89
```

#### 7.1.2.122 FR_KEY_Z

```
#define FR_KEY_Z 90
```

#### 7.1.2.123 FR_MOUSE_BUTTON_1

```
#define FR_MOUSE_BUTTON_1 0
```

#### 7.1.2.124 FR_MOUSE_BUTTON_2

```
#define FR_MOUSE_BUTTON_2 1
```

#### 7.1.2.125 FR_MOUSE_BUTTON_3

```
#define FR_MOUSE_BUTTON_3 2
```

#### 7.1.2.126 FR_MOUSE_BUTTON_4

```
#define FR_MOUSE_BUTTON_4 3
```

#### 7.1.2.127 FR_MOUSE_BUTTON_5

```
#define FR_MOUSE_BUTTON_5 4
```

#### 7.1.2.128 FR_MOUSE_BUTTON_6

```
#define FR_MOUSE_BUTTON_6 5
```

#### 7.1.2.129 FR_MOUSE_BUTTON_7

```
#define FR_MOUSE_BUTTON_7 6
```

#### 7.1.2.130 FR_MOUSE_BUTTON_8

```
#define FR_MOUSE_BUTTON_8 7
```

#### 7.1.2.131 FR_MOUSE_BUTTON_LAST

```
#define FR_MOUSE_BUTTON_LAST FR_MOUSE_BUTTON_8
```

#### 7.1.2.132 FR_MOUSE_BUTTON_LEFT

```
#define FR_MOUSE_BUTTON_LEFT FR_MOUSE_BUTTON_1
```

#### 7.1.2.133 FR_MOUSE_BUTTON_MIDDLE

```
#define FR_MOUSE_BUTTON_MIDDLE FR_MOUSE_BUTTON_3
```

#### 7.1.2.134 FR_MOUSE_BUTTON_RIGHT

```
#define FR_MOUSE_BUTTON_RIGHT FR_MOUSE_BUTTON_2
```

# Chapter 8

# Namespace Documentation

## 8.1 Fracture Namespace Reference

### Namespaces

- namespace Utils

### Classes

- class Application

  *The Application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers.*
- class AppRenderEvent
- class AppTickEvent
- class AppUpdateEvent
- struct BufferElement

  *The BufferElement struct is used to store the elements of the vertex buffer layout.*
- class BufferLayout

  *The BufferLayout class is used to store the layout of the vertex buffer. Each vertex buffer has a buffer layout.*
- class Event

  *Base class for all events.*
- class EventDispatcher
- class GraphicsContext

  *The GraphicsContext class is an abstract class that is used to create a graphics context for the application. Each renderer will have its own implementation of the graphics context.*
- class ImGuiLayer
- class IndexBuffer

  *The IndexBuffer class is an abstract class that is used to store the index buffer. Each renderer will have its own implementation of the index buffer.*
- class Input

  *The base class for Input polling. This class will be implemented per platform.*
- class KeyEvent

  *the base class for KeyEvents*
- class KeyPressedEvent

  *Event class for when a key is pressed.*
- class KeyReleasedEvent

*Event class for when a key is released.*

- class KeyTypedEvent

  *Event class for when a key is typed.*

- class Layer

  *The Layer class is the base class for all layers in the engine. Layers are used to separate different parts of the application and set an order of execution.*

- class LayerStack

  *The LayerStack class is used to store all the layers that are currently active.*

- class Log

  *The Log class is used to log messages to the console.*

- class MouseButtonEvent

  *Base class for mouse button events.*

- class MouseButtonPressedEvent

  *Event class for when a mouse button is pressed.*

- class MouseButtonReleasedEvent

  *Event class for when a mouse button is released.*

- class MouseMovedEvent

  *Event for when the mouse is moved.*

- class MouseScrolledEvent

  *Event for when the mouse is scrolled.*

- class OpenGLContext

  *The OpenGLContext class is an implementation of the GraphicsContext class for the OpenGL renderer.*

- class OpenGLIndexBuffer
- class OpenGLRendererAPI

  *Implementation of the RendererAPI for OpenGL.*

- class OpenGLShader

  *The OpenGLShader class is an implementation of the Shader class. It is used to create a shader program for the OpenGL renderer.*

- class OpenGLTexture2D

  *OpenGL implementation of the Texture2D class.*

- class OpenGLVertexArray

  *OpenGLVertexArray class that implements the VertexArray class for OpenGL.*

- class OpenGLVertexBuffer

  *The OpenGLVertexBuffer class is an implementation of the VertexBuffer class for OpenGL.*

- class OrthographicCamera
- class OrthographicCameraController

  *The OrthographicCameraController class is used to control the orthographic camera.*

- class RenderCommand

  *The RenderCommand class is used to send commands to the renderer. It is a thin wrapper around the RendererAPI class.*

- class Renderer

  *The Renderer class is used to render a scene. It provides an interface to render a scene.*

- class RendererAPI

  *The RendererAPI class provides an interface for the RendererAPI that needs to be implemented by each renderer.*

- class Shader

  *The Shader class is an abstract class that is used to create a shader for the application. Each renderer will have its own implementation of the shader.*

- class ShaderLibrary

  *The ShaderLibrary class is a singleton class that is used to store all the shaders that are created in the application.*

- class Texture

  *The Texture class is an abstract class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.*

- class Texture2D

  *The Texture2D class is an abstract class that is used to store references to 2D textures needed for rendering. Each renderer will have its own implementation of the texture class.*
- class TransformComponent
- class VertexArray

  *The VertexArray class is an abstract class that is used to create a VertexArray object. Each renderer will have its own implementation of the VertexArray class.*
- class VertexBuffer

  *The VertexBuffer class is an abstract class that is used to store the vertex buffer. Each renderer will have its own implementation of the vertex buffer.*
- class Window

  *Window interface representing a desktop system based Window. This is an abstract class.*
- class WindowCloseEvent

  *Event class for holding information about window close events.*
- struct WindowProperties

  *Stores the necessary information for a window.*
- class WindowResizeEvent

  *Event class for holding information about window resize events.*
- class WindowsInput

  *The Windows implementation of the Input class.*
- class WindowsWindow

  *The WindowsWindow class is used to create a window for the application. It owns the renderer context.*

### Typedefs

- template<typename T >
  using Scope = std::unique_ptr< T >
- template<typename T >
  using Ref = std::shared_ptr< T >

### Enumerations

- enum class EventType {
  None = 0 , WindowClose , WindowResize , WindowFocus ,
  WindowLostFocus , WindowMoved , AppTick , AppUpdate ,
  AppRender , KeyPressed , KeyReleased , KeyTyped ,
  MouseButtonPressed , MouseButtonReleased , MouseMoved , MouseScrolled }

  *Enum class for the different types of events.*
- enum EventCategory {
  None = 0 , EventCategoryApplication = BIT(0) , EventCategoryInput = BIT(1) , EventCategoryKeyboard = BIT(2) ,
  EventCategoryMouse = BIT(3) , EventCategoryMouseButton = BIT(4) }

  *Enum class for the different categories of events. These are bit masks that can be combined.*
- enum class ShaderDataType {
  None = 0 , Float , Float2 , Float3 ,
  Float4 , Mat3 , Mat4 , Int ,
  Int2 , Int3 , Int4 , Bool }

  *The ShaderDataType enum is used to store the data type of the vertex buffer layout.*

### Functions

- Application * CreateApplication ()
- template<typename T , typename ... Args>
  constexpr Scope< T > CreateScope (Args &&... args)
- template<typename T , typename ... Args>
  constexpr Ref< T > CreateRef (Args &&... args)
- std::ostream & operator<< (std::ostream &stream, const Event &event)

  *Overload of the << operator for events. It calls the ToString() function of the event and then pushes it to the stream.*
- static uint32_t ShaderDataTypeSize (ShaderDataType type)

  *The ShaderDataTypeSize function is used to return the size of the data type of the vertex buffer layout.*
- static GLenum ShaderTypeFromString (const std::string &type)
- static std::string ShaderTypeToString (GLenum type)
- static void CompileShaders (GLuint handle, const std::string &source, GLenum type)
- static GLenum ShaderDataTypeToOpenGLBaseType (ShaderDataType type)
- static void GLFWErrorCallback (int error, const char *description)

### Variables

- static uint8_t s_GLFWWindowCount = 0

## 8.1.1 Typedef Documentation

### 8.1.1.1 Ref

```
template<typename T >
using Fracture::Ref = typedef std::shared_ptr<T>
```

### 8.1.1.2 Scope

```
template<typename T >
using Fracture::Scope = typedef std::unique_ptr<T>
```

## 8.1.2 Enumeration Type Documentation

### 8.1.2.1 EventCategory

```
enum Fracture::EventCategory
```

Enum class for the different categories of events. These are bit masks that can be combined.

**Enumerator**

| | |
|---|---|
| None | |
| EventCategoryApplication | |
| EventCategoryInput | |
| EventCategoryKeyboard | |
| EventCategoryMouse | |
| EventCategoryMouseButton | |

#### 8.1.2.2 EventType

`enum class Fracture::EventType [strong]`

Enum class for the different types of events.

**Enumerator**

| | |
|---|---|
| None | |
| WindowClose | |
| WindowResize | |
| WindowFocus | |
| WindowLostFocus | |
| WindowMoved | |
| AppTick | |
| AppUpdate | |
| AppRender | |
| KeyPressed | |
| KeyReleased | |
| KeyTyped | |
| MouseButtonPressed | |
| MouseButtonReleased | |
| MouseMoved | |
| MouseScrolled | |

#### 8.1.2.3 ShaderDataType

`enum class Fracture::ShaderDataType [strong]`

The ShaderDataType enum is used to store the data type of the vertex buffer layout.

**Enumerator**

| | |
|---|---|
| None | |
| Float | |
| Float2 | |
| Float3 | |
| Float4 | |
| Mat3 | |
| Mat4 | |
| Int | |
| Int2 | |
| Int3 | |
| Int4 | |
| Bool | |

### 8.1.3 Function Documentation

#### 8.1.3.1 CompileShaders()

```
static void Fracture::CompileShaders (
            GLuint handle,
            const std::string & source,
            GLenum type )  [static]
```

#### 8.1.3.2 CreateApplication()

```
Application * Fracture::CreateApplication ( )
```

#### 8.1.3.3 CreateRef()

```
template<typename T , typename ...  Args>
constexpr Ref< T > Fracture::CreateRef (
            Args &&...  args )  [constexpr]
```

#### 8.1.3.4 CreateScope()

```
template<typename T , typename ...  Args>
constexpr Scope< T > Fracture::CreateScope (
            Args &&...  args )  [constexpr]
```

#### 8.1.3.5 GLFWErrorCallback()

```
static void Fracture::GLFWErrorCallback (
            int error,
            const char * description )  [static]
```

#### 8.1.3.6 operator<<()

```
std::ostream & Fracture::operator<< (
            std::ostream & stream,
            const Event & event )  [inline]
```

Overload of the << operator for events. It calls the ToString() function of the event and then pushes it to the stream.

**Parameters**

| | |
|---|---|
| *std::ostream&* | stream: The stream to push the event to |
| *const* | Event& event: The event to push to the stream |

**Returns**

std::ostream&: The stream with the event pushed to it

#### 8.1.3.7 ShaderDataTypeSize()

```
static uint32_t Fracture::ShaderDataTypeSize (
            ShaderDataType type )  [static]
```

The ShaderDataTypeSize function is used to return the size of the data type of the vertex buffer layout.

#### 8.1.3.8 ShaderDataTypeToOpenGLBaseType()

```
static GLenum Fracture::ShaderDataTypeToOpenGLBaseType (
            ShaderDataType type )  [static]
```

#### 8.1.3.9 ShaderTypeFromString()

```
static GLenum Fracture::ShaderTypeFromString (
            const std::string & type )  [static]
```

#### 8.1.3.10 ShaderTypeToString()

```
static std::string Fracture::ShaderTypeToString (
            GLenum type )  [static]
```

### 8.1.4 Variable Documentation

#### 8.1.4.1 s_GLFWWindowCount

```
uint8_t Fracture::s_GLFWWindowCount = 0  [static]
```

## 8.2 Fracture::Utils Namespace Reference

**Classes**

- struct InstrumentationSession
- class InstrumentationTimer
- class Instrumentor
- struct ProfileResult
- struct Timestep

  *data structure used to store time in seconds*

### Functions

- std::string ReadFile (const std::string &filePath)

  *Reads a file and returns the contents as a string.*

## 8.2.1 Function Documentation

#### 8.2.1.1 ReadFile()

```
std::string Fracture::Utils::ReadFile (
            const std::string & filePath )
```

Reads a file and returns the contents as a string.

**Parameters**

| | |
|---|---|
| *const* | std::string& filePath: The path to the file to read |

**Returns**

std::string: The contents of the file as a string

# Chapter 9

# Class Documentation

## 9.1 Fracture::Application Class Reference

The Application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers.

`#include <Application.h>`

### Public Member Functions

- Application ()

  *This is a constructor for the application class.*
- virtual ~Application ()=default
- void Run ()

  *This is a function that is the main loop of the application.*
- void OnEvent (Event &e)

  *This is a function that will be called when an event is triggered.*
- void PushLayer (Layer *layer)

  *This is a function that will push a layer onto the application layer stack.*
- void PushOverlay (Layer *layer)

  *This is a function that will push an overlay onto the application layer stack.*
- Window & GetWindow ()

  *This is a function that will return a reference to the window.*

### Static Public Member Functions

- static Application & Get ()

  *This is a static function that will return a reference to the application class.*

### Private Member Functions

- bool OnWindowClose (WindowCloseEvent &e)

  *This is a boolean function that will be called when the window is closed.*
- bool OnWindowResize (WindowResizeEvent &e)

  *This is a boolean function that will be called when the window is resized.*

**Private Attributes**

- Ref< Window > m_Window

  *A unique pointer to a window object tha is managed by the application class.*
- LayerStack m_LayerStack

  *The application layer stack. This will store all the layers that are currently active and will be updated every frame.*
- ImGuiLayer * m_ImGuiLayer
- bool m_Running = true

  *A pointer to the ImGuiLayer object that is managed by the application class. This is used to render the ImGui UI.*
- bool m_isMinimized = false

  *this is a boolean that will be used to determine if the application is running or not.*
- long long m_LastFrameTime = 0

  *this is a boolean that will be used to determine if the application is minimized or not.*

**Static Private Attributes**

- static Application * s_Instance = nullptr

  *Stores the start time of the last frame. Used to calculate the delta time.*

## 9.1.1 Detailed Description

The Application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers.

The application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers. It also contains the layer stack which is used to store all the layers that are currently active. It handles all the events that are triggered by the window and dispatches them to the appropriate callback functions. There can only be one instance of the application class and it is created in the main function. The application class is a singleton class.

**See also**

Window

LayerStack

## 9.1.2 Constructor & Destructor Documentation

### 9.1.2.1 Application()

```
Fracture::Application::Application ( )
```

This is a constructor for the application class.

this is a static pointer to the application class.

The constructor will create a window object, set the event callback function to the OnEvent function in the application class, and initialize the Renderer.

**See also**

Window

Renderer

OnEvent

#### 9.1.2.2 ~Application()

```
virtual Fracture::Application::~Application ( )  [virtual], [default]
```

### 9.1.3 Member Function Documentation

#### 9.1.3.1 Get()

```
static Application & Fracture::Application::Get ( )  [inline], [static]
```

This is a static function that will return a reference to the application class.

**See also**

Application

**Returns**

Application& - returns a reference to the static instance of the application.

#### 9.1.3.2 GetWindow()

```
Window & Fracture::Application::GetWindow ( )  [inline]
```

This is a function that will return a reference to the window.

**See also**

Window

**Returns**

Window& - returns a reference to the current window.

#### 9.1.3.3 OnEvent()

```
void Fracture::Application::OnEvent (
            Event & e )
```

This is a function that will be called when an event is triggered.

The function will be called when an event is triggered. The function will then dispatch the events to the appropriate callback functions. Currently it is only triggered by the window events(windows close/resize, key events, and mouse events, etc.).

**See also**

Event
WindowResizeEvent
WindowCloseEvent
MouseScrolledEvent
MouseMovedEvent
MouseButtonPressedEvent
MouseButtonReleasedEvent
KeyPressedEvent
KeyReleasedEvent
KeyTypedEvent

#### 9.1.3.4 OnWindowClose()

```
bool Fracture::Application::OnWindowClose (
            WindowCloseEvent & e )  [private]
```

This is a boolean function that will be called when the window is closed.

The function will set the running boolean to false and return true. This will stop the application from running.

**Parameters**

| | | |
|---|---|---|
| in | *WindowCloseEvent&* | e - a reference to the window close event. |

**Returns**

bool - returns true if the window close event is handled here false if it needs to continue to be propogated.

#### 9.1.3.5 OnWindowResize()

```
bool Fracture::Application::OnWindowResize (
            WindowResizeEvent & e )  [private]
```

This is a boolean function that will be called when the window is resized.

The function will check if the window is minimized and if it is it will return false. If the window is not minimized it will call the OnWindowResize function in the renderer class.

**Parameters**

| | | |
|---|---|---|
| in | *WindowResizeEvent&* | e - a reference to the window resize event. |

**Returns**

bool - returns true if the window resize event is handled here false if it needs to continue to be propogated.

#### 9.1.3.6 PushLayer()

```
void Fracture::Application::PushLayer (
            Layer * layer )
```

This is a function that will push a layer onto the application layer stack.

The function will push a layer onto the application layer stack. The layer is added to the first half of the stack. layers will be updated in the order they are added to the stack. So this layer will be updated first in the order they were pushed.

**See also**

Layer

LayerStack

**Parameters**

| | | |
|---|---|---|
| in | *Layer*∗ | layer - a pointer to the layer that will be pushed onto the stack. |

#### 9.1.3.7 PushOverlay()

```
void Fracture::Application::PushOverlay (
            Layer * layer )
```

This is a function that will push an overlay onto the application layer stack.

The function will push an overlay onto the application layer stack. The overlay is added to the second half of the stack. layers will be updated in the order they are added to the stack. So this overlay will be updated last in the order they were pushed.

**See also**

Layer

LayerStack

**Parameters**

| | | |
|---|---|---|
| in | *Layer*∗ | layer - a pointer to the overlay that will be pushed onto the stack. |

#### 9.1.3.8 Run()

```
void Fracture::Application::Run ( )
```

This is a function that is the main loop of the application.

The function will run the main loop of the application. The function will update the layers in the layer stack and render the ImGui UI. The function will also poll for events and dispatch them to the appropriate callback functions. In the future this function will also update the physics engine.

**See also**

LayerStack

ImGuiLayer

Event

Window

Renderer

### 9.1.4 Member Data Documentation

#### 9.1.4.1 m_ImGuiLayer

```
ImGuiLayer* Fracture::Application::m_ImGuiLayer  [private]
```

#### 9.1.4.2 m_isMinimized

```
bool Fracture::Application::m_isMinimized = false  [private]
```

this is a boolean that will be used to determine if the application is running or not.

#### 9.1.4.3 m_LastFrameTime

```
long long Fracture::Application::m_LastFrameTime = 0  [private]
```

this is a boolean that will be used to determine if the application is minimized or not.

#### 9.1.4.4 m_LayerStack

```
LayerStack Fracture::Application::m_LayerStack  [private]
```

The application layer stack. This will store all the layers that are currently active and will be updated every frame.

#### 9.1.4.5 m_Running

```
bool Fracture::Application::m_Running = true  [private]
```

A pointer to the ImGuiLayer object that is managed by the application class. This is used to render the ImGui UI.

#### 9.1.4.6 m_Window

```
Ref<Window> Fracture::Application::m_Window  [private]
```

A unique pointer to a window object tha is managed by the application class.

this is a unique pointer to a window object because we only want one window object in our application class and we want to manage it ourselves. When the application class is destroyed the window object will be destroyed as well

**See also**

Window

#### 9.1.4.7 s_Instance

```
Application * Fracture::Application::s_Instance = nullptr  [static], [private]
```

Stores the start time of the last frame. Used to calculate the delta time.

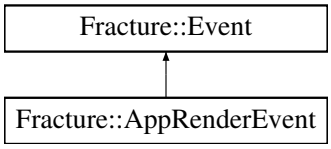
The documentation for this class was generated from the following files:

- Fracture/src/Fracture/Core/Application.h
- Fracture/src/Fracture/Core/Application.cpp

## 9.2 Fracture::AppRenderEvent Class Reference

`#include <ApplicationEvent.h>`

Inheritance diagram for Fracture::AppRenderEvent:




#### Public Member Functions

- AppRenderEvent ()

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.2.1 Constructor & Destructor Documentation

#### 9.2.1.1 AppRenderEvent()

`Fracture::AppRenderEvent::AppRenderEvent ( ) [inline]`

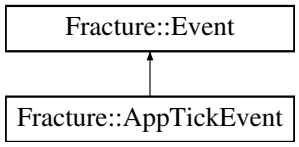
The documentation for this class was generated from the following file:

- Fracture/src/Fracture/Events/ApplicationEvent.h

## 9.3 Fracture::AppTickEvent Class Reference

`#include <ApplicationEvent.h>`

Inheritance diagram for Fracture::AppTickEvent:




#### Public Member Functions

- AppTickEvent ()

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.3.1 Constructor & Destructor Documentation

#### 9.3.1.1 AppTickEvent()

`Fracture::AppTickEvent::AppTickEvent ( ) [inline]`

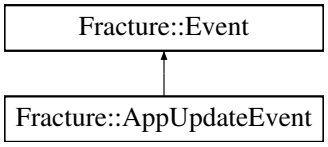
The documentation for this class was generated from the following file:

- Fracture/src/Fracture/Events/ApplicationEvent.h

## 9.4 Fracture::AppUpdateEvent Class Reference

`#include <ApplicationEvent.h>`

Inheritance diagram for Fracture::AppUpdateEvent:




#### Public Member Functions

- AppUpdateEvent ()

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.4.1 Constructor & Destructor Documentation

#### 9.4.1.1 AppUpdateEvent()

`Fracture::AppUpdateEvent::AppUpdateEvent ( ) [inline]`

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

- Fracture/src/Fracture/Events/ApplicationEvent.h

## 9.5 Fracture::BufferElement Struct Reference

The BufferElement struct is used to store the elements of the vertex buffer layout.

`#include <Buffer.h>`

### Public Member Functions

- BufferElement ()=default

  *Whether the element is normalized or not.*
- BufferElement (ShaderDataType type, const std::string &name, bool normalized=false)

  *Constructor for the BufferElement struct. Takes the data type, name and whether the element is normalized or not as parameters.*
- uint32_t GetElementCount () const

  *Function that returns the number of elements in a specific data type.*

### Public Attributes

- std::string Name
- uint32_t Offset

  *The name of the element.*
- uint32_t Size

  *The offset of the element in the vertex buffer layout.*
- ShaderDataType Type

  *The size of the element in the vertex buffer layout in bytes.*
- bool Normalized

  *The data type of the element in the vertex buffer layout.*

### 9.5.1 Detailed Description

The BufferElement struct is used to store the elements of the vertex buffer layout.

**See also**

BufferLayout

### 9.5.2 Constructor & Destructor Documentation

#### 9.5.2.1 BufferElement() [1/2]

```
Fracture::BufferElement::BufferElement ( )  [default]
```

Whether the element is normalized or not.

Default constructor for the BufferElement struct.

#### 9.5.2.2 BufferElement() [2/2]

```
Fracture::BufferElement::BufferElement (
            ShaderDataType type,
            const std::string & name,
            bool normalized = false )  [inline]
```

Constructor for the BufferElement struct. Takes the data type, name and whether the element is normalized or not as parameters.

**Parameters**

| | | |
|---|---|---|
| in | *ShaderDataType* | type: The data type of the element in the vertex buffer layout. |
| in | *const* | std::string& name: The name of the element. |
| in | *bool* | normalized: Whether the element is normalized or not. |

## 9.5.3 Member Function Documentation

### 9.5.3.1 GetElementCount()

```
uint32_t Fracture::BufferElement::GetElementCount ( ) const  [inline]
```

Function that returns the number of elements in a specific data type.

## 9.5.4 Member Data Documentation

### 9.5.4.1 Name

```
std::string Fracture::BufferElement::Name
```

### 9.5.4.2 Normalized

```
bool Fracture::BufferElement::Normalized
```

The data type of the element in the vertex buffer layout.

### 9.5.4.3 Offset

```
uint32_t Fracture::BufferElement::Offset
```

The name of the element.

### 9.5.4.4 Size

```
uint32_t Fracture::BufferElement::Size
```

The offset of the element in the vertex buffer layout.

### 9.5.4.5 Type

```
ShaderDataType Fracture::BufferElement::Type
```

The size of the element in the vertex buffer layout in bytes.

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

- Fracture/src/Fracture/Renderer/Buffer.h

## 9.6 Fracture::BufferLayout Class Reference

The BufferLayout class is used to store the layout of the vertex buffer. Each vertex buffer has a buffer layout.

`#include <Buffer.h>`

### Public Member Functions

- BufferLayout ()

  *Default constructor for the BufferLayout class.*
- BufferLayout (const std::initializer_list< BufferElement > &elements)

  *Constructor for the BufferLayout class. Takes a vector of elements as a parameter.*
- std::vector< BufferElement >::iterator begin ()

  *function that returns an iterator to the beginning of the vector of elements.*
- std::vector< BufferElement >::iterator end ()

  *function that returns an iterator to the end of the vector of elements.*
- std::vector< BufferElement >::const_iterator begin () const

  *a const iterator to the beginning of the vector of elements. For use with const objects.*
- std::vector< BufferElement >::const_iterator end () const

  *a const iterator to the end of the vector of elements. For use with const objects.*
- uint32_t GetStride () const

  *function that returns the size of the vertex buffer layout in bytes.*
- const std::vector< BufferElement > & GetElements () const

  *function that returns the vector of elements in the vertex buffer layout.*

### Private Member Functions

- void CalculateOffsetsAndStride ()

  *Function that calculates the offsets and stride of the vertex buffer layout.*

### Private Attributes

- std::vector< BufferElement > m_Elements
- uint32_t m_Stride = 0

  *Vector of elements in the buffer layout.*

### 9.6.1 Detailed Description

The BufferLayout class is used to store the layout of the vertex buffer. Each vertex buffer has a buffer layout.

**See also**

VertexBuffer

## 9.6.2 Constructor & Destructor Documentation

### 9.6.2.1 BufferLayout() [1/2]

```
Fracture::BufferLayout::BufferLayout ( )  [inline]
```

Default constructor for the BufferLayout class.

### 9.6.2.2 BufferLayout() [2/2]

```
Fracture::BufferLayout::BufferLayout (
            const std::initializer_list< BufferElement > & elements )  [inline]
```

Constructor for the BufferLayout class. Takes a vector of elements as a parameter.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::vector<BufferElement>& elements: The vector of elements. |

### 9.6.3 Member Function Documentation

#### 9.6.3.1 begin() [1/2]

```
std::vector< BufferElement >::iterator Fracture::BufferLayout::begin ( )  [inline]
```

function that returns an iterator to the beginning of the vector of elements.

**Returns**

std::vector<BufferElement>::iterator: An iterator to the beginning of the vector of elements.

#### 9.6.3.2 begin() [2/2]

```
std::vector< BufferElement >::const_iterator Fracture::BufferLayout::begin ( ) const  [inline]
```

a const iterator to the beginning of the vector of elements. For use with const objects.

**Returns**

std::vector<BufferElement>::const_iterator: A const iterator to the beginning of the vector of elements.

#### 9.6.3.3 CalculateOffsetsAndStride()

```
void Fracture::BufferLayout::CalculateOffsetsAndStride ( )  [inline], [private]
```

Function that calculates the offsets and stride of the vertex buffer layout.

Calculates the offset of each element in the order they are stored in the vector of elements. Also calculates the stride of the vertex buffer layout.

#### 9.6.3.4 end() [1/2]

```
std::vector< BufferElement >::iterator Fracture::BufferLayout::end ( )  [inline]
```

function that returns an iterator to the end of the vector of elements.

**Returns**

std::vector<BufferElement>::iterator: An iterator to the end of the vector of elements.

#### 9.6.3.5 end() [2/2]

```
std::vector< BufferElement >::const_iterator Fracture::BufferLayout::end ( ) const  [inline]
```

a const iterator to the end of the vector of elements. For use with const objects.

**Returns**

std::vector<BufferElement>::const_iterator: A const iterator to the end of the vector of elements.

#### 9.6.3.6 GetElements()

```
const std::vector< BufferElement > & Fracture::BufferLayout::GetElements ( ) const  [inline]
```

function that returns the vector of elements in the vertex buffer layout.

**Returns**

const std::vector<BufferElement>&: The vector of elements in the vertex buffer layout.

#### 9.6.3.7 GetStride()

```
uint32_t Fracture::BufferLayout::GetStride ( ) const  [inline]
```

function that returns the size of the vertex buffer layout in bytes.

### 9.6.4 Member Data Documentation

#### 9.6.4.1 m_Elements

```
std::vector<BufferElement> Fracture::BufferLayout::m_Elements  [private]
```

#### 9.6.4.2 m_Stride

```
uint32_t Fracture::BufferLayout::m_Stride = 0  [private]
```

Vector of elements in the buffer layout.

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

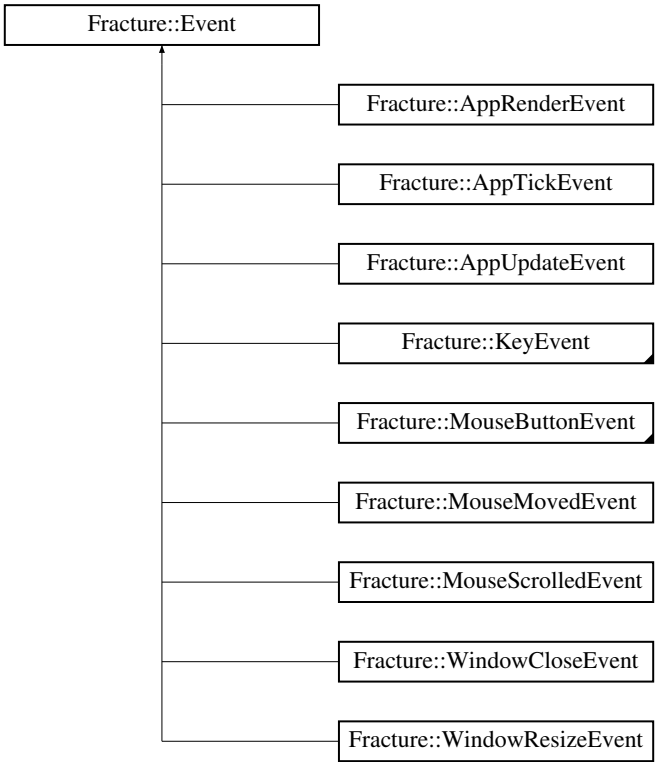
- Fracture/src/Fracture/Renderer/Buffer.h

## 9.7 Fracture::Event Class Reference

Base class for all events.

`#include <Event.h>`

Inheritance diagram for Fracture::Event:




### Public Member Functions

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Public Attributes

- bool Handled = false

**Friends**

- class EventDispatcher

## 9.7.1 Detailed Description

Base class for all events.

## 9.7.2 Member Function Documentation

### 9.7.2.1 GetCategoryFlags()

```
virtual int Fracture::Event::GetCategoryFlags ( ) const  [pure virtual]
```

Pure Virtual function, to get the category flags of the event.

**Returns**

int The category flags of the event

### 9.7.2.2 GetEventType()

```
virtual EventType Fracture::Event::GetEventType ( ) const  [pure virtual]
```

Pure Virtual function, to get the type of the event.

**See also**

EventType

**Returns**

EventType The type of the event

### 9.7.2.3 GetName()

```
virtual const char * Fracture::Event::GetName ( ) const  [pure virtual]
```

Pure Virtual function, to get the name of the event.

**Returns**

const char* The name of the event

### 9.7.2.4 IsInCategory()

```
bool Fracture::Event::IsInCategory (
            EventCategory category )  [inline]
```

Function to check if the event is in a certain category.

**Parameters**

| | |
|---|---|
| *EventCategory* | category: The category to check |

#### 9.7.2.5 ToString()

```
virtual std::string Fracture::Event::ToString ( ) const  [inline], [virtual]
```

Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.

**Returns**

std::string The string representation of the event

Reimplemented in Fracture::WindowResizeEvent, Fracture::KeyPressedEvent, Fracture::KeyReleasedEvent, Fracture::KeyTypedEvent, Fracture::MouseMovedEvent, Fracture::MouseScrolledEvent, Fracture::MouseButtonPressedEvent, and Fracture::MouseButtonReleasedEvent.

### 9.7.3 Friends And Related Symbol Documentation

#### 9.7.3.1 EventDispatcher

```
friend class EventDispatcher  [friend]
```

### 9.7.4 Member Data Documentation

#### 9.7.4.1 Handled

```
bool Fracture::Event::Handled = false
```

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

- Fracture/src/Fracture/Events/Event.h

## 9.8 Fracture::EventDispatcher Class Reference

```
#include <Event.h>
```

**Public Member Functions**

- EventDispatcher (Event &event)

  *Constructor for the event dispatcher.*
- template<typename T , typename F >
  bool Dispatch (const F &func)

  *Dispatch function that takes a function as a parameter and calls it if the event is of type T.*

**Private Attributes**

- Event & mEvent

## 9.8.1 Constructor & Destructor Documentation

### 9.8.1.1 EventDispatcher()

```
Fracture::EventDispatcher::EventDispatcher (
            Event & event )  [inline]
```

Constructor for the event dispatcher.

**Parameters**

| | |
|---|---|
| *Event&* | event: The event that is being dispatched |

## 9.8.2 Member Function Documentation

### 9.8.2.1 Dispatch()

```
template<typename T , typename F >
bool Fracture::EventDispatcher::Dispatch (
            const F & func )  [inline]
```

Dispatch function that takes a function as a parameter and calls it if the event is of type T.

**Template Parameters**

| | |
|---|---|
| *T* | The type of the event |
| *F* | The type of the function |

**Parameters**

| | |
|---|---|
| *const* | F& func: The function that is being to be called. It takes a reference of type T and returns a bool |

**Returns**

bool: Returns true if the event is of type T

## 9.8.3 Member Data Documentation

### 9.8.3.1 mEvent

```
Event& Fracture::EventDispatcher::mEvent  [private]
```

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

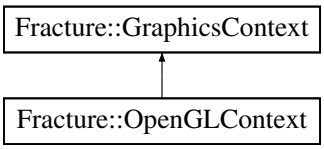
- Fracture/src/Fracture/Events/Event.h

## 9.9 Fracture::GraphicsContext Class Reference

The GraphicsContext class is an abstract class that is used to create a graphics context for the application. Each renderer will have its own implementation of the graphics context.

`#include <GraphicsContext.h>`

Inheritance diagram for Fracture::GraphicsContext:




### Public Member Functions

- virtual void Init ()=0

  *Function that initializes the graphics context. Must be implemented by each renderer.*
- virtual void SwapBuffers ()=0

  *Function that swaps the buffers of the graphics context and dispalys the image to the screen. Must be implemented by each renderer.*

### 9.9.1 Detailed Description

The GraphicsContext class is an abstract class that is used to create a graphics context for the application. Each renderer will have its own implementation of the graphics context.

**See also**

OpenGLContext

### 9.9.2 Member Function Documentation

#### 9.9.2.1 Init()

```
virtual void Fracture::GraphicsContext::Init ( )  [pure virtual]
```

Function that initializes the graphics context. Must be implemented by each renderer.

Implemented in Fracture::OpenGLContext.

#### 9.9.2.2 SwapBuffers()

```
virtual void Fracture::GraphicsContext::SwapBuffers ( )  [pure virtual]
```

Function that swaps the buffers of the graphics context and dispalys the image to the screen. Must be implemented by each renderer.

Implemented in Fracture::OpenGLContext.

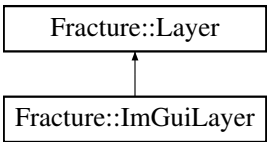
The documentation for this class was generated from the following file:

- Fracture/src/Fracture/Renderer/GraphicsContext.h

## 9.10 Fracture::ImGuiLayer Class Reference

`#include <ImGuiLayer.h>`

Inheritance diagram for Fracture::ImGuiLayer:




### Public Member Functions

- ImGuiLayer ()
- ~ImGuiLayer ()=default
- virtual void OnAttach () override

  *Called when the layer is attached to the layer stack.*
- virtual void OnDetach () override

  *Called when the layer is detached from the layer stack.*
- virtual void OnImGuiRender () override

  *Function called every frame by the application for rendering ImGui elements.*
- void Begin ()
- void End ()

### Public Member Functions inherited from Fracture::Layer

- Layer (const std::string &name="Layer")

  *Layer constructor. Currently only stores the name of the layer for debugging purposes.*
- virtual ~Layer ()=default
- virtual void OnUpdate (Utils::Timestep delta_time)

  *Function called every frame by the application.*
- virtual void OnEvent (Event &event)

  *the function called by the application OnEvent function for each layer in the layerstack with the current event being handled.*
- const std::string & GetName () const

  *Getter for the name of the layer. Mostly used for debugging purposes.*

### Private Attributes

- float m_Time = 0.0f

### Additional Inherited Members

### Protected Attributes inherited from Fracture::Layer

- std::string m_DebugName

### 9.10.1 Constructor & Destructor Documentation

#### 9.10.1.1 ImGuiLayer()

```
Fracture::ImGuiLayer::ImGuiLayer ( )
```

#### 9.10.1.2 ~ImGuiLayer()

```
Fracture::ImGuiLayer::~ImGuiLayer ( )  [default]
```

### 9.10.2 Member Function Documentation

#### 9.10.2.1 Begin()

```
void Fracture::ImGuiLayer::Begin ( )
```

#### 9.10.2.2 End()

```
void Fracture::ImGuiLayer::End ( )
```

#### 9.10.2.3 OnAttach()

```
void Fracture::ImGuiLayer::OnAttach ( )  [override], [virtual]
```

Called when the layer is attached to the layer stack.

Reimplemented from Fracture::Layer.

#### 9.10.2.4 OnDetach()

```
void Fracture::ImGuiLayer::OnDetach ( )  [override], [virtual]
```

Called when the layer is detached from the layer stack.

Reimplemented from Fracture::Layer.

#### 9.10.2.5 OnImGuiRender()

```
void Fracture::ImGuiLayer::OnImGuiRender ( )  [override], [virtual]
```

Function called every frame by the application for rendering ImGui elements.

**See also**

ImGui

Reimplemented from Fracture::Layer.

### 9.10.3 Member Data Documentation

#### 9.10.3.1 m_Time

```
float Fracture::ImGuiLayer::m_Time = 0.0f  [private]
```

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

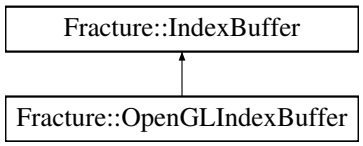
- Fracture/src/Fracture/ImGui/ImGuiLayer.h
- Fracture/src/Fracture/ImGui/ImGuiLayer.cpp

## 9.11 Fracture::IndexBuffer Class Reference

The IndexBuffer class is an abstract class that is used to store the index buffer. Each renderer will have its own implementation of the index buffer.

```
#include <Buffer.h>
```

Inheritance diagram for Fracture::IndexBuffer:




#### Public Member Functions

- virtual ~IndexBuffer ()=default
- virtual void SetData (const void *data, uint32_t size)=0
- virtual void Bind () const =0
- virtual void Unbind () const =0
- virtual uint32_t GetCount () const =0

#### Static Public Member Functions

- static Ref< IndexBuffer > Create (uint32_t *indices, uint32_t size)

  *Function that creates an index buffer. This function will create an index buffer based on the platform that the application is running on.*

### 9.11.1 Detailed Description

The IndexBuffer class is an abstract class that is used to store the index buffer. Each renderer will have its own implementation of the index buffer.

**See also**

OpenGLIndexBuffer

## 9.11.2 Constructor & Destructor Documentation

### 9.11.2.1 ~IndexBuffer()

```
virtual Fracture::IndexBuffer::~IndexBuffer ( )  [virtual], [default]
```

## 9.11.3 Member Function Documentation

### 9.11.3.1 Bind()

```
virtual void Fracture::IndexBuffer::Bind ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLIndexBuffer.

### 9.11.3.2 Create()

```
Ref< IndexBuffer > Fracture::IndexBuffer::Create (
            uint32_t * indices,
            uint32_t size )  [static]
```

Function that creates an index buffer. This function will create an index buffer based on the platform that the application is running on.

We check the renderer api that is being used and create the appropriate index buffer for that renderer.

**Todo** : Currently only supports uint32_t indices. We need to add support for other types of indices.

**See also**

OpenGLIndexBuffer

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t** | indices: The indices of the index buffer. |
| in | *uint32_t* | size: The size of the index buffer. |

**Returns**

A shared pointer to the index buffer.

### 9.11.3.3 GetCount()

```
virtual uint32_t Fracture::IndexBuffer::GetCount ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLIndexBuffer.

#### 9.11.3.4 SetData()

```
virtual void Fracture::IndexBuffer::SetData (
            const void * data,
            uint32_t size )  [pure virtual]
```

Implemented in Fracture::OpenGLIndexBuffer.

#### 9.11.3.5 Unbind()

```
virtual void Fracture::IndexBuffer::Unbind ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLIndexBuffer.

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

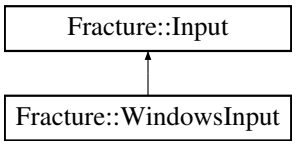
- Fracture/src/Fracture/Renderer/Buffer.h
- Fracture/src/Fracture/Renderer/Buffer.cpp

## 9.12 Fracture::Input Class Reference

The base class for Input polling. This class will be implemented per platform.

`#include <Input.h>`

Inheritance diagram for Fracture::Input:




### Public Member Functions

- Input (const Input &)=delete

  *Deleted copy constructor so that we can not copy this class since it is a singleton.*
- Input & operator= (const Input &)=delete

  *Deleted assignment operator so that we can not copy this class since it is a singleton.*

### Static Public Member Functions

- static bool IsKeyPressed (int keyCode)

  *Static function that returns if a key is pressed or not.*
- static bool IsMouseButtonPressed (int button)

  *Static function that returns if a mouse button is pressed or not.*
- static float GetMouseX ()

  *Static function that returns the current x coordinate of the mouse.*
- static float GetMouseY ()

  *Static function that returns the current y coordinate of the mouse.*
- static std::pair< float, float > GetMousePosition ()

  *Static function that returns the current x and y coordinates of the mouse at once.*

#### Protected Member Functions

- Input ()=default

  *protected constructor so that only the child classes can create an instance of this class.*
- virtual bool IsKeyPressedImpl (int keyCode)=0
- virtual bool IsMouseButtonPressedImpl (int button)=0
- virtual float GetMouseXImpl ()=0
- virtual float GetMouseYImpl ()=0
- virtual std::pair< float, float > GetMousePositionImpl ()=0

#### Static Private Attributes

- static Scope< Input > s_Instance = CreateScope<WindowsInput>()

### 9.12.1 Detailed Description

The base class for Input polling. This class will be implemented per platform.

This class will be implemented per platform. This is a singleton class.

**See also**

WindowsInput

### 9.12.2 Constructor & Destructor Documentation

#### 9.12.2.1 Input() [1/2]

```
Fracture::Input::Input ( )  [protected], [default]
```

protected constructor so that only the child classes can create an instance of this class.

#### 9.12.2.2 Input() [2/2]

```
Fracture::Input::Input (
            const Input &  )  [delete]
```

Deleted copy constructor so that we can not copy this class since it is a singleton.

### 9.12.3 Member Function Documentation

#### 9.12.3.1 GetMousePosition()

```
static std::pair< float, float > Fracture::Input::GetMousePosition ( )  [inline], [static]
```

Static function that returns the current x and y coordinates of the mouse at once.

**Returns**

std::pair<float, float> the current x and y coordinates of the mouse

#### 9.12.3.2 GetMousePositionImpl()

```
virtual std::pair< float, float > Fracture::Input::GetMousePositionImpl ( )  [protected],
[pure virtual]
```

Implemented in Fracture::WindowsInput.

#### 9.12.3.3 GetMouseX()

```
static float Fracture::Input::GetMouseX ( )  [inline], [static]
```

Static function that returns the current x coordinate of the mouse.

**Returns**

float the current x coordinate of the mouse

#### 9.12.3.4 GetMouseXImpl()

```
virtual float Fracture::Input::GetMouseXImpl ( )  [protected], [pure virtual]
```

Implemented in Fracture::WindowsInput.

#### 9.12.3.5 GetMouseY()

```
static float Fracture::Input::GetMouseY ( )  [inline], [static]
```

Static function that returns the current y coordinate of the mouse.

**Returns**

float the current y coordinate of the mouse

#### 9.12.3.6 GetMouseYImpl()

```
virtual float Fracture::Input::GetMouseYImpl ( )  [protected], [pure virtual]
```

Implemented in Fracture::WindowsInput.

#### 9.12.3.7 IsKeyPressed()

```
static bool Fracture::Input::IsKeyPressed (
            int keyCode )  [inline], [static]
```

Static function that returns if a key is pressed or not.

**Parameters**

| | | |
|---|---|---|
| in | *int* | keyCode the key code of the key that we want to check if it is pressed |

**Returns**

bool true if the key is pressed, false otherwise

#### 9.12.3.8 IsKeyPressedImpl()

```
virtual bool Fracture::Input::IsKeyPressedImpl (
            int keyCode )  [protected], [pure virtual]
```

Implemented in Fracture::WindowsInput.

#### 9.12.3.9 IsMouseButtonPressed()

```
static bool Fracture::Input::IsMouseButtonPressed (
            int button )  [inline], [static]
```

Static function that returns if a mouse button is pressed or not.

**Parameters**

| | | |
|---|---|---|
| in | *int* | button the mouse button code of the mouse button that we want to check if it is pressed |

**Returns**

bool true if the mouse button is pressed, false otherwise

#### 9.12.3.10 IsMouseButtonPressedImpl()

```
virtual bool Fracture::Input::IsMouseButtonPressedImpl (
            int button )  [protected], [pure virtual]
```

Implemented in Fracture::WindowsInput.

#### 9.12.3.11 operator=()

```
Input & Fracture::Input::operator= (
            const Input &  )  [delete]
```

Deleted assignment operator so that we can not copy this class since it is a singleton.

### 9.12.4 Member Data Documentation

#### 9.12.4.1 s_Instance

```
Scope< Input > Fracture::Input::s_Instance = CreateScope<WindowsInput>()  [static], [private]
```

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

- Fracture/src/Fracture/Input/Input.h
- Fracture/src/Platform/Windows/WindowsInput.cpp

## 9.13 Fracture::Utils::InstrumentationSession Struct Reference

```
#include <Instrumentation.h>
```

#### Public Attributes

- std::string Name

### 9.13.1 Member Data Documentation

#### 9.13.1.1 Name

```
std::string Fracture::Utils::InstrumentationSession::Name
```

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

- Fracture/src/Fracture/Utils/Instrumentation.h

## 9.14 Fracture::Utils::InstrumentationTimer Class Reference

```
#include <Instrumentation.h>
```

#### Public Member Functions

- InstrumentationTimer (const char *name)
- ~InstrumentationTimer ()
- void Stop ()

#### Private Attributes

- const char * m_Name
- std::chrono::time_point< std::chrono::high_resolution_clock > m_StartTimepoint
- bool m_Stopped

### 9.14.1 Constructor & Destructor Documentation

#### 9.14.1.1 InstrumentationTimer()

```
Fracture::Utils::InstrumentationTimer::InstrumentationTimer (
            const char * name )  [inline]
```

#### 9.14.1.2 ~InstrumentationTimer()

```
Fracture::Utils::InstrumentationTimer::~InstrumentationTimer ( )  [inline]
```

### 9.14.2 Member Function Documentation

#### 9.14.2.1 Stop()

```
void Fracture::Utils::InstrumentationTimer::Stop ( )  [inline]
```

### 9.14.3 Member Data Documentation

#### 9.14.3.1 m_Name

```
const char* Fracture::Utils::InstrumentationTimer::m_Name  [private]
```

#### 9.14.3.2 m_StartTimepoint

```
std::chrono::time_point<std::chrono::high_resolution_clock> Fracture::Utils::Instrumentation↩
Timer::m_StartTimepoint  [private]
```

#### 9.14.3.3 m_Stopped

```
bool Fracture::Utils::InstrumentationTimer::m_Stopped  [private]
```

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

- Fracture/src/Fracture/Utils/Instrumentation.h

## 9.15 Fracture::Utils::Instrumentor Class Reference

```
#include <Instrumentation.h>
```

**Public Member Functions**

- Instrumentor ()
- void BeginSession (const std::string &name, const std::string &filepath="../Logs/results.json")
- void EndSession ()
- void WriteProfile (const ProfileResult &result)
- void WriteHeader ()
- void WriteFooter ()

**Static Public Member Functions**

- static Instrumentor & Get ()

**Private Attributes**

- InstrumentationSession * m_CurrentSession
- std::ofstream m_OutputStream
- int m_ProfileCount

## 9.15.1 Constructor & Destructor Documentation

### 9.15.1.1 Instrumentor()

```
Fracture::Utils::Instrumentor::Instrumentor ( )  [inline]
```

## 9.15.2 Member Function Documentation

### 9.15.2.1 BeginSession()

```
void Fracture::Utils::Instrumentor::BeginSession (
            const std::string & name,
            const std::string & filepath = "../Logs/results.json" )  [inline]
```

### 9.15.2.2 EndSession()

```
void Fracture::Utils::Instrumentor::EndSession ( )  [inline]
```

### 9.15.2.3 Get()

```
static Instrumentor & Fracture::Utils::Instrumentor::Get ( )  [inline], [static]
```

### 9.15.2.4 WriteFooter()

```
void Fracture::Utils::Instrumentor::WriteFooter ( )  [inline]
```

#### 9.15.2.5 WriteHeader()

```
void Fracture::Utils::Instrumentor::WriteHeader ( )  [inline]
```

#### 9.15.2.6 WriteProfile()

```
void Fracture::Utils::Instrumentor::WriteProfile (
            const ProfileResult & result )  [inline]
```

### 9.15.3 Member Data Documentation

#### 9.15.3.1 m_CurrentSession

```
InstrumentationSession* Fracture::Utils::Instrumentor::m_CurrentSession  [private]
```

#### 9.15.3.2 m_OutputStream

```
std::ofstream Fracture::Utils::Instrumentor::m_OutputStream  [private]
```

#### 9.15.3.3 m_ProfileCount

```
int Fracture::Utils::Instrumentor::m_ProfileCount  [private]
```

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

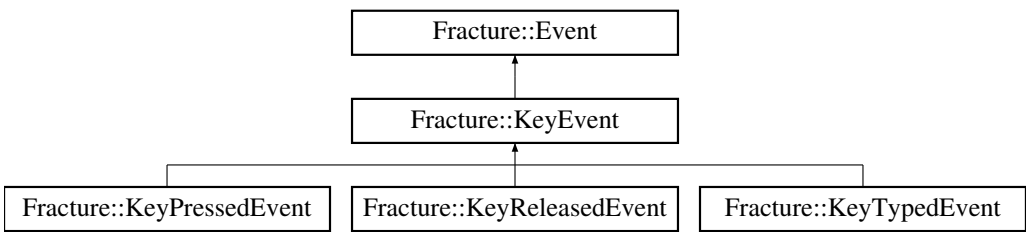
- Fracture/src/Fracture/Utils/Instrumentation.h

## 9.16 Fracture::KeyEvent Class Reference

the base class for KeyEvents

```
#include <KeyEvent.h>
```

Inheritance diagram for Fracture::KeyEvent:



#### Public Member Functions

- int GetKeyCode () const

  *return the key code of the key that was pressed*
- int GetKeyMods () const

  *return the mods of the key that was pressed*

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Protected Member Functions

- KeyEvent (int keyCode, int mods)

#### Protected Attributes

- int m_KeyCode
- int m_Mods

  *Stores the key code of the key that was pressed.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

## 9.16.1 Detailed Description

the base class for KeyEvents

**See also**

Event

## 9.16.2 Constructor & Destructor Documentation

### 9.16.2.1 KeyEvent()

```
Fracture::KeyEvent::KeyEvent (
            int keyCode,
            int mods )  [inline], [protected]
```

@breif Protected constructor so that only the child classes can create an instance of this class

**Parameters**

| | | |
|---|---|---|
| in | *int* | keyCode the key code of the key that was pressed |
| in | *int* | mods the mods of the key that was pressed |

## 9.16.3 Member Function Documentation

### 9.16.3.1 GetKeyCode()

```
int Fracture::KeyEvent::GetKeyCode ( ) const  [inline]
```

return the key code of the key that was pressed

**Returns**

int the key code of the key that was pressed

### 9.16.3.2 GetKeyMods()

```
int Fracture::KeyEvent::GetKeyMods ( ) const  [inline]
```

return the mods of the key that was pressed

**Returns**

int the mods of the key that was pressed

## 9.16.4 Member Data Documentation

### 9.16.4.1 m_KeyCode

```
int Fracture::KeyEvent::m_KeyCode  [protected]
```

### 9.16.4.2 m_Mods

```
int Fracture::KeyEvent::m_Mods  [protected]
```

Stores the key code of the key that was pressed.

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

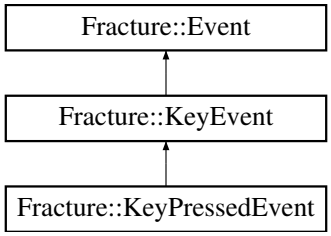
- Fracture/src/Fracture/Events/KeyEvent.h

# 9.17 Fracture::KeyPressedEvent Class Reference

Event class for when a key is pressed.

`#include <KeyEvent.h>`

Inheritance diagram for Fracture::KeyPressedEvent:




### Public Member Functions

- KeyPressedEvent (int keyCode, bool repeatCount, int mods)

  *Constructor for the KeyPressedEvent.*
- bool IsRepeated () const

  *getter for checking if this is a repeated key press or the first time the key was pressed*
- std::string ToString () const override

  *returns a string representation of the event*

## Public Member Functions inherited from Fracture::KeyEvent

- int GetKeyCode () const

  *return the key code of the key that was pressed*
- int GetKeyMods () const

  *return the mods of the key that was pressed*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Private Attributes

- bool m_IsRepeated

**Additional Inherited Members**

## Public Attributes inherited from Fracture::Event

- bool Handled = false

## Protected Member Functions inherited from Fracture::KeyEvent

- KeyEvent (int keyCode, int mods)

## Protected Attributes inherited from Fracture::KeyEvent

- int m_KeyCode
- int m_Mods

  *Stores the key code of the key that was pressed.*

### 9.17.1 Detailed Description

Event class for when a key is pressed.

**See also**

KeyEvent

Event

### 9.17.2 Constructor & Destructor Documentation

#### 9.17.2.1 KeyPressedEvent()

```
Fracture::KeyPressedEvent::KeyPressedEvent (
            int keyCode,
            bool repeatCount,
            int mods )  [inline]
```

Constructor for the KeyPressedEvent.

When we press a key a signal will be sent. If we keep it pressed after a certain amount of time the signal will be sent again and this is the repeat.

**Parameters**

| | | |
|---|---|---|
| in | *int* | keyCode the key code of the key that was pressed |
| in | *bool* | repeatCount the number of times the key was pressed |

### 9.17.3 Member Function Documentation

#### 9.17.3.1 IsRepeated()

```
bool Fracture::KeyPressedEvent::IsRepeated ( ) const  [inline]
```

getter for checking if this is a repeated key press or the first time the key was pressed

**Returns**

bool true if the key was pressed more than once, false otherwise

#### 9.17.3.2 ToString()

```
std::string Fracture::KeyPressedEvent::ToString ( ) const  [inline], [override], [virtual]
```

returns a string representation of the event

**Returns**

std::string the string representation of the event

Reimplemented from Fracture::Event.

### 9.17.4 Member Data Documentation

#### 9.17.4.1 m_IsRepeated

```
bool Fracture::KeyPressedEvent::m_IsRepeated  [private]
```

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

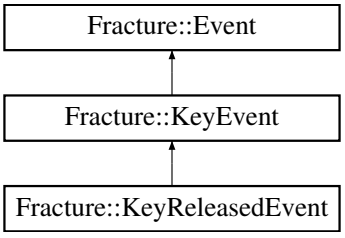
- Fracture/src/Fracture/Events/KeyEvent.h

## 9.18 Fracture::KeyReleasedEvent Class Reference

Event class for when a key is released.

```
#include <KeyEvent.h>
```

Inheritance diagram for Fracture::KeyReleasedEvent:



#### Public Member Functions

- KeyReleasedEvent (int keyCode, int mods)

  *Constructor for the KeyReleasedEvent.*
- std::string ToString () const override

  *returns a string representation of the event*

### Public Member Functions inherited from Fracture::KeyEvent

- int GetKeyCode () const

  *return the key code of the key that was pressed*
- int GetKeyMods () const

  *return the mods of the key that was pressed*

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

### Protected Member Functions inherited from Fracture::KeyEvent

- KeyEvent (int keyCode, int mods)

### Protected Attributes inherited from Fracture::KeyEvent

- int m_KeyCode
- int m_Mods

  *Stores the key code of the key that was pressed.*

## 9.18.1 Detailed Description

Event class for when a key is released.

**See also**

KeyEvent

Event

### 9.18.2 Constructor & Destructor Documentation

#### 9.18.2.1 KeyReleasedEvent()

```
Fracture::KeyReleasedEvent::KeyReleasedEvent (
            int keyCode,
            int mods )  [inline]
```

Constructor for the KeyReleasedEvent.

**Parameters**

| | | |
|---|---|---|
| in | *int* | keyCode the key code of the key that was pressed |
| in | *int* | mods the mods of the key that was pressed |

### 9.18.3 Member Function Documentation

#### 9.18.3.1 ToString()

```
std::string Fracture::KeyReleasedEvent::ToString ( ) const  [inline], [override], [virtual]
```

returns a string representation of the event

Reimplemented from Fracture::Event.

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

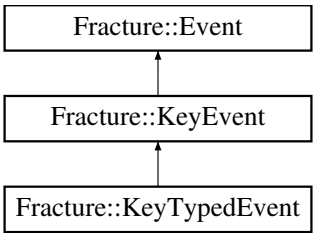
- Fracture/src/Fracture/Events/KeyEvent.h

## 9.19 Fracture::KeyTypedEvent Class Reference

Event class for when a key is typed.

```
#include <KeyEvent.h>
```

Inheritance diagram for Fracture::KeyTypedEvent:



#### Public Member Functions

- KeyTypedEvent (int keyCode)
- std::string ToString () const override

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*

### Public Member Functions inherited from Fracture::KeyEvent

- int GetKeyCode () const

  *return the key code of the key that was pressed*
- int GetKeyMods () const

  *return the mods of the key that was pressed*

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

#### Additional Inherited Members

### Public Attributes inherited from Fracture::Event

- bool Handled = false

### Protected Member Functions inherited from Fracture::KeyEvent

- KeyEvent (int keyCode, int mods)

### Protected Attributes inherited from Fracture::KeyEvent

- int m_KeyCode
- int m_Mods

  *Stores the key code of the key that was pressed.*

## 9.19.1 Detailed Description

Event class for when a key is typed.

**See also**

KeyEvent

Event

### 9.19.2 Constructor & Destructor Documentation

#### 9.19.2.1 KeyTypedEvent()

```
Fracture::KeyTypedEvent::KeyTypedEvent (
            int keyCode )  [inline]
```

### 9.19.3 Member Function Documentation

#### 9.19.3.1 ToString()

```
std::string Fracture::KeyTypedEvent::ToString ( ) const  [inline], [override], [virtual]
```

Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.

**Returns**

std::string The string representation of the event

Reimplemented from Fracture::Event.

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

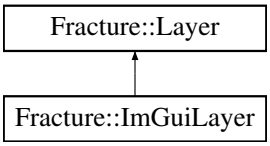
- Fracture/src/Fracture/Events/KeyEvent.h

## 9.20 Fracture::Layer Class Reference

The Layer class is the base class for all layers in the engine. Layers are used to separate different parts of the application and set an order of execution.

```
#include <Layer.h>
```

Inheritance diagram for Fracture::Layer:



**Public Member Functions**

- Layer (const std::string &name="Layer")

  *Layer constructor. Currently only stores the name of the layer for debugging purposes.*
- virtual ~Layer ()=default
- virtual void OnAttach ()

  *Called when the layer is attached to the layer stack.*
- virtual void OnDetach ()

  *Called when the layer is detached from the layer stack.*
- virtual void OnUpdate (Utils::Timestep delta_time)

  *Function called every frame by the application.*
- virtual void OnEvent (Event &event)

  *the function called by the application OnEvent function for each layer in the layerstack with the current event being handled.*
- virtual void OnImGuiRender ()

  *Function called every frame by the application for rendering ImGui elements.*
- const std::string & GetName () const

  *Getter for the name of the layer. Mostly used for debugging purposes.*

**Protected Attributes**

- std::string m_DebugName

### 9.20.1 Detailed Description

The Layer class is the base class for all layers in the engine. Layers are used to separate different parts of the application and set an order of execution.

**See also**

LayerStack

### 9.20.2 Constructor & Destructor Documentation

#### 9.20.2.1 Layer()

```
Fracture::Layer::Layer (
            const std::string & name = "Layer" )
```

Layer constructor. Currently only stores the name of the layer for debugging purposes.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the layer. |

#### 9.20.2.2 ~Layer()

```
virtual Fracture::Layer::~Layer ( )  [virtual], [default]
```

## 9.20.3 Member Function Documentation

### 9.20.3.1 GetName()

```
const std::string & Fracture::Layer::GetName ( ) const  [inline]
```

Getter for the name of the layer. Mostly used for debugging purposes.

**Returns**

const std::string&: The name of the layer.

### 9.20.3.2 OnAttach()

```
virtual void Fracture::Layer::OnAttach ( )  [inline], [virtual]
```

Called when the layer is attached to the layer stack.

Reimplemented in Fracture::ImGuiLayer.

### 9.20.3.3 OnDetach()

```
virtual void Fracture::Layer::OnDetach ( )  [inline], [virtual]
```

Called when the layer is detached from the layer stack.

Reimplemented in Fracture::ImGuiLayer.

### 9.20.3.4 OnEvent()

```
virtual void Fracture::Layer::OnEvent (
            Event & event )  [inline], [virtual]
```

the function called by the application OnEvent function for each layer in the layerstack with the current event being handled.

**See also**

Application::OnEvent

Event

**Parameters**

| | | |
|---|---|---|
| in | *Event&* | event: The event being handled. |

#### 9.20.3.5 OnImGuiRender()

```
virtual void Fracture::Layer::OnImGuiRender ( )  [inline], [virtual]
```

Function called every frame by the application for rendering ImGui elements.

**See also**

ImGui

Reimplemented in Fracture::ImGuiLayer.

#### 9.20.3.6 OnUpdate()

```
virtual void Fracture::Layer::OnUpdate (
            Utils::Timestep delta_time )  [inline], [virtual]
```

Function called every frame by the application.

**Parameters**

| | | |
|---|---|---|
| in | *Timestep* | delta_time: The time passed since the last frame |

### 9.20.4 Member Data Documentation

#### 9.20.4.1 m_DebugName

```
std::string Fracture::Layer::m_DebugName  [protected]
```

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

- Fracture/src/Fracture/Core/Layer.h
- Fracture/src/Fracture/Core/Layer.cpp

## 9.21 Fracture::LayerStack Class Reference

The LayerStack class is used to store all the layers that are currently active.

```
#include <LayerStack.h>
```

**Public Member Functions**

- LayerStack ()
- ~LayerStack ()

  *Constructor.*
- void PushLayer (Layer *layer)

  *Function that will attach a layer to the layer stack.*
- void PushOverlay (Layer *layer)

  *Function that will attach an overlay to the layer stack.*
- void PopLayer (Layer *layer)

  *Function that will detach a layer from the layer stack.*
- void PopOverlay (Layer *layer)

  *Function that will detach an overlay from the layer stack.*
- std::vector< Layer * >::iterator begin ()

  *create an iterator to the beginning of the layer stack.*
- std::vector< Layer * >::iterator end ()

  *create an iterator to the end of the layer stack.*

**Private Attributes**

- std::vector< Layer * > m_Layers
- uint32_t m_LayerInsertIndex = 0

  *The vector of Layer pointers that will hold the layers.*

## 9.21.1 Detailed Description

The LayerStack class is used to store all the layers that are currently active.

The LayerStack class is used to store all the layers that are currently active. The layers are stored in a vector of Layer pointers. The layers are stored in the first half of the vector and the overlays are stored in the second half of the vector. The demarcation between the layers and the overlays is stored in the m_LayerInsertIndex variable.

**See also**

Layer

## 9.21.2 Constructor & Destructor Documentation

### 9.21.2.1 LayerStack()

```
Fracture::LayerStack::LayerStack ( )
```

### 9.21.2.2 ~LayerStack()

```
Fracture::LayerStack::~LayerStack ( )
```

Constructor.

Destructor of the LayerStack class. This will detach all the layers from the layer stack and delete them.

**Todo** : Should we return the pointer to the layer instead of deleting it for the application to handle?

### 9.21.3 Member Function Documentation

#### 9.21.3.1 begin()

```
std::vector< Layer * >::iterator Fracture::LayerStack::begin ( )  [inline]
```

create an iterator to the beginning of the layer stack.

**Returns**

std::vector<Layer*>::iterator: An iterator to the beginning of the layer stack.

#### 9.21.3.2 end()

```
std::vector< Layer * >::iterator Fracture::LayerStack::end ( )  [inline]
```

create an iterator to the end of the layer stack.

**Returns**

std::vector<Layer*>::iterator: An iterator to the end of the layer stack.

#### 9.21.3.3 PopLayer()

```
void Fracture::LayerStack::PopLayer (
            Layer * layer )
```

Function that will detach a layer from the layer stack.

Here we will find the layer in the layer stack call its Layer::OnDetach function and erase it from the LayerStack.

**See also**

Layer

**Todo** : Should we return the pointer to the layer after detaching it?

**Parameters**

| | | |
|---|---|---|
| in | *Layer** | layer: The layer to be detached from the layer stack. |

#### 9.21.3.4 PopOverlay()

```
void Fracture::LayerStack::PopOverlay (
            Layer * layer )
```

Function that will detach an overlay from the layer stack.

Here we will find the overlay in the layer stack call its Layer::OnDetach function and erase it from the LayerStack.

**See also**

Layer

**Todo** : Should we return the pointer to the layer after detaching it?

**Parameters**

| | | |
|---|---|---|
| in | *Layer*∗ | layer: The overlay to be detached from the layer stack. |

#### 9.21.3.5 PushLayer()

```
void Fracture::LayerStack::PushLayer (
            Layer * layer )
```

Function that will attach a layer to the layer stack.

The function will attach a layer to the layer stack. The layer will be added to the first half of the layer stack.

**See also**

Layer

**Parameters**

| | | |
|---|---|---|
| in | *Layer*∗ | layer: The layer to be attached to the layer stack. |

#### 9.21.3.6 PushOverlay()

```
void Fracture::LayerStack::PushOverlay (
            Layer * layer )
```

Function that will attach an overlay to the layer stack.

The function will attach an overlay to the layer stack. The overlay will be emplaced to the back of the layer stack.

**See also**

Layer

**Parameters**

| | | |
|---|---|---|
| in | *Layer*∗ | layer: The overlay to be attached to the layer stack at the back. |

### 9.21.4 Member Data Documentation

#### 9.21.4.1 m_LayerInsertIndex

```
uint32_t Fracture::LayerStack::m_LayerInsertIndex = 0  [private]
```

The vector of Layer pointers that will hold the layers.

#### 9.21.4.2 m_Layers

```
std::vector<Layer*> Fracture::LayerStack::m_Layers  [private]
```

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

- Fracture/src/Fracture/Core/LayerStack.h
- Fracture/src/Fracture/Core/LayerStack.cpp

## 9.22 Fracture::Log Class Reference

The Log class is used to log messages to the console.

```
#include <Log.h>
```

#### Static Public Member Functions

- static void Init ()

  *Function that initializes the logging system. Sets up the core logger and the client logger with the appropriate formatting.*
- static Ref< spdlog::logger > & GetCoreLogger ()

  *Function that returns the core logger.*
- static Ref< spdlog::logger > & GetClientLogger ()

  *Function that returns the client logger.*

#### Static Private Attributes

- static Ref< spdlog::logger > s_ClientLogger
- static Ref< spdlog::logger > s_CoreLogger

  *The client logger.*

### 9.22.1 Detailed Description

The Log class is used to log messages to the console.

## 9.22.2 Member Function Documentation

### 9.22.2.1 GetClientLogger()

```
static Ref< spdlog::logger > & Fracture::Log::GetClientLogger ( )  [inline], [static]
```

Function that returns the client logger.

**Returns**

Ref<spdlog::logger>&: The client logger.

### 9.22.2.2 GetCoreLogger()

```
static Ref< spdlog::logger > & Fracture::Log::GetCoreLogger ( )  [inline], [static]
```

Function that returns the core logger.

**Returns**

Ref<spdlog::logger>&: The core logger.

### 9.22.2.3 Init()

```
void Fracture::Log::Init ( )  [static]
```

Function that initializes the logging system. Sets up the core logger and the client logger with the appropriate formatting.

## 9.22.3 Member Data Documentation

### 9.22.3.1 s_ClientLogger

```
Ref< spdlog::logger > Fracture::Log::s_ClientLogger  [static], [private]
```

### 9.22.3.2 s_CoreLogger

```
Ref< spdlog::logger > Fracture::Log::s_CoreLogger  [static], [private]
```

The client logger.

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

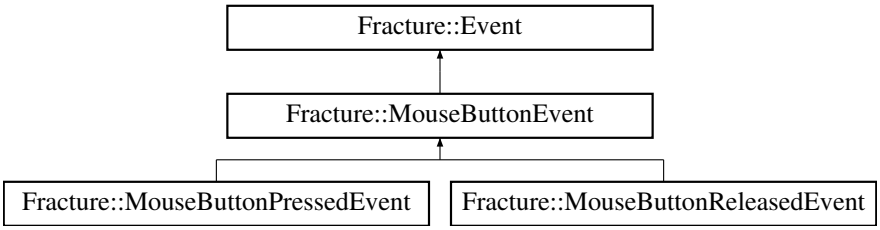
- Fracture/src/Fracture/Utils/Log.h
- Fracture/src/Fracture/Utils/Log.cpp

## 9.23 Fracture::MouseButtonEvent Class Reference

Base class for mouse button events.

`#include <MouseEvent.h>`

Inheritance diagram for Fracture::MouseButtonEvent:




### Public Member Functions

- int GetMouseButton () const

  *return the mouse code of the mouse button that was pressed*
- int GetMouseMod () const

  *return the mouse code of the mouse button that was pressed*

### Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Protected Member Functions

- MouseButtonEvent (int button, int mods)

### Protected Attributes

- int m_Button
- int m_Mods

  *The mouse button code of the mouse button that was pressed.*

**Additional Inherited Members**

## Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.23.1 Detailed Description

Base class for mouse button events.

Similar to KeyEvent, we have a base class for mouse button events and then we have child classes for the specific events

**See also**

Event

### 9.23.2 Constructor & Destructor Documentation

#### 9.23.2.1 MouseButtonEvent()

```
Fracture::MouseButtonEvent::MouseButtonEvent (
            int button,
            int mods )  [inline], [protected]
```

@breif Protected constructor so that only the child classes can create an instance of this class

**Parameters**

| | | |
|---|---|---|
| in | *int* | button the code of the mouse button that was pressed |
| in | *int* | mods the mods of the mouse button that was pressed |

### 9.23.3 Member Function Documentation

#### 9.23.3.1 GetMouseButton()

```
int Fracture::MouseButtonEvent::GetMouseButton ( ) const  [inline]
```

return the mouse code of the mouse button that was pressed

**Returns**

int the mouse code of the mouse button that was pressed

#### 9.23.3.2 GetMouseMod()

```
int Fracture::MouseButtonEvent::GetMouseMod ( ) const  [inline]
```

return the mouse code of the mouse button that was pressed

**Returns**

int the mouse code of the mouse button that was pressed

### 9.23.4 Member Data Documentation

#### 9.23.4.1 m_Button

```
int Fracture::MouseButtonEvent::m_Button  [protected]
```

#### 9.23.4.2 m_Mods

```
int Fracture::MouseButtonEvent::m_Mods  [protected]
```

The mouse button code of the mouse button that was pressed.

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

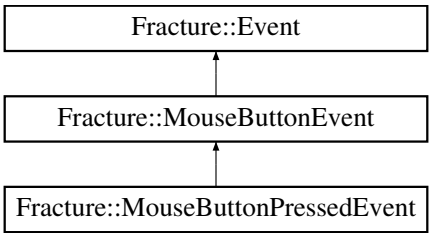
- Fracture/src/Fracture/Events/MouseEvent.h

## 9.24 Fracture::MouseButtonPressedEvent Class Reference

Event class for when a mouse button is pressed.

`#include <MouseEvent.h>`

Inheritance diagram for Fracture::MouseButtonPressedEvent:




**Public Member Functions**

- MouseButtonPressedEvent (int button, int mods)
- std::string ToString () const override

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*

## Public Member Functions inherited from Fracture::MouseButtonEvent

- int GetMouseButton () const

  *return the mouse code of the mouse button that was pressed*
- int GetMouseMod () const

  *return the mouse code of the mouse button that was pressed*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

## Protected Member Functions inherited from Fracture::MouseButtonEvent

- MouseButtonEvent (int button, int mods)

## Protected Attributes inherited from Fracture::MouseButtonEvent

- int m_Button
- int m_Mods

  *The mouse button code of the mouse button that was pressed.*

### 9.24.1 Detailed Description

Event class for when a mouse button is pressed.

**See also**

MouseButtonEvent

Event

### 9.24.2 Constructor & Destructor Documentation

#### 9.24.2.1 MouseButtonPressedEvent()

```
Fracture::MouseButtonPressedEvent::MouseButtonPressedEvent (
            int button,
            int mods )  [inline]
```

### 9.24.3 Member Function Documentation

#### 9.24.3.1 ToString()

```
std::string Fracture::MouseButtonPressedEvent::ToString ( ) const  [inline], [override], [virtual]
```

Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.

**Returns**

std::string The string representation of the event

Reimplemented from Fracture::Event.

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

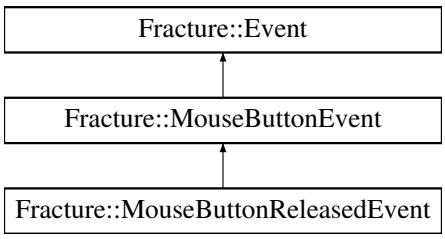
- Fracture/src/Fracture/Events/MouseEvent.h

## 9.25 Fracture::MouseButtonReleasedEvent Class Reference

Event class for when a mouse button is released.

```
#include <MouseEvent.h>
```

Inheritance diagram for Fracture::MouseButtonReleasedEvent:




**Public Member Functions**

- MouseButtonReleasedEvent (int button, int mods)
- std::string ToString () const override

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*

## Public Member Functions inherited from Fracture::MouseButtonEvent

- int GetMouseButton () const

  *return the mouse code of the mouse button that was pressed*
- int GetMouseMod () const

  *return the mouse code of the mouse button that was pressed*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

## Protected Member Functions inherited from Fracture::MouseButtonEvent

- MouseButtonEvent (int button, int mods)

## Protected Attributes inherited from Fracture::MouseButtonEvent

- int m_Button
- int m_Mods

  *The mouse button code of the mouse button that was pressed.*

### 9.25.1 Detailed Description

Event class for when a mouse button is released.

**See also**

MouseButtonEvent

Event

### 9.25.2 Constructor & Destructor Documentation

#### 9.25.2.1 MouseButtonReleasedEvent()

```
Fracture::MouseButtonReleasedEvent::MouseButtonReleasedEvent (
            int button,
            int mods )  [inline]
```

### 9.25.3 Member Function Documentation

#### 9.25.3.1 ToString()

```
std::string Fracture::MouseButtonReleasedEvent::ToString ( ) const  [inline], [override],
[virtual]
```

Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.

**Returns**

std::string The string representation of the event

Reimplemented from Fracture::Event.

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

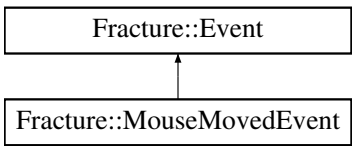
- Fracture/src/Fracture/Events/MouseEvent.h

## 9.26 Fracture::MouseMovedEvent Class Reference

Event for when the mouse is moved.

`#include <MouseEvent.h>`

Inheritance diagram for Fracture::MouseMovedEvent:




**Public Member Functions**

- MouseMovedEvent (float x, float y)

  *Constructor for the MouseMovedEvent.*
- float GetX () const

  *Getter for the x position of the mouse.*
- float GetY () const

  *Getter for the y position of the mouse.*
- std::string ToString () const override

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Private Attributes

- float m_MouseX
- float m_MouseY

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

## 9.26.1 Detailed Description

Event for when the mouse is moved.

**See also**

Event

## 9.26.2 Constructor & Destructor Documentation

### 9.26.2.1 MouseMovedEvent()

```
Fracture::MouseMovedEvent::MouseMovedEvent (
            float x,
            float y )  [inline]
```

Constructor for the MouseMovedEvent.

**Parameters**

| | |
|---|---|
| *x* | The x position of the mouse |
| *y* | The y position of the mouse |

## 9.26.3 Member Function Documentation

### 9.26.3.1 GetX()

```
float Fracture::MouseMovedEvent::GetX ( ) const  [inline]
```

Getter for the x position of the mouse.

**Returns**

float x position of the mouse

### 9.26.3.2 GetY()

```
float Fracture::MouseMovedEvent::GetY ( ) const  [inline]
```

Getter for the y position of the mouse.

**Returns**

float y position of the mouse

### 9.26.3.3 ToString()

```
std::string Fracture::MouseMovedEvent::ToString ( ) const  [inline], [override], [virtual]
```

Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.

**Returns**

std::string The string representation of the event

Reimplemented from Fracture::Event.

## 9.26.4 Member Data Documentation

### 9.26.4.1 m_MouseX

```
float Fracture::MouseMovedEvent::m_MouseX  [private]
```

### 9.26.4.2 m_MouseY

```
float Fracture::MouseMovedEvent::m_MouseY  [private]
```

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

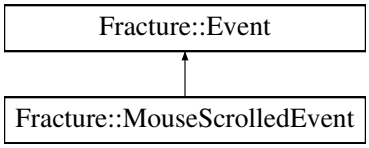
- Fracture/src/Fracture/Events/MouseEvent.h

# 9.27 Fracture::MouseScrolledEvent Class Reference

Event for when the mouse is scrolled.

`#include <MouseEvent.h>`

Inheritance diagram for Fracture::MouseScrolledEvent:




### Public Member Functions

- MouseScrolledEvent (float xOffset, float yOffset)
- float GetXOffset () const

  *Getter for the x offset of the mouse.*
- float GetYOffset () const

  *Getter for the y offset of the mouse.*
- std::string ToString () const override

  *Serialise the event data to string.*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Private Attributes

- float m_XOffset
- float m_YOffset

  *The X offset of the mouse scroll.*

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.27.1 Detailed Description

Event for when the mouse is scrolled.

**See also**

Event

### 9.27.2 Constructor & Destructor Documentation

#### 9.27.2.1 MouseScrolledEvent()

```
Fracture::MouseScrolledEvent::MouseScrolledEvent (
            float xOffset,
            float yOffset )  [inline]
```

### 9.27.3 Member Function Documentation

#### 9.27.3.1 GetXOffset()

```
float Fracture::MouseScrolledEvent::GetXOffset ( ) const  [inline]
```

Getter for the x offset of the mouse.

**Returns**

float x offset of the mouse

#### 9.27.3.2 GetYOffset()

```
float Fracture::MouseScrolledEvent::GetYOffset ( ) const  [inline]
```

Getter for the y offset of the mouse.

**Returns**

float y offset of the mouse

#### 9.27.3.3 ToString()

```
std::string Fracture::MouseScrolledEvent::ToString ( ) const  [inline], [override], [virtual]
```

Serialise the event data to string.

Reimplemented from Fracture::Event.

### 9.27.4 Member Data Documentation

#### 9.27.4.1 m_XOffset

```
float Fracture::MouseScrolledEvent::m_XOffset  [private]
```

#### 9.27.4.2 m_YOffset

```
float Fracture::MouseScrolledEvent::m_YOffset  [private]
```

The X offset of the mouse scroll.

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

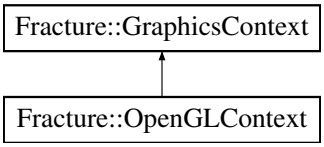
- Fracture/src/Fracture/Events/MouseEvent.h

## 9.28 Fracture::OpenGLContext Class Reference

The OpenGLContext class is an implementation of the GraphicsContext class for the OpenGL renderer.

```
#include <OpenGLContext.h>
```

Inheritance diagram for Fracture::OpenGLContext:




#### Public Member Functions

- OpenGLContext (GLFWwindow *windowHandle)

  *Constructor for the OpenGLContext class.*
- virtual void Init () override

  *Function that initializes the OpenGL context. Calls the gladLoadGLLoader function to load the OpenGL function pointers.*
- virtual void SwapBuffers () override

  *Function that swaps the buffers of the OpenGL context. Calls the glfwSwapBuffers function.*

#### Private Attributes

- GLFWwindow * m_WindowHandle

### 9.28.1 Detailed Description

The OpenGLContext class is an implementation of the GraphicsContext class for the OpenGL renderer.

The context stores the window handle of the application window. The context is initialized using the glfwMake↩ ContextCurrent function.

### 9.28.2 Constructor & Destructor Documentation

#### 9.28.2.1 OpenGLContext()

```
Fracture::OpenGLContext::OpenGLContext (
            GLFWwindow * windowHandle )
```

Constructor for the OpenGLContext class.

**Parameters**

| | | |
|---|---|---|
| in | *GLFWwindow*∗ | windowHandle: The window handle of the application window. |

**See also**

GLFWwindow

### 9.28.3 Member Function Documentation

#### 9.28.3.1 Init()

```
void Fracture::OpenGLContext::Init ( )  [override], [virtual]
```

Function that initializes the OpenGL context. Calls the gladLoadGLLoader function to load the OpenGL function pointers.

makes the provided window handle the current context using the glfwMakeContextCurrent function. Sets up the glad OpenGL function pointers.

**See also**

gladLoadGLLoader

Implements Fracture::GraphicsContext.

#### 9.28.3.2 SwapBuffers()

```
void Fracture::OpenGLContext::SwapBuffers ( )  [override], [virtual]
```

Function that swaps the buffers of the OpenGL context. Calls the glfwSwapBuffers function.

**See also**

glfwSwapBuffers

Implements Fracture::GraphicsContext.

### 9.28.4 Member Data Documentation

#### 9.28.4.1 m_WindowHandle

```
GLFWwindow* Fracture::OpenGLContext::m_WindowHandle  [private]
```

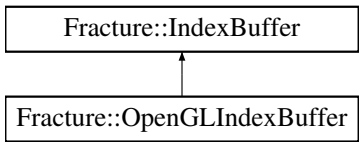
The documentation for this class was generated from the following files:

- Fracture/src/Platform/OpenGL/OpenGLContext.h
- Fracture/src/Platform/OpenGL/OpenGLContext.cpp

## 9.29 Fracture::OpenGLIndexBuffer Class Reference

```
#include <OpenGLBuffer.h>
```

Inheritance diagram for Fracture::OpenGLIndexBuffer:




#### Public Member Functions

- OpenGLIndexBuffer (uint32_t *indices, uint32_t count)
- ~OpenGLIndexBuffer ()
- virtual void SetData (const void *data, uint32_t size) override
- virtual void Bind () const override
- virtual void Unbind () const override
- virtual uint32_t GetCount () const override

### Public Member Functions inherited from Fracture::IndexBuffer

- virtual ~IndexBuffer ()=default

#### Private Attributes

- uint32_t m_RendererID
- uint32_t m_Count

#### Additional Inherited Members

### Static Public Member Functions inherited from Fracture::IndexBuffer

- static Ref< IndexBuffer > Create (uint32_t *indices, uint32_t size)

  *Function that creates an index buffer. This function will create an index buffer based on the platform that the application is running on.*

### 9.29.1 Constructor & Destructor Documentation

#### 9.29.1.1 OpenGLIndexBuffer()

```
Fracture::OpenGLIndexBuffer::OpenGLIndexBuffer (
            uint32_t * indices,
            uint32_t count )
```

#### 9.29.1.2 ~OpenGLIndexBuffer()

```
Fracture::OpenGLIndexBuffer::~OpenGLIndexBuffer ( )
```

### 9.29.2 Member Function Documentation

#### 9.29.2.1 Bind()

```
void Fracture::OpenGLIndexBuffer::Bind ( ) const  [override], [virtual]
```

Implements Fracture::IndexBuffer.

#### 9.29.2.2 GetCount()

```
virtual uint32_t Fracture::OpenGLIndexBuffer::GetCount ( ) const  [inline], [override], [virtual]
```

Implements Fracture::IndexBuffer.

#### 9.29.2.3 SetData()

```
void Fracture::OpenGLIndexBuffer::SetData (
            const void * data,
            uint32_t size )  [override], [virtual]
```

Implements Fracture::IndexBuffer.

#### 9.29.2.4 Unbind()

```
void Fracture::OpenGLIndexBuffer::Unbind ( ) const  [override], [virtual]
```

Implements Fracture::IndexBuffer.

### 9.29.3 Member Data Documentation

#### 9.29.3.1 m_Count

```
uint32_t Fracture::OpenGLIndexBuffer::m_Count  [private]
```

#### 9.29.3.2 m_RendererID

```
uint32_t Fracture::OpenGLIndexBuffer::m_RendererID  [private]
```

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

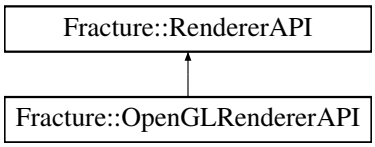
- Fracture/src/Platform/OpenGL/OpenGLBuffer.h
- Fracture/src/Platform/OpenGL/OpenGLBuffer.cpp

## 9.30 Fracture::OpenGLRendererAPI Class Reference

Implementation of the RendererAPI for OpenGL.

```
#include <OpenGLRendererAPI.h>
```

Inheritance diagram for Fracture::OpenGLRendererAPI:




### Public Member Functions

- OpenGLRendererAPI ()

  *Constructor for the OpenGLRendererAPI class.*
- ~OpenGLRendererAPI ()
- virtual void Init () override

  *Function that initializes the OpenGLRendererAPI.*
- virtual void SetClearColor (const glm::vec4 &color) override

  *Function that sets the clear color for OpenGL.*
- virtual void SetViewport (uint32_t x, uint32_t y, uint32_t width, uint32_t height) override

  *Function that sets the viewport for OpenGL.*
- virtual void Clear () override

  *Function that clears the screen.*
- virtual void DrawIndexed (uint32_t indexCount=0) override

  *Function that draws the currently bound vertex array with the currently bound index buffer with the provided index count.*
- virtual bool IsInitialized () const override

  *Checks if the OpenGLRendererAPI is initialized.*

### Private Attributes

- bool m_IsInitialized = false

**Additional Inherited Members**

## Public Types inherited from Fracture::RendererAPI

- enum class API { None = 0 , OpenGL = 1 }

  *The API enum class defines the different types of APIs that can be used by the renderer.*

## Static Public Member Functions inherited from Fracture::RendererAPI

- static API GetAPI ()

  *Function that returns the current API that is being used by the renderer.*

### 9.30.1 Detailed Description

Implementation of the RendererAPI for OpenGL.

### 9.30.2 Constructor & Destructor Documentation

#### 9.30.2.1 OpenGLRendererAPI()

```
Fracture::OpenGLRendererAPI::OpenGLRendererAPI ( )
```

Constructor for the OpenGLRendererAPI class.

Calls the Init() function

**See also**

RendererAPI

**Todo** : Add more states to be set up.

#### 9.30.2.2 ~OpenGLRendererAPI()

```
Fracture::OpenGLRendererAPI::~OpenGLRendererAPI ( )
```

### 9.30.3 Member Function Documentation

#### 9.30.3.1 Clear()

```
void Fracture::OpenGLRendererAPI::Clear ( )  [override], [virtual]
```

Function that clears the screen.

Calls the glClear function to clear the screen

Implements Fracture::RendererAPI.

#### 9.30.3.2 DrawIndexed()

```
void Fracture::OpenGLRendererAPI::DrawIndexed (
            uint32_t indexCount = 0 )  [override], [virtual]
```

Function that draws the currently bound vertex array with the currently bound index buffer with the provided index count.

Calls glDrawElements with the provided index count.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | indexCount: The number of indices to draw. |

Implements Fracture::RendererAPI.

#### 9.30.3.3 Init()

```
void Fracture::OpenGLRendererAPI::Init ( )  [override], [virtual]
```

Function that initializes the OpenGLRendererAPI.

Initializes all the OpenGL states that are required for the renderer to work. Currently only enables blending.

**See also**

RendererAPI

Implements Fracture::RendererAPI.

#### 9.30.3.4 IsInitialized()

```
virtual bool Fracture::OpenGLRendererAPI::IsInitialized ( ) const  [inline], [override], [virtual]
```

Checks if the OpenGLRendererAPI is initialized.

**Returns**

bool: True if the OpenGLRendererAPI is initialized.

Implements Fracture::RendererAPI.

#### 9.30.3.5 SetClearColor()

```
void Fracture::OpenGLRendererAPI::SetClearColor (
            const glm::vec4 & color )  [override], [virtual]
```

Function that sets the clear color for OpenGL.

Calls the glClearColor function to set the clear color for OpenGL.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec4& color: The color to set the clear color to. |

Implements Fracture::RendererAPI.

#### 9.30.3.6 SetViewport()

```
void Fracture::OpenGLRendererAPI::SetViewport (
            uint32_t x,
            uint32_t y,
            uint32_t width,
            uint32_t height )  [override], [virtual]
```

Function that sets the viewport for OpenGL.

Calls the glViewport function to set the viewport for OpenGL.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | x: The x coordinate of the viewport. |
| in | *uint32_t* | y: The y coordinate of the viewport. |
| in | *uint32_t* | width: The width of the viewport. |
| in | *uint32_t* | height: The height of the viewport. |

Implements Fracture::RendererAPI.

### 9.30.4 Member Data Documentation

#### 9.30.4.1 m_IsInitialized

```
bool Fracture::OpenGLRendererAPI::m_IsInitialized = false  [private]
```

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

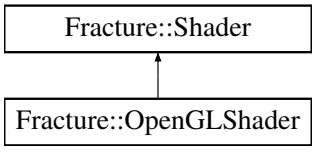
- Fracture/src/Platform/OpenGL/OpenGLRendererAPI.h
- Fracture/src/Platform/OpenGL/OpenGLRendererAPI.cpp

## 9.31 Fracture::OpenGLShader Class Reference

The OpenGLShader class is an implementation of the Shader class. It is used to create a shader program for the OpenGL renderer.

```
#include <OpenGLShader.h>
```

Inheritance diagram for Fracture::OpenGLShader:



### Public Member Functions

- OpenGLShader (const std::string &name, const std::string &vertex_source, const std::string fragment_source)

  *Constructor for the OpenGLShader class that takes in the name of the shader, the vertex source, and the fragment source.*
- OpenGLShader (const std::string &name, const std::string &shaderFilePath)

  *Constructor for the OpenGLShader class that takes in the name of the shader and the path to the shader file.*
- ~OpenGLShader ()

  *Destructor for the OpenGLShader class deletes the shader program.*
- virtual void Bind () const override

  *Function that binds to be used by the subsequent draw calls.*
- virtual void Unbind () const override

  *Function that unbinds the shader.*
- virtual void SetInt (const std::string &name, int value) override
- virtual void SetInt2 (const std::string &name, const glm::ivec2 &values) override
- virtual void SetInt3 (const std::string &name, const glm::ivec3 &values) override
- virtual void SetInt4 (const std::string &name, const glm::ivec4 &values) override
- virtual void SetFloat (const std::string &name, float value) override
- virtual void SetFloat2 (const std::string &name, const glm::vec2 &values) override
- virtual void SetFloat3 (const std::string &name, const glm::vec3 &values) override
- virtual void SetFloat4 (const std::string &name, const glm::vec4 &values) override
- virtual void SetMat3 (const std::string &name, const glm::mat3 &matrix) override
- virtual void SetMat4 (const std::string &name, const glm::mat4 &matrix) override
- virtual void SetBool (const std::string &name, bool value) override
- void UploadUniformInt (const std::string &name, int value)
- void UploadUniformInt2 (const std::string &name, const glm::ivec2 &values)
- void UploadUniformInt3 (const std::string &name, const glm::ivec3 &values)
- void UploadUniformInt4 (const std::string &name, const glm::ivec4 &values)
- void UploadUniformFloat (const std::string &name, float value)
- void UploadUniformFloat2 (const std::string &name, const glm::vec2 &values)
- void UploadUniformFloat3 (const std::string &name, const glm::vec3 &values)
- void UploadUniformFloat4 (const std::string &name, const glm::vec4 &values)
- void UploadUniformMat3 (const std::string &name, const glm::mat3 &matrix)
- void UploadUniformMat4 (const std::string &name, const glm::mat4 &matrix)
- void UploadUniformBool (const std::string &name, bool value)
- virtual const std::string & GetName () const override

  *Function to get the name of the shader. Must be implemented by the platform specific shader class.*
- virtual const uint32_t & GetHandle () const override

  *Function to get the handle ID of the shader. Must be implemented by the platform specific shader class.*

## Public Member Functions inherited from Fracture::Shader

- virtual ~Shader ()

### Private Member Functions

- int32_t GetUniformLocation (const std::string &name)

  *Function to get the location of a uniform from chache or from the shader program.*
- std::unordered_map< GLenum, std::string > PreProcess (const std::string &source)

  *Preprocess the shader source code to get the shader source code for each shader type.*
- void Compile (const std::unordered_map< GLenum, std::string > &shaderSources)

  *Compile the shader program given the shader source code for each shader type.*

#### Private Attributes

- uint32_t m_RendererID
- std::string m_Name

  *The handle to the shader program.*
- std::unordered_map< std::string, int32_t > m_UniformLocationCache

  *The name of the shader mostly used for debugging and identification.*

#### Additional Inherited Members

### Static Public Member Functions inherited from Fracture::Shader

- static Ref< Shader > Create (const std::string &name, const std::string &vertex_source, const std::string fragment_source)

  *Function that creates a shader from 2 strings containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*
- static Ref< Shader > Create (const std::string &name, const std::string &shaderFilePath)

  *Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*
- static Ref< Shader > Create (const std::string &shaderFilePath)

  *Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*

## 9.31.1 Detailed Description

The OpenGLShader class is an implementation of the Shader class. It is used to create a shader program for the OpenGL renderer.

## 9.31.2 Constructor & Destructor Documentation

### 9.31.2.1 OpenGLShader() [1/2]

```
Fracture::OpenGLShader::OpenGLShader (
            const std::string & name,
            const std::string & vertex_source,
            const std::string fragment_source )
```

Constructor for the OpenGLShader class that takes in the name of the shader, the vertex source, and the fragment source.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name The name of the shader |
| in | *const* | std::string& vertex_source The vertex source of the shader |
| in | *const* | std::string& fragment_source The fragment source of the shader |

**See also**

Shader

OpenGLShader Constructor with a vertex and fragment shader source

This function will create a shader program.

Arguments: vertex_source(const std::string&): The source code for the vertex shader fragment_source(const std::string&): The source code for the fragment shader

#### 9.31.2.2 OpenGLShader() [2/2]

```
Fracture::OpenGLShader::OpenGLShader (
            const std::string & name,
            const std::string & shaderFilePath )
```

Constructor for the OpenGLShader class that takes in the name of the shader and the path to the shader file.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name The name of the shader |
| in | *const* | std::string& shaderFilePath The path to the shader file |

OpenGLShader Constructor with a shader file path

This function will read the shader file and create a shader program. The vertex shader needs to be defined within a ifdef _TYPE_VERTEX_SHADER endif and the fragment shader needs to be defined within a ifdef _TYPE_FRAGMENT_SHADER endif

do not include the version 450 core in the shader file. That will be added by the preprocessor.

Arguments: ShaderFilePath(const std::string&): Path to the shader file

#### 9.31.2.3 ~OpenGLShader()

```
Fracture::OpenGLShader::~OpenGLShader ( )
```

Destructor for the OpenGLShader class deletes the shader program.

### 9.31.3 Member Function Documentation

#### 9.31.3.1 Bind()

```
void Fracture::OpenGLShader::Bind ( ) const  [override], [virtual]
```

Function that binds to be used by the subsequent draw calls.

Implements Fracture::Shader.

#### 9.31.3.2 Compile()

```
void Fracture::OpenGLShader::Compile (
            const std::unordered_map< GLenum, std::string > & shaderSources )  [private]
```

Compile the shader program given the shader source code for each shader type.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::unordered_map<GLenum, std::string>& shaderSources The shader source code for each shader type |

#### 9.31.3.3 GetHandle()

```
virtual const uint32_t & Fracture::OpenGLShader::GetHandle ( ) const  [inline], [override],
[virtual]
```

Function to get the handle ID of the shader. Must be implemented by the platform specific shader class.

**Returns**

const uint32_t&: The handle ID of the shader.

Implements Fracture::Shader.

#### 9.31.3.4 GetName()

```
virtual const std::string & Fracture::OpenGLShader::GetName ( ) const  [inline], [override],
[virtual]
```

Function to get the name of the shader. Must be implemented by the platform specific shader class.

**Returns**

const std::string&: The name of the shader.

Implements Fracture::Shader.

#### 9.31.3.5 GetUniformLocation()

```
int32_t Fracture::OpenGLShader::GetUniformLocation (
            const std::string & name )  [private]
```

Function to get the location of a uniform from chache or from the shader program.

The function checks if the uniform is in the cache. If it is, then it returns the location of the uniform from the cache. If it is not, then it gets the location of the uniform from the shader program and adds it to the cache.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name The name of the uniform |

**Returns**

int32_t The location of the uniform

#### 9.31.3.6 PreProcess()

```
std::unordered_map< GLenum, std::string > Fracture::OpenGLShader::PreProcess (
            const std::string & source )  [private]
```

Preprocess the shader source code to get the shader source code for each shader type.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& source The source code of the shader |

**Returns**

std::unordered_map<GLenum, std::string> The shader source code for each shader type

#### 9.31.3.7 SetBool()

```
void Fracture::OpenGLShader::SetBool (
            const std::string & name,
            bool value )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.8 SetFloat()

```
void Fracture::OpenGLShader::SetFloat (
            const std::string & name,
            float value )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.9 SetFloat2()

```
void Fracture::OpenGLShader::SetFloat2 (
            const std::string & name,
            const glm::vec2 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.10 SetFloat3()

```
void Fracture::OpenGLShader::SetFloat3 (
            const std::string & name,
            const glm::vec3 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.11 SetFloat4()

```
void Fracture::OpenGLShader::SetFloat4 (
            const std::string & name,
            const glm::vec4 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.12 SetInt()

```
void Fracture::OpenGLShader::SetInt (
            const std::string & name,
            int value )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.13 SetInt2()

```
void Fracture::OpenGLShader::SetInt2 (
            const std::string & name,
            const glm::ivec2 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.14 SetInt3()

```
void Fracture::OpenGLShader::SetInt3 (
            const std::string & name,
            const glm::ivec3 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.15 SetInt4()

```
void Fracture::OpenGLShader::SetInt4 (
            const std::string & name,
            const glm::ivec4 & values )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.16 SetMat3()

```
void Fracture::OpenGLShader::SetMat3 (
            const std::string & name,
            const glm::mat3 & matrix )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.17 SetMat4()

```
void Fracture::OpenGLShader::SetMat4 (
            const std::string & name,
            const glm::mat4 & matrix )  [override], [virtual]
```

Implements Fracture::Shader.

#### 9.31.3.18 Unbind()

```
void Fracture::OpenGLShader::Unbind ( ) const  [override], [virtual]
```

Function that unbinds the shader.

Implements Fracture::Shader.

#### 9.31.3.19 UploadUniformBool()

```
void Fracture::OpenGLShader::UploadUniformBool (
            const std::string & name,
            bool value )
```

#### 9.31.3.20 UploadUniformFloat()

```
void Fracture::OpenGLShader::UploadUniformFloat (
            const std::string & name,
            float value )
```

#### 9.31.3.21 UploadUniformFloat2()

```
void Fracture::OpenGLShader::UploadUniformFloat2 (
            const std::string & name,
            const glm::vec2 & values )
```

#### 9.31.3.22 UploadUniformFloat3()

```
void Fracture::OpenGLShader::UploadUniformFloat3 (
            const std::string & name,
            const glm::vec3 & values )
```

#### 9.31.3.23 UploadUniformFloat4()

```
void Fracture::OpenGLShader::UploadUniformFloat4 (
            const std::string & name,
            const glm::vec4 & values )
```

#### 9.31.3.24 UploadUniformInt()

```
void Fracture::OpenGLShader::UploadUniformInt (
            const std::string & name,
            int value )
```

#### 9.31.3.25 UploadUniformInt2()

```
void Fracture::OpenGLShader::UploadUniformInt2 (
            const std::string & name,
            const glm::ivec2 & values )
```

#### 9.31.3.26 UploadUniformInt3()

```
void Fracture::OpenGLShader::UploadUniformInt3 (
            const std::string & name,
            const glm::ivec3 & values )
```

#### 9.31.3.27 UploadUniformInt4()

```
void Fracture::OpenGLShader::UploadUniformInt4 (
            const std::string & name,
            const glm::ivec4 & values )
```

#### 9.31.3.28 UploadUniformMat3()

```
void Fracture::OpenGLShader::UploadUniformMat3 (
            const std::string & name,
            const glm::mat3 & matrix )
```

#### 9.31.3.29 UploadUniformMat4()

```
void Fracture::OpenGLShader::UploadUniformMat4 (
            const std::string & name,
            const glm::mat4 & matrix )
```

### 9.31.4 Member Data Documentation

#### 9.31.4.1 m_Name

```
std::string Fracture::OpenGLShader::m_Name  [private]
```

The handle to the shader program.

#### 9.31.4.2 m_RendererID

```
uint32_t Fracture::OpenGLShader::m_RendererID  [private]
```

#### 9.31.4.3 m_UniformLocationCache

```
std::unordered_map<std::string, int32_t> Fracture::OpenGLShader::m_UniformLocationCache  [private]
```

The name of the shader mostly used for debugging and identification.

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

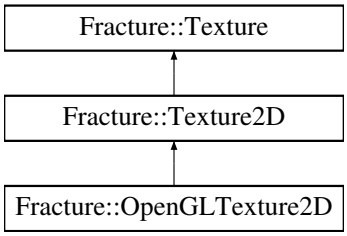
- Fracture/src/Platform/OpenGL/OpenGLShader.h
- Fracture/src/Platform/OpenGL/OpenGLShader.cpp

## 9.32 Fracture::OpenGLTexture2D Class Reference

OpenGL implementation of the Texture2D class.

```
#include <OpenGLTexture.h>
```

Inheritance diagram for Fracture::OpenGLTexture2D:




### Public Member Functions

- OpenGLTexture2D (const std::string &path)

  *Constructor for the OpenGLTexture2D class that takes in a path to the texture file.*
- OpenGLTexture2D (uint32_t width, uint32_t height, glm::vec4 color)

  *Constructor for the OpenGLTexture2D class that takes in a width, height, and color.*
- virtual ~OpenGLTexture2D ()
- virtual uint32_t GetWidth () const override

  *Function that returns the width of the texture.*
- virtual uint32_t GetHeight () const override

  *Function that returns the height of the texture.*
- virtual uint32_t GetHandle () const override

  *Function that returns the handle of the texture.*
- virtual void Bind (uint32_t slot=0) const override

  *Sets the texture data to a specific texture slot.*

## Public Member Functions inherited from Fracture::Texture

- virtual ~Texture ()=default

### Private Attributes

- std::string m_Path
- uint32_t m_Width

  *The path to the texture file. Stored for debugging purposes.*
- uint32_t m_Height
- uint32_t m_RendererID

  *The width and height of the texture.*

### Additional Inherited Members

## Static Public Member Functions inherited from Fracture::Texture2D

- static Ref< Texture2D > Create (uint32_t width, uint32_t height, glm::vec4 color)

  *Function that creates a 2D texture from a given colour and a width and height. The texture will be 4 channel RGBA.*
- static Ref< Texture2D > Create (const std::string &path)

  *Function that creates a 2D texture from a given path to an image file.*

## 9.32.1 Detailed Description

OpenGL implementation of the Texture2D class.

## 9.32.2 Constructor & Destructor Documentation

### 9.32.2.1 OpenGLTexture2D() [1/2]

```
Fracture::OpenGLTexture2D::OpenGLTexture2D (
            const std::string & path )
```

Constructor for the OpenGLTexture2D class that takes in a path to the texture file.

The constructor will create a texture from the image in file path and store the texture on the GPU

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& path: The path to the texture file. |

### 9.32.2.2 OpenGLTexture2D() [2/2]

```
Fracture::OpenGLTexture2D::OpenGLTexture2D (
            uint32_t width,
```

```
            uint32_t height,
            glm::vec4 color )
```

Constructor for the OpenGLTexture2D class that takes in a width, height, and color.

Creates an image with the specified width, height, and color and stores it on the GPU as a texture.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | width: The width of the texture. |
| in | *uint32_t* | height: The height of the texture. |
| in | *glm::vec4* | color: The color of the texture. |

#### 9.32.2.3 ~OpenGLTexture2D()

```
Fracture::OpenGLTexture2D::~OpenGLTexture2D ( )  [virtual]
```

### 9.32.3 Member Function Documentation

#### 9.32.3.1 Bind()

```
void Fracture::OpenGLTexture2D::Bind (
            uint32_t slot = 0 ) const  [override], [virtual]
```

Sets the texture data to a specific texture slot.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | slot: The texture slot to bind the texture to. |

Implements Fracture::Texture.

#### 9.32.3.2 GetHandle()

```
virtual uint32_t Fracture::OpenGLTexture2D::GetHandle ( ) const  [inline], [override], [virtual]
```

Function that returns the handle of the texture.

**Returns**

uint32_t The handle of the texture.

Implements Fracture::Texture.

#### 9.32.3.3 GetHeight()

```
virtual uint32_t Fracture::OpenGLTexture2D::GetHeight ( ) const  [inline], [override], [virtual]
```

Function that returns the height of the texture.

**Returns**

uint32_t The height of the texture.

Implements Fracture::Texture.

#### 9.32.3.4 GetWidth()

```
virtual uint32_t Fracture::OpenGLTexture2D::GetWidth ( ) const  [inline], [override], [virtual]
```

Function that returns the width of the texture.

**Returns**

uint32_t The width of the texture.

Implements Fracture::Texture.

### 9.32.4 Member Data Documentation

#### 9.32.4.1 m_Height

```
uint32_t Fracture::OpenGLTexture2D::m_Height  [private]
```

#### 9.32.4.2 m_Path

```
std::string Fracture::OpenGLTexture2D::m_Path  [private]
```

#### 9.32.4.3 m_RendererID

```
uint32_t Fracture::OpenGLTexture2D::m_RendererID  [private]
```

The width and height of the texture.

#### 9.32.4.4 m_Width

```
uint32_t Fracture::OpenGLTexture2D::m_Width  [private]
```

The path to the texture file. Stored for debugging purposes.

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

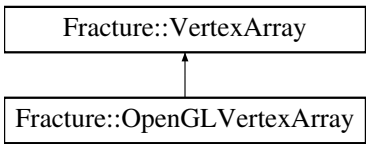
- Fracture/src/Platform/OpenGL/OpenGLTexture.h
- Fracture/src/Platform/OpenGL/OpenGLTexture.cpp

# 9.33 Fracture::OpenGLVertexArray Class Reference

OpenGLVertexArray class that implements the VertexArray class for OpenGL.

`#include <OpenGLVertexArray.h>`

Inheritance diagram for Fracture::OpenGLVertexArray:




### Public Member Functions

- OpenGLVertexArray ()

  *Create a VertexArray object and store the ID in m_RendererID. Binds the VertexArray object.*
- ~OpenGLVertexArray ()

  *Destructor for the OpenGLVertexArray class. Deletes the VertexArray object.*
- virtual void AddVertexBuffer (const Ref< VertexBuffer > &vertexBuffer) override

  *Add a VertexBuffer to the VertexArray object.*
- virtual void SetIndexBuffer (const Ref< IndexBuffer > &indexBuffer) override

  *Set the IndexBuffer of the VertexArray object.*
- virtual const Ref< IndexBuffer > & GetIndexBuffer () const override

  *Get the IndexBuffer of the VertexArray object.*
- virtual const std::vector< Ref< VertexBuffer > > & GetVertexBuffers () const override

  *Gets all the vertex buffers of the vertex array.*
- virtual void Bind () const override

  *Binds the vertex array.*
- virtual void Unbind () const override

  *Unbinds the vertex array.*

## Public Member Functions inherited from Fracture::VertexArray

- virtual ~VertexArray ()

### Private Attributes

- uint32_t m_RendererID
- std::vector< Ref< VertexBuffer > > m_VertexBuffers

  *The ID of the vertex array.*
- Ref< IndexBuffer > m_IndexBuffer

  *A vector of vertex buffers.*
- uint32_t m_VertexBufferIndex = 0

  *The index buffer of the vertex array.*

**Additional Inherited Members**

## Static Public Member Functions inherited from Fracture::VertexArray

- static Ref< VertexArray > Create ()

  *Function that creates a VertexArray. This function will create a VertexArray based on the current active renderer.*

### 9.33.1 Detailed Description

OpenGLVertexArray class that implements the VertexArray class for OpenGL.

**See also**

VertexArray

### 9.33.2 Constructor & Destructor Documentation

#### 9.33.2.1 OpenGLVertexArray()

```
Fracture::OpenGLVertexArray::OpenGLVertexArray ( )
```

Create a VertexArray object and store the ID in m_RendererID. Binds the VertexArray object.

#### 9.33.2.2 ~OpenGLVertexArray()

```
Fracture::OpenGLVertexArray::~OpenGLVertexArray ( )
```

Destructor for the OpenGLVertexArray class. Deletes the VertexArray object.

### 9.33.3 Member Function Documentation

#### 9.33.3.1 AddVertexBuffer()

```
void Fracture::OpenGLVertexArray::AddVertexBuffer (
            const Ref< VertexBuffer > & vertexBuffer )  [override], [virtual]
```

Add a VertexBuffer to the VertexArray object.

Binds the vertex buffer to the vertex array state and stores the vertex buffer in the m_VertexBuffers vector. It sets the vertexAttributePointers for the vertex buffer using the vertex buffer layout.

**See also**

VertexBuffer

**Parameters**

| | | |
|---|---|---|
| in | *const* | Ref<VertexBuffer>& vertexBuffer: The vertex buffer to be added to the vertex array. |

Implements Fracture::VertexArray.

#### 9.33.3.2 Bind()

```
void Fracture::OpenGLVertexArray::Bind ( ) const  [override], [virtual]
```

Binds the vertex array.

Implements Fracture::VertexArray.

#### 9.33.3.3 GetIndexBuffer()

```
virtual const Ref< IndexBuffer > & Fracture::OpenGLVertexArray::GetIndexBuffer ( ) const
[inline], [override], [virtual]
```

Get the IndexBuffer of the VertexArray object.

**Returns**

A reference to the index buffer of the vertex array.

Implements Fracture::VertexArray.

#### 9.33.3.4 GetVertexBuffers()

```
virtual const std::vector< Ref< VertexBuffer > > & Fracture::OpenGLVertexArray::GetVertex↩
Buffers ( ) const  [inline], [override], [virtual]
```

Gets all the vertex buffers of the vertex array.

**Returns**

A vector of references to the vertex buffers of the vertex array.

Implements Fracture::VertexArray.

#### 9.33.3.5 SetIndexBuffer()

```
void Fracture::OpenGLVertexArray::SetIndexBuffer (
            const Ref< IndexBuffer > & indexBuffer )  [override], [virtual]
```

Set the IndexBuffer of the VertexArray object.

Binds the index buffer to the vertex array state and stores the index buffer in the m_IndexBuffer variable.

Implements Fracture::VertexArray.

#### 9.33.3.6 Unbind()

```
void Fracture::OpenGLVertexArray::Unbind ( ) const  [override], [virtual]
```

Unbinds the vertex array.

Implements Fracture::VertexArray.

### 9.33.4 Member Data Documentation

#### 9.33.4.1 m_IndexBuffer

```
Ref<IndexBuffer> Fracture::OpenGLVertexArray::m_IndexBuffer  [private]
```

A vector of vertex buffers.

#### 9.33.4.2 m_RendererID

```
uint32_t Fracture::OpenGLVertexArray::m_RendererID  [private]
```

#### 9.33.4.3 m_VertexBufferIndex

```
uint32_t Fracture::OpenGLVertexArray::m_VertexBufferIndex = 0  [private]
```

The index buffer of the vertex array.

#### 9.33.4.4 m_VertexBuffers

```
std::vector<Ref<VertexBuffer> > Fracture::OpenGLVertexArray::m_VertexBuffers  [private]
```

The ID of the vertex array.

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

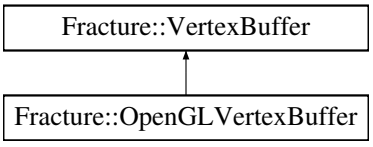
- Fracture/src/Platform/OpenGL/OpenGLVertexArray.h
- Fracture/src/Platform/OpenGL/OpenGLVertexArray.cpp

## 9.34 Fracture::OpenGLVertexBuffer Class Reference

The OpenGLVertexBuffer class is an implementation of the VertexBuffer class for OpenGL.

```
#include <OpenGLBuffer.h>
```

Inheritance diagram for Fracture::OpenGLVertexBuffer:



#### Public Member Functions

- OpenGLVertexBuffer (float *vertices, uint32_t size)

  *Constructor for the OpenGLVertexBuffer class.*
- ~OpenGLVertexBuffer ()

  *Destructor for the OpenGLVertexBuffer class. Deletes the buffers.*
- virtual void SetData (const void *data, uint32_t size) override

  *Function that sets the data of the vertex buffer.*
- virtual void SetLayout (const BufferLayout &layout) override

  *Function that sets the layout of the vertex buffer. This is needed to be used to draw.*
- virtual const BufferLayout & GetLayout () const override

  *Function that returns the layout of the vertex buffer.*
- virtual void Bind () const override

  *Function that binds the vertex buffer.*
- virtual void Unbind () const override

  *Function that unbinds the vertex buffer.*

### Public Member Functions inherited from Fracture::VertexBuffer

- virtual ~VertexBuffer ()=default

#### Private Attributes

- uint32_t m_RendererID
- BufferLayout m_Layout

  *The handle to the vertex buffer.*

#### Additional Inherited Members

### Static Public Member Functions inherited from Fracture::VertexBuffer

- static Ref< VertexBuffer > Create (float *vertices, uint32_t size)

  *Function that creates a vertex buffer. This function will create a vertex buffer based on the platform that the application is running on.*

## 9.34.1 Detailed Description

The OpenGLVertexBuffer class is an implementation of the VertexBuffer class for OpenGL.

**See also**

VertexBuffer

## 9.34.2 Constructor & Destructor Documentation

### 9.34.2.1 OpenGLVertexBuffer()

```
Fracture::OpenGLVertexBuffer::OpenGLVertexBuffer (
            float * vertices,
            uint32_t size )
```

Constructor for the OpenGLVertexBuffer class.

Creates the buffers, binds it and then sets the data.

**Parameters**

| | | |
|---|---|---|
| in | *float*∗ | vertices: The vertices of the vertex buffer |
| in | *uint32_t* | size: The size of the vertex buffer |

#### 9.34.2.2 ∼OpenGLVertexBuffer()

```
Fracture::OpenGLVertexBuffer::~OpenGLVertexBuffer ( )
```

Destructor for the OpenGLVertexBuffer class. Deletes the buffers.

### 9.34.3 Member Function Documentation

#### 9.34.3.1 Bind()

```
void Fracture::OpenGLVertexBuffer::Bind ( ) const  [override], [virtual]
```

Function that binds the vertex buffer.

Implements Fracture::VertexBuffer.

#### 9.34.3.2 GetLayout()

```
virtual const BufferLayout & Fracture::OpenGLVertexBuffer::GetLayout ( ) const  [inline],
[override], [virtual]
```

Function that returns the layout of the vertex buffer.

**Returns**

BufferLayout& The layout of the vertex buffer

Implements Fracture::VertexBuffer.

#### 9.34.3.3 SetData()

```
void Fracture::OpenGLVertexBuffer::SetData (
            const void * data,
            uint32_t size )  [override], [virtual]
```

Function that sets the data of the vertex buffer.

Function assumes the vertex buffer is already bound.

**Todo** : Currently the draw call is of type OPENGL_STATIC_DRAW. This needs to be changed to be customizable.

**Parameters**

| | | |
|---|---|---|
| in | *void** | data: The data to be set in the vertex buffer |
| in | *uint32_t* | size: The size of the data to be set in the vertex buffer |

Implements Fracture::VertexBuffer.

#### 9.34.3.4 SetLayout()

```
virtual void Fracture::OpenGLVertexBuffer::SetLayout (
            const BufferLayout & layout )  [inline], [override], [virtual]
```

Function that sets the layout of the vertex buffer. This is needed to be used to draw.

**Parameters**

| | | |
|---|---|---|
| in | *const* | BufferLayout& layout: The layout of the vertex buffer |

**See also**

BufferLayout

Implements Fracture::VertexBuffer.

#### 9.34.3.5 Unbind()

```
void Fracture::OpenGLVertexBuffer::Unbind ( ) const  [override], [virtual]
```

Function that unbinds the vertex buffer.

Implements Fracture::VertexBuffer.

### 9.34.4 Member Data Documentation

#### 9.34.4.1 m_Layout

```
BufferLayout Fracture::OpenGLVertexBuffer::m_Layout  [private]
```

The handle to the vertex buffer.

#### 9.34.4.2 m_RendererID

```
uint32_t Fracture::OpenGLVertexBuffer::m_RendererID  [private]
```

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

- Fracture/src/Platform/OpenGL/OpenGLBuffer.h
- Fracture/src/Platform/OpenGL/OpenGLBuffer.cpp

## 9.35 Fracture::OrthographicCamera Class Reference

`#include <OrthographicCamera.h>`

### Public Member Functions

- OrthographicCamera (float left, float right, float bottom, float top)

  *Constructor for the OrthographicCamera class that takes in the left, right, bottom, and top values of the camera frustum.*
- OrthographicCamera (float left, float right, float bottom, float top, float nearval, float farval)

  *Constructor for the OrthographicCamera class that takes in the left, right, bottom, top, near, and far values of the camera frustum.*
- ~OrthographicCamera ()

  *Destructor for the OrthographicCamera class.*
- void SetProjection (float left, float right, float bottom, float top, float nearval=-1, float farval=1)

  *Function that sets the projection matrix of the camera with the left, right, bottom, and top values of the camera frustum.*
- const glm::mat4 & GetProjectionMatrix () const

  *Getter for the projection matrix of the camera.*
- const glm::mat4 & GetViewMatrix ()

  *Getter for the view matrix of the camera.*
- const glm::mat4 & GetViewProjectionMatrix ()

  *Getter for the view projection matrix of the camera.*
- void SetProjectionMatrix (const glm::mat4 &projection)

  *Setter for the projection matrix of the camera. Recalculates the view projection matrix.*
- void SetViewMatrix (const glm::mat4 &view)

  *Setter for the view matrix of the camera. Recalculates the view projection matrix.*

### Private Attributes

- glm::mat4 m_ProjectionMatrix
- glm::mat4 m_ViewMatrix

  *4x4 projection matrix of the camera*
- glm::mat4 m_ViewProjectionMatrix

  *4x4 view matrix of the camera*

### 9.35.1 Constructor & Destructor Documentation

#### 9.35.1.1 OrthographicCamera() [1/2]

```
Fracture::OrthographicCamera::OrthographicCamera (
            float left,
            float right,
            float bottom,
            float top )
```

Constructor for the OrthographicCamera class that takes in the left, right, bottom, and top values of the camera frustum.

@detials The near and far values are set to -1 and 1 respectively. The values set by the user are used to create the projection matrix. The view matrix is set to the identity matrix. The 4 values also determine the units of the world space. For example, if the left and right values are set to -10 and 10 respectively, then the world space will be from -10 to 10 in the x direction.

**Parameters**

| | | |
|---|---|---|
| in | *float* | left The left value of the camera frustum |
| in | *float* | right The right value of the camera frustum |
| in | *float* | bottom The bottom value of the camera frustum |
| in | *float* | top The top value of the camera frustum |

#### 9.35.1.2 OrthographicCamera() [2/2]

```
Fracture::OrthographicCamera::OrthographicCamera (
            float left,
            float right,
            float bottom,
            float top,
            float nearval,
            float farval )
```

Constructor for the OrthographicCamera class that takes in the left, right, bottom, top, near, and far values of the camera frustum.

@detials The values set by the user are used to create the projection matrix. The view matrix is set to the identity matrix. The 6 values also determine the units of the world space. For example, if the left and right values are set to -10 and 10 respectively, then the world space will be from -10 to 10 in the x direction.

**Parameters**

| | | |
|---|---|---|
| in | *float* | left The left value of the camera frustum |
| in | *float* | right The right value of the camera frustum |
| in | *float* | bottom The bottom value of the camera frustum |
| in | *float* | top The top value of the camera frustum |
| in | *float* | nearval The near value of the camera frustum |
| in | *float* | farval The far value of the camera frustum |

#### 9.35.1.3 ~OrthographicCamera()

```
Fracture::OrthographicCamera::~OrthographicCamera ( )
```

Destructor for the OrthographicCamera class.

### 9.35.2 Member Function Documentation

#### 9.35.2.1 GetProjectionMatrix()

```
const glm::mat4 & Fracture::OrthographicCamera::GetProjectionMatrix ( ) const  [inline]
```

Getter for the projection matrix of the camera.

**Returns**

const glm::mat4& The projection matrix of the camera.

#### 9.35.2.2 GetViewMatrix()

```
const glm::mat4 & Fracture::OrthographicCamera::GetViewMatrix ( )  [inline]
```

Getter for the view matrix of the camera.

**Returns**

const glm::mat4& The view matrix of the camera.

#### 9.35.2.3 GetViewProjectionMatrix()

```
const glm::mat4 & Fracture::OrthographicCamera::GetViewProjectionMatrix ( )  [inline]
```

Getter for the view projection matrix of the camera.

**Returns**

const glm::mat4& The view projection matrix of the camera.

#### 9.35.2.4 SetProjection()

```
void Fracture::OrthographicCamera::SetProjection (
            float left,
            float right,
            float bottom,
            float top,
            float nearval = -1,
            float farval = 1 )
```

Function that sets the projection matrix of the camera with the left, right, bottom, and top values of the camera frustum.

@detials The near and far values are set to -1 and 1 respectively. The values set by the user are used to create the projection matrix.

#### 9.35.2.5 SetProjectionMatrix()

```
void Fracture::OrthographicCamera::SetProjectionMatrix (
            const glm::mat4 & projection )  [inline]
```

Setter for the projection matrix of the camera. Recalculates the view projection matrix.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::mat4& projection The projection matrix of the camera. |

#### 9.35.2.6 SetViewMatrix()

```
void Fracture::OrthographicCamera::SetViewMatrix (
            const glm::mat4 & view )  [inline]
```

Setter for the view matrix of the camera. Recalculates the view projection matrix.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::mat4& view The view matrix of the camera. |

### 9.35.3 Member Data Documentation

#### 9.35.3.1 m_ProjectionMatrix

```
glm::mat4 Fracture::OrthographicCamera::m_ProjectionMatrix  [private]
```

#### 9.35.3.2 m_ViewMatrix

```
glm::mat4 Fracture::OrthographicCamera::m_ViewMatrix  [private]
```

4x4 projection matrix of the camera

#### 9.35.3.3 m_ViewProjectionMatrix

```
glm::mat4 Fracture::OrthographicCamera::m_ViewProjectionMatrix  [private]
```

4x4 view matrix of the camera

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

- Fracture/src/Fracture/Renderer/OrthographicCamera.h
- Fracture/src/Fracture/Renderer/OrthographicCamera.cpp

## 9.36 Fracture::OrthographicCameraController Class Reference

The OrthographicCameraController class is used to control the orthographic camera.

```
#include <OrthographicCameraController.h>
```

### Public Member Functions

- OrthographicCameraController (float aspectRatio, float enableRotation=false)

  *Constructor for the OrthographicCameraController class that takes in the aspect ratio of the window and whether or not the camera should be able to rotate.*
- void OnUpdate (Utils::Timestep ts)

  *the Update function that is called every frame by layer that owns the OrthographicCameraController.*
- void OnEvent (Event &e)

  *the OnEvent function that is called by the layer that owns the OrthographicCameraController when an event is triggered.*
- OrthographicCamera & GetCamera ()

  *Getter for the OrthographicCamera.*
- const OrthographicCamera & GetCamera () const

  *Getter for the OrthographicCamera to be used by a const OrthographicCameraController.*
- float GetAspectRatio () const

  *Getter for the aspect ratio of the window.*
- const glm::vec3 & GetPosition ()

  *Getter for the OrthographicCamera's position.*
- void SetPosition (const glm::vec3 &position)

  *Setter for the OrthographicCamera's position. Sets the position of the OrthographicCamera to the position passed in.*
- void Translate (const glm::vec3 &translation)

  *Translates the OrthographicCamera's position by the translation passed in.*
- const float & GetRotation ()

  *Getter for the OrthographicCamera's rotation.*
- void SetRotation (float rotation)

  *Setter for the OrthographicCamera's rotation. Sets the rotation of the OrthographicCamera to the rotation passed in if rotation is enabled. Only Z axis roation is supported due to this camera currently being 2D.*
- void Rotate (float rotation)

  *Rotates the OrthographicCamera's rotation by the rotation passed in if rotation is enabled. Only Z axis roation is supported due to this camera currently being 2D.*
- float & GetZoomLevel ()

  *Getter for the OrthographicCamera's zoom level.*
- void Zoom (float zoom)

  *Setter for the OrthographicCamera's zoom level. Sets the zoom level of the OrthographicCamera to the zoom level passed in.*
- void SetZoom (float zoom)

  *Setter for the OrthographicCamera's zoom level. Sets the zoom level of the OrthographicCamera to the zoom level passed in.*
- const TransformComponent & GetCameraTransform ()

  *Getter for the OrthographicCamera's transform component. Returns a constant reference to the transform component of the OrthographicCamera.*
- void SetCameraTransform (const TransformComponent &transform)

  *Setter for the OrthographicCamera's transform component. Sets the transform component of the OrthographicCamera to the transform component passed in. Sets the isChanged boolean to true.*
- void SetCameraZoomSpeed (float speed)

  *Setter for the camera zoom speed.*
- void ToggleRotation (bool enable)

  *Toggle to enable or disable camera rotation.*
- void SetMaxZoom (float zoom)

  *Setter for the max and min zoom levels.*
- void SetMinZoom (float zoom)

  *Setter for the max and min zoom levels.*
- float & GetCameraZoomSpeed ()

*Getter for the camera zoom speed. Returns a reference to the camera zoom speed so that it can be changed.*

- bool & GetRotationEnabled ()

  *getter for the current current ability to zoom. Returns a reference so that it can be changed.*
- float & GetMaxZoom ()

  *Getter for the max zoom level. Returns a reference so that they can be changed.*
- float & GetMinZoom ()

  *Getter for the min zoom level. Returns a reference so that they can be changed.*

### Private Member Functions

- bool OnMouseScrolledEvent (MouseScrolledEvent &e)

  *Function that is called when mouse is scrolled this modifies the zoom level by zooming in and out.*
- bool OnWindowResizedEvent (WindowResizeEvent &e)

  *Function that is called when the window is resized. This function updates the aspect ratio of the window and updates the projection matrix.*
- bool OnMouseButtonDownEvent (MouseButtonPressedEvent &e)

  *Function that is called when a mouse button is pressed.*
- bool OnMouseButtonUpEvent (MouseButtonReleasedEvent &e)

  *Function that is called when a mouse button is released.*

### Private Attributes

- Utils::Timestep m_LastFrameTime
- float m_AspectRatio

  *The delta time stored from the last frame to be used to smooth out the camera movement.*
- float m_ZoomLevel = 1.0f

  *The current aspect ratio of the camera frustum.*
- bool m_EnableRotation

  *The current zoom level of the camera.*
- TransformComponent m_CameraTransform

  *Whether or not the camera can rotate.*
- OrthographicCamera m_Camera

  *The transform component of the camera.*
- glm::vec2 m_InitialMousePosition = { 0.0f, 0.0f }

  *The orthographic camera.*
- glm::vec3 m_InitialCameraPosition = { 0.0f, 0.0f, 0.0f }

  *The initial mouse position when the middle mouse button is pressed.*
- float m_MiddleMouseScale = 0.005f

  *The initial camera position when the middle mouse button is pressed.*
- float m_cameraTranslationSpeed = 1.0f

  *The scale of the middle mouse movement.*
- float m_cameraRotationSpeed = 1.0f

  *The speed of the camera translation.*
- float m_cameraZoomSpeed = 40.0f

  *The speed of the camera rotation.*
- float m_MaxZoom = 100.0f

  *The speed of the camera zoom.*
- float m_MinZoom = 0.25f

  *The max zoom level of the camera.*
- bool isChanged = true

  *The min zoom level of the camera.*
- bool m_canMoveMiddleMouse = false

  *Boolean to check if the camera has changed.*

### 9.36.1 Detailed Description

The OrthographicCameraController class is used to control the orthographic camera.

**See also**

OrthographicCamera

### 9.36.2 Constructor & Destructor Documentation

#### 9.36.2.1 OrthographicCameraController()

```
Fracture::OrthographicCameraController::OrthographicCameraController (
            float aspectRatio,
            float enableRotation = false )
```

Constructor for the OrthographicCameraController class that takes in the aspect ratio of the window and whether or not the camera should be able to rotate.

**Parameters**

| | | |
|---|---|---|
| in | *float* | aspectRatio The aspect ratio of the window. |
| in | *float* | enableRotation Whether or not the camera should be able to rotate. |

### 9.36.3 Member Function Documentation

#### 9.36.3.1 GetAspectRatio()

```
float Fracture::OrthographicCameraController::GetAspectRatio ( ) const  [inline]
```

Getter for the aspect ratio of the window.

**Returns**

float The aspect ratio of the window.

#### 9.36.3.2 GetCamera() [1/2]

```
OrthographicCamera & Fracture::OrthographicCameraController::GetCamera ( )  [inline]
```

Getter for the OrthographicCamera.

**Returns**

OrthographicCamera& The reference to OrthographicCamera.

#### 9.36.3.3 GetCamera() [2/2]

```
const OrthographicCamera & Fracture::OrthographicCameraController::GetCamera ( ) const  [inline]
```

Getter for the OrthographicCamera to be used by a const OrthographicCameraController.

**Returns**

const OrthographicCamera& The const reference to OrthographicCamera.

#### 9.36.3.4 GetCameraTransform()

```
const TransformComponent & Fracture::OrthographicCameraController::GetCameraTransform ( )
[inline]
```

Getter for the OrthographicCamera's transform component. Returns a constant reference to the transform component of the OrthographicCamera.

**Returns**

cosnt TransformComponent&: The transform component of the OrthographicCamera.

#### 9.36.3.5 GetCameraZoomSpeed()

```
float & Fracture::OrthographicCameraController::GetCameraZoomSpeed ( )  [inline]
```

Getter for the camera zoom speed. Returns a reference to the camera zoom speed so that it can be changed.

**Returns**

float& The camera zoom speed.

#### 9.36.3.6 GetMaxZoom()

```
float & Fracture::OrthographicCameraController::GetMaxZoom ( )  [inline]
```

Getter for the max zoom level. Returns a reference so that they can be changed.

**Returns**

float& The max zoom level.

#### 9.36.3.7 GetMinZoom()

```
float & Fracture::OrthographicCameraController::GetMinZoom ( )  [inline]
```

Getter for the min zoom level. Returns a reference so that they can be changed.

**Returns**

float& The min zoom level.

#### 9.36.3.8 GetPosition()

```
const glm::vec3 & Fracture::OrthographicCameraController::GetPosition ( )  [inline]
```

Getter for the OrthographicCamera's position.

**Returns**

const glm::vec3& The position of the OrthographicCamera.

#### 9.36.3.9 GetRotation()

```
const float & Fracture::OrthographicCameraController::GetRotation ( )  [inline]
```

Getter for the OrthographicCamera's rotation.

**Returns**

const float& The rotation of the OrthographicCamera returned as a const reference

#### 9.36.3.10 GetRotationEnabled()

```
bool & Fracture::OrthographicCameraController::GetRotationEnabled ( )  [inline]
```

getter for the current current ability to zoom. Returns a reference so that it can be changed.

**Returns**

bool& The current ability to zoom.

#### 9.36.3.11 GetZoomLevel()

```
float & Fracture::OrthographicCameraController::GetZoomLevel ( )  [inline]
```

Getter for the OrthographicCamera's zoom level.

**Returns**

float&: Reference to the zoom level of the OrthographicCamera.

#### 9.36.3.12 OnEvent()

```
void Fracture::OrthographicCameraController::OnEvent (
            Event & e )
```

the OnEvent function that is called by the layer that owns the OrthographicCameraController when an event is triggered.

**Parameters**

| | | |
|---|---|---|
| in | *Event&* | e The event that is triggered. |

#### 9.36.3.13 OnMouseButtonDownEvent()

```
bool Fracture::OrthographicCameraController::OnMouseButtonDownEvent (
            MouseButtonPressedEvent & e )  [private]
```

Function that is called when a mouse button is pressed.

The function checks if the middle mouse button is pressed and if it is it sets the m_canMoveMiddleMouse boolean to true and sets the m_InitialMousePosition to the current mouse position and the m_InitialCameraPosition to the current camera position. If the middle mouse button is not pressed the function returns false.

**See also**

MouseButtonPressedEvent

**Parameters**

| | | |
|---|---|---|
| in | *MouseButtonPressedEvent&* | e The mouse button pressed event that is triggered. |

#### 9.36.3.14 OnMouseButtonUpEvent()

```
bool Fracture::OrthographicCameraController::OnMouseButtonUpEvent (
            MouseButtonReleasedEvent & e )  [private]
```

Function that is called when a mouse button is released.

The function checks if the middle mouse button is released and if it is it sets the m_canMoveMiddleMouse boolean to false. If the middle mouse button is not released the function returns false.

**See also**

MouseButtonReleasedEvent

**Parameters**

| | | |
|---|---|---|
| in | *MouseButtonReleasedEvent&* | e The mouse button released event that is triggered. |

#### 9.36.3.15 OnMouseScrolledEvent()

```
bool Fracture::OrthographicCameraController::OnMouseScrolledEvent (
            MouseScrolledEvent & e )  [private]
```

Function that is called when mouse is scrolled this modifies the zoom level by zooming in and out.

The function modifies the zoom level by zooming in and out. The zoom level is modified by the mouse scroll value multiplied by the camera zoom speed. The zoom level is then clamped between m_MinZoom and m_MaxZoom. Then the projection matrix is updated.

**See also**

MouseScrolledEvent

OrthographicCameraController::SetZoom

**Parameters**

| | | |
|---|---|---|
| in | *MouseScrolledEvent&* | e The mouse scrolled event that is triggered. |

#### 9.36.3.16 OnUpdate()

```
void Fracture::OrthographicCameraController::OnUpdate (
            Utils::Timestep ts )
```

the Update function that is called every frame by layer that owns the OrthographicCameraController.

**Parameters**

| | | |
|---|---|---|
| in | *Timestep* | ts The time passed since the last frame. |

#### 9.36.3.17 OnWindowResizedEvent()

```
bool Fracture::OrthographicCameraController::OnWindowResizedEvent (
            WindowResizeEvent & e )  [private]
```

Function that is called when the window is resized. This function updates the aspect ratio of the window and updates the projection matrix.

**See also**

WindowResizeEvent

**Parameters**

| | | |
|---|---|---|
| in | *WindowResizeEvent&* | e The window resize event that is triggered. |

#### 9.36.3.18 Rotate()

```
void Fracture::OrthographicCameraController::Rotate (
            float rotation )  [inline]
```

Rotates the OrthographicCamera's rotation by the rotation passed in if rotation is enabled. Only Z axis roation is supported due to this camera currently being 2D.

The function rotates the OrthographicCamera's rotation by the rotation passed in and sets the isChanged boolean to true. This will cause the view matrix to be updated in the OnUpdate function.

**Parameters**

| | | |
|---|---|---|
| in | *float* | rotation The rotation that the OrthographicCamera's rotation will be rotated by. |

#### 9.36.3.19 SetCameraTransform()

```
void Fracture::OrthographicCameraController::SetCameraTransform (
            const TransformComponent & transform )  [inline]
```

Setter for the OrthographicCamera's transform component. Sets the transform component of the OrthographicCamera to the transform component passed in. Sets the isChanged boolean to true.

the transform component being changed causes the view matrix to be updated in the OnUpdate function.

**Parameters**

| | | |
|---|---|---|
| in | *const* | TransformComponent& transform The transform component to be set to. |

#### 9.36.3.20 SetCameraZoomSpeed()

```
void Fracture::OrthographicCameraController::SetCameraZoomSpeed (
            float speed )  [inline]
```

Setter for the camera zoom speed.

**Parameters**

| | | |
|---|---|---|
| in | *float* | speed The speed to set the camera zoom speed to. |

#### 9.36.3.21 SetMaxZoom()

```
void Fracture::OrthographicCameraController::SetMaxZoom (
            float zoom )  [inline]
```

Setter for the max and min zoom levels.

**Parameters**

| | | |
|---|---|---|
| in | *float* | zoom The zoom level to set the max and min zoom levels to. |

#### 9.36.3.22 SetMinZoom()

```
void Fracture::OrthographicCameraController::SetMinZoom (
```

```
            float zoom )  [inline]
```

Setter for the max and min zoom levels.

**Parameters**

| | | |
|---|---|---|
| in | *float* | zoom The zoom level to set the max and min zoom levels to. |

#### 9.36.3.23 SetPosition()

```
void Fracture::OrthographicCameraController::SetPosition (
            const glm::vec3 & position )  [inline]
```

Setter for the OrthographicCamera's position. Sets the position of the OrthographicCamera to the position passed in.

The function sets the position of the OrthographicCamera to the position passed in and sets the isChanged boolean to true. This will cause the view matrix to be updated in the OnUpdate function.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& position The position that the OrthographicCamera's position will be set to. |

#### 9.36.3.24 SetRotation()

```
void Fracture::OrthographicCameraController::SetRotation (
            float rotation )  [inline]
```

Setter for the OrthographicCamera's rotation. Sets the rotation of the OrthographicCamera to the rotation passed in if rotation is enabled. Only Z axis roation is supported due to this camera currently being 2D.

The function sets the rotation of the OrthographicCamera to the rotation passed in and sets the isChanged boolean to true. This will cause the view matrix to be updated in the OnUpdate function.

**Parameters**

| | | |
|---|---|---|
| in | *float* | rotation The rotation that the OrthographicCamera's rotation will be set to. |

#### 9.36.3.25 SetZoom()

```
void Fracture::OrthographicCameraController::SetZoom (
            float zoom )
```

Setter for the OrthographicCamera's zoom level. Sets the zoom level of the OrthographicCamera to the zoom level passed in.

The function sets the zoom level to the zoom level passed in clamps the zoom level between m_MinZoom and m_MaxZoom. Then the projection matrix is updated.

**Parameters**

| | | |
|---|---|---|
| in | *float* | zoom The zoom level to be set to. |

#### 9.36.3.26 ToggleRotation()

```
void Fracture::OrthographicCameraController::ToggleRotation (
            bool enable )  [inline]
```

Toggle to enable or disable camera rotation.

**Parameters**

| | | |
|---|---|---|
| in | *bool* | enable Whether or not to enable camera rotation. |

#### 9.36.3.27 Translate()

```
void Fracture::OrthographicCameraController::Translate (
            const glm::vec3 & translation )  [inline]
```

Translates the OrthographicCamera's position by the translation passed in.

The function translates the OrthographicCamera's position by the translation passed in and sets the isChanged boolean to true. This will cause the view matrix to be updated in the OnUpdate function.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& translation The translation that the OrthographicCamera's position will be translated by. |

#### 9.36.3.28 Zoom()

```
void Fracture::OrthographicCameraController::Zoom (
            float zoom )
```

Setter for the OrthographicCamera's zoom level. Sets the zoom level of the OrthographicCamera to the zoom level passed in.

The function adds the given zoom value to the current zoom level clamps the zoom level between m_MinZoom and m_MaxZoom. Then the projection matrix is updated.

**Parameters**

| | | |
|---|---|---|
| in | *float* | zoom The zoom level to be added to the current zoom level. |

### 9.36.4 Member Data Documentation

#### 9.36.4.1 isChanged

```
bool Fracture::OrthographicCameraController::isChanged = true  [private]
```

The min zoom level of the camera.

#### 9.36.4.2 m_AspectRatio

```
float Fracture::OrthographicCameraController::m_AspectRatio  [private]
```

The delta time stored from the last frame to be used to smooth out the camera movement.

#### 9.36.4.3 m_Camera

```
OrthographicCamera Fracture::OrthographicCameraController::m_Camera  [private]
```

The transform component of the camera.

#### 9.36.4.4 m_cameraRotationSpeed

```
float Fracture::OrthographicCameraController::m_cameraRotationSpeed = 1.0f  [private]
```

The speed of the camera translation.

#### 9.36.4.5 m_CameraTransform

```
TransformComponent Fracture::OrthographicCameraController::m_CameraTransform  [private]
```

Whether or not the camera can rotate.

#### 9.36.4.6 m_cameraTranslationSpeed

```
float Fracture::OrthographicCameraController::m_cameraTranslationSpeed = 1.0f  [private]
```

The scale of the middle mouse movement.

#### 9.36.4.7 m_cameraZoomSpeed

```
float Fracture::OrthographicCameraController::m_cameraZoomSpeed = 40.0f  [private]
```

The speed of the camera rotation.

#### 9.36.4.8 m_canMoveMiddleMouse

```
bool Fracture::OrthographicCameraController::m_canMoveMiddleMouse = false  [private]
```

Boolean to check if the camera has changed.

#### 9.36.4.9 m_EnableRotation

```
bool Fracture::OrthographicCameraController::m_EnableRotation  [private]
```

The current zoom level of the camera.

#### 9.36.4.10 m_InitialCameraPosition

```
glm::vec3 Fracture::OrthographicCameraController::m_InitialCameraPosition = { 0.0f, 0.0f, 0.0f
}  [private]
```

The initial mouse position when the middle mouse button is pressed.

#### 9.36.4.11 m_InitialMousePosition

```
glm::vec2 Fracture::OrthographicCameraController::m_InitialMousePosition = { 0.0f, 0.0f }
[private]
```

The orthographic camera.

#### 9.36.4.12 m_LastFrameTime

```
Utils::Timestep Fracture::OrthographicCameraController::m_LastFrameTime  [private]
```

#### 9.36.4.13 m_MaxZoom

```
float Fracture::OrthographicCameraController::m_MaxZoom = 100.0f  [private]
```

The speed of the camera zoom.

#### 9.36.4.14 m_MiddleMouseScale

```
float Fracture::OrthographicCameraController::m_MiddleMouseScale = 0.005f  [private]
```

The initial camera position when the middle mouse button is pressed.

#### 9.36.4.15 m_MinZoom

```
float Fracture::OrthographicCameraController::m_MinZoom = 0.25f  [private]
```

The max zoom level of the camera.

#### 9.36.4.16 m_ZoomLevel

```
float Fracture::OrthographicCameraController::m_ZoomLevel = 1.0f  [private]
```

The current aspect ratio of the camera frustum.

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

- Fracture/src/Fracture/Renderer/OrthographicCameraController.h
- Fracture/src/Fracture/Renderer/OrthographicCameraController.cpp

## 9.37 Fracture::Utils::ProfileResult Struct Reference

```
#include <Instrumentation.h>
```

### Public Attributes

- std::string Name
- long long Start
- long long End
- uint32_t ThreadID

### 9.37.1 Member Data Documentation

#### 9.37.1.1 End

```
long long Fracture::Utils::ProfileResult::End
```

#### 9.37.1.2 Name

```
std::string Fracture::Utils::ProfileResult::Name
```

#### 9.37.1.3 Start

```
long long Fracture::Utils::ProfileResult::Start
```

#### 9.37.1.4 ThreadID

```
uint32_t Fracture::Utils::ProfileResult::ThreadID
```

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

- Fracture/src/Fracture/Utils/Instrumentation.h

# 9.38 Fracture::RenderCommand Class Reference

The RenderCommand class is used to send commands to the renderer. It is a thin wrapper around the RendererAPI class.

`#include <RenderCommand.h>`

### Static Public Member Functions

- static Scope< RendererAPI > & GetRendererAPI ()

  *Function that initializes the renderer API. This function is called once per renderer. It can be used to get current renderer API It is has a static member variable that is initialized once. This is done to be thread safe.*
- static void DrawIndexed (uint32_t indexCount)

  *Function that draws calls the DrawIndexed function of the current renderer API. Draws the index_count number of indices from the currently bound vertex array.*
- static void SetClearColor (const glm::vec4 &color)

  *Function that sets the clear color of the renderer API. Calls the SetClearColor function of the current renderer API.*
- static void SetViewport (uint32_t x, uint32_t y, uint32_t width, uint32_t height)

  *Function that sets the viewport of the renderer API. Calls the SetViewport function of the current renderer API.*
- static void Clear ()

  *Function that clears the screen to be ready to show the next frame. Calls the Clear function of the current renderer API.*

### Static Private Member Functions

- static Scope< RendererAPI > CreateRendererAPI ()

  *Function that creates the renderer API instance. This function is called once per renderer. It consideres the current renderer set and creates the appropriate renderer API.*

## 9.38.1 Detailed Description

The RenderCommand class is used to send commands to the renderer. It is a thin wrapper around the RendererAPI class.

## 9.38.2 Member Function Documentation

### 9.38.2.1 Clear()

`static void Fracture::RenderCommand::Clear ( ) [inline], [static]`

Function that clears the screen to be ready to show the next frame. Calls the Clear function of the current renderer API.

**See also**

RendererAPI

#### 9.38.2.2 CreateRendererAPI()

```
Scope< RendererAPI > Fracture::RenderCommand::CreateRendererAPI ( )  [static], [private]
```

Function that creates the renderer API instance. This function is called once per renderer. It consideres the current renderer set and creates the appropriate renderer API.

**See also**

RendererAPI

**Returns**

Scope<RendererAPI>: A unuque pointer to the renderer API.

#### 9.38.2.3 DrawIndexed()

```
static void Fracture::RenderCommand::DrawIndexed (
            uint32_t indexCount )  [inline], [static]
```

Function that draws calls the DrawIndexed function of the current renderer API. Draws the index_count number of indices from the currently bound vertex array.

**See also**

RendererAPI

OpenGLRendererAPI

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | indexCount: The number of indices to draw. |

#### 9.38.2.4 GetRendererAPI()

```
static Scope< RendererAPI > & Fracture::RenderCommand::GetRendererAPI ( )  [inline], [static]
```

Function that initializes the renderer API. This function is called once per renderer. It can be used to get current renderer API It is has a static member variable that is initialized once. This is done to be thread safe.

**See also**

RendererAPI

**Returns**

Scope<RendererAPI>&: reference to the current renderer API.

#### 9.38.2.5 SetClearColor()

```
static void Fracture::RenderCommand::SetClearColor (
            const glm::vec4 & color )  [inline], [static]
```

Function that sets the clear color of the renderer API. Calls the SetClearColor function of the current renderer API.

**See also**

RendererAPI

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec4& color: The color to set the clear color to. |

#### 9.38.2.6 SetViewport()

```
static void Fracture::RenderCommand::SetViewport (
            uint32_t x,
            uint32_t y,
            uint32_t width,
            uint32_t height )  [inline], [static]
```

Function that sets the viewport of the renderer API. Calls the SetViewport function of the current renderer API.

The viewport is the area of the window that the renderer will render to. This is usually called on window resize.

**See also**

RendererAPI

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | x: The x coordinate of the viewport. |
| in | *uint32_t* | y: The y coordinate of the viewport. |
| in | *uint32_t* | width: The width of the viewport. |
| in | *uint32_t* | height: The height of the viewport. |

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

- Fracture/src/Fracture/Renderer/RenderCommand.h
- Fracture/src/Fracture/Renderer/RenderCommand.cpp

## 9.39 Fracture::Renderer Class Reference

The Renderer class is used to render a scene. It provides an interface to render a scene.

`#include <Renderer.h>`

### Classes

- struct SceneData

  *This is a temporary structure that is used to store all the data that is needed to render the current scene.*

### Static Public Member Functions

- static void Init ()

  *Function that initializes the renderer. This function is called once application. Currently it only initializes the renderer API.*
- static void BeginScene (OrthographicCamera &camera)

  *Function that declares the beginning of a scene. It sets the view projection matrix for the scene from the provided camera.*
- static void EndScene ()

  *Function that declares the end of a scene. Currently does nothing.*
- static void OnWindowResize (uint32_t width, uint32_t height)

  *The OnWindowResize function is called when the window is resized. It sets the viewport of the renderer API to the new window size.*
- static void Submit (const Ref< VertexArray > &vertexArray, const Ref< Shader > &shader, const glm::mat4 &transform)

  *Function that submits a vertex array, shader, and transform to the renderer API. Sets the shader uniforms and does a draw call.*
- static RendererAPI::API GetAPI ()

  *Function that returns the current renderer API.*

### Static Private Attributes

- static Scope< SceneData > s_SceneData = CreateScope<Renderer::SceneData>()

## 9.39.1 Detailed Description

The Renderer class is used to render a scene. It provides an interface to render a scene.

## 9.39.2 Member Function Documentation

### 9.39.2.1 BeginScene()

```
void Fracture::Renderer::BeginScene (
            OrthographicCamera & camera )  [static]
```

Function that declares the beginning of a scene. It sets the view projection matrix for the scene from the provided camera.

**Todo** : Currently only supports orthographic camera. Add support for any camera.

**Parameters**

| | | |
|---|---|---|
| in | *OrthographicCamera&* | camera: The camera that is used to set the view projection matrix. |

#### 9.39.2.2 EndScene()

```
void Fracture::Renderer::EndScene ( )  [static]
```

Function that declares the end of a scene. Currently does nothing.

#### 9.39.2.3 GetAPI()

```
static RendererAPI::API Fracture::Renderer::GetAPI ( )  [inline], [static]
```

Function that returns the current renderer API.

return RendererAPI::API: The current renderer API.

#### 9.39.2.4 Init()

```
void Fracture::Renderer::Init ( )  [static]
```

Function that initializes the renderer. This function is called once application. Currently it only initializes the renderer API.

#### 9.39.2.5 OnWindowResize()

```
void Fracture::Renderer::OnWindowResize (
            uint32_t width,
            uint32_t height )  [static]
```

The OnWindowResize function is called when the window is resized. It sets the viewport of the renderer API to the new window size.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | width: The new width of the window. |
| in | *uint32_t* | height: The new height of the window. |

#### 9.39.2.6 Submit()

```
void Fracture::Renderer::Submit (
            const Ref< VertexArray > & vertexArray,
```

```
const Ref< Shader > & shader,
const glm::mat4 & transform = glm::mat4(1.0) )  [static]
```

Function that submits a vertex array, shader, and transform to the renderer API. Sets the shader uniforms and does a draw call.

The shader uniforms for the view projection matrix and the model matrix are set. The draw call is done using the current renderer API.

**Todo** : Add support for materails.

: Add support for batch rendering.

: Add support for instanced rendering.

**Parameters**

| | | |
|---|---|---|
| in | *const* | Ref<VertexArray>& vertexArray: Pointer to the vertex array to submit. |
| in | *const* | Ref<Shader>& shader: Pointer to the shader to submit. |
| in | *const* | glm::mat4& transform: The transform to submit (model matrix). |

### 9.39.3 Member Data Documentation

#### 9.39.3.1 s_SceneData

```
Scope< Renderer::SceneData > Fracture::Renderer::s_SceneData = CreateScope<Renderer::SceneData>()
[static], [private]
```

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

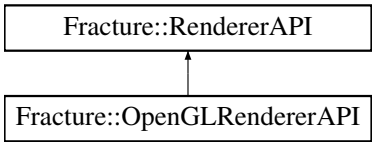
- Fracture/src/Fracture/Renderer/Renderer.h
- Fracture/src/Fracture/Renderer/Renderer.cpp

## 9.40 Fracture::RendererAPI Class Reference

The RendererAPI class provides an interface for the RendererAPI that needs to be implemented by each renderer.

```
#include <RendererAPI.h>
```

Inheritance diagram for Fracture::RendererAPI:



### Public Types

- enum class API { None = 0 , OpenGL = 1 }

  *The API enum class defines the different types of APIs that can be used by the renderer.*

### Public Member Functions

- virtual void Init ()=0

  *Function that initializes the renderer API. Must be implemented by each renderer.*
- virtual void SetClearColor (const glm::vec4 &color)=0

  *Function that sets the clear color of the renderer API. Must be implemented by each renderer.*
- virtual void SetViewport (uint32_t x, uint32_t y, uint32_t width, uint32_t height)=0

  *Function that sets the viewport of the renderer API. Must be implemented by each renderer.*
- virtual void Clear ()=0

  *Function that clears the renderer API. Must be implemented by each renderer.*
- virtual void DrawIndexed (uint32_t indexCount=0)=0

  *Indexed draw call. Must be implemented by each renderer.*
- virtual bool IsInitialized () const =0

  *Function that returns the current state of initialization of the renderer API. Must be implemented by each renderer.*

### Static Public Member Functions

- static API GetAPI ()

  *Function that returns the current API that is being used by the renderer.*

## 9.40.1 Detailed Description

The RendererAPI class provides an interface for the RendererAPI that needs to be implemented by each renderer.

**See also**

OpenGLRendererAPI

RenderCommand

## 9.40.2 Member Enumeration Documentation

### 9.40.2.1 API

```
enum class Fracture::RendererAPI::API  [strong]
```

The API enum class defines the different types of APIs that can be used by the renderer.

**Enumerator**

| | |
|---|---|
| None | |
| OpenGL | |

### 9.40.3 Member Function Documentation

#### 9.40.3.1 Clear()

```
virtual void Fracture::RendererAPI::Clear ( )  [pure virtual]
```

Function that clears the renderer API. Must be implemented by each renderer.

Implemented in Fracture::OpenGLRendererAPI.

#### 9.40.3.2 DrawIndexed()

```
virtual void Fracture::RendererAPI::DrawIndexed (
            uint32_t indexCount = 0 )  [pure virtual]
```

Indexed draw call. Must be implemented by each renderer.

Must draw the index_count number of indices from the currently bound vertex array.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | indexCount: The number of indices to draw |

Implemented in Fracture::OpenGLRendererAPI.

#### 9.40.3.3 GetAPI()

```
static API Fracture::RendererAPI::GetAPI ( )  [inline], [static]
```

Function that returns the current API that is being used by the renderer.

**Todo** : Provide a way for the api to be set by the user and not be hardcoded.

**Returns**

API: The current API that is being used by the renderer.

#### 9.40.3.4 Init()

```
virtual void Fracture::RendererAPI::Init ( )  [pure virtual]
```

Function that initializes the renderer API. Must be implemented by each renderer.

Implemented in Fracture::OpenGLRendererAPI.

#### 9.40.3.5 IsInitialized()

```
virtual bool Fracture::RendererAPI::IsInitialized ( ) const  [pure virtual]
```

Function that returns the current state of initialization of the renderer API. Must be implemented by each renderer.

Implemented in Fracture::OpenGLRendererAPI.

#### 9.40.3.6 SetClearColor()

```
virtual void Fracture::RendererAPI::SetClearColor (
            const glm::vec4 & color )  [pure virtual]
```

Function that sets the clear color of the renderer API. Must be implemented by each renderer.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec4& color: The color to set the clear color to |

Implemented in Fracture::OpenGLRendererAPI.

#### 9.40.3.7 SetViewport()

```
virtual void Fracture::RendererAPI::SetViewport (
            uint32_t x,
            uint32_t y,
            uint32_t width,
            uint32_t height )  [pure virtual]
```

Function that sets the viewport of the renderer API. Must be implemented by each renderer.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | x: The x coordinate of the viewport |
| in | *uint32_t* | y: The y coordinate of the viewport |
| in | *uint32_t* | width: The width of the viewport |
| in | *uint32_t* | height: The height of the viewport |

Implemented in Fracture::OpenGLRendererAPI.

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

- Fracture/src/Fracture/Renderer/RendererAPI.h

## 9.41 Fracture::Renderer::SceneData Struct Reference

This is a temporary structure that is used to store all the data that is needed to render the current scene.

**Public Attributes**

- glm::mat4 ViewProjectionMatrix
- uint32_t CurrentBoundShader = 0

*The view projection matrix of the camera that is used to render the scene.*

### 9.41.1 Detailed Description

This is a temporary structure that is used to store all the data that is needed to render the current scene.

**Todo** : Move this to a scene class.

### 9.41.2 Member Data Documentation

#### 9.41.2.1 CurrentBoundShader

```
uint32_t Fracture::Renderer::SceneData::CurrentBoundShader = 0
```

The view projection matrix of the camera that is used to render the scene.

#### 9.41.2.2 ViewProjectionMatrix

```
glm::mat4 Fracture::Renderer::SceneData::ViewProjectionMatrix
```

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

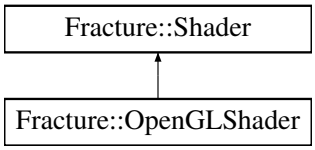
- Fracture/src/Fracture/Renderer/Renderer.h

## 9.42 Fracture::Shader Class Reference

The Shader class is an abstract class that is used to create a shader for the application. Each renderer will have its own implementation of the shader.

```
#include <Shader.h>
```

Inheritance diagram for Fracture::Shader:



### Public Member Functions

- virtual ~Shader ()
- virtual void Bind () const =0

  *Function that binds to be used by the subsequent draw calls.*
- virtual void Unbind () const =0

  *Function that unbinds the shader.*
- virtual void SetInt (const std::string &name, int value)=0
- virtual void SetInt2 (const std::string &name, const glm::ivec2 &values)=0
- virtual void SetInt3 (const std::string &name, const glm::ivec3 &values)=0
- virtual void SetInt4 (const std::string &name, const glm::ivec4 &values)=0
- virtual void SetFloat (const std::string &name, float value)=0
- virtual void SetFloat2 (const std::string &name, const glm::vec2 &values)=0
- virtual void SetFloat3 (const std::string &name, const glm::vec3 &values)=0
- virtual void SetFloat4 (const std::string &name, const glm::vec4 &values)=0
- virtual void SetMat3 (const std::string &name, const glm::mat3 &matrix)=0
- virtual void SetMat4 (const std::string &name, const glm::mat4 &matrix)=0
- virtual void SetBool (const std::string &name, bool value)=0
- virtual const std::string & GetName () const =0

  *Function to get the name of the shader. Must be implemented by the platform specific shader class.*
- virtual const uint32_t & GetHandle () const =0

  *Function to get the handle ID of the shader. Must be implemented by the platform specific shader class.*

### Static Public Member Functions

- static Ref< Shader > Create (const std::string &name, const std::string &vertex_source, const std::string fragment_source)

  *Function that creates a shader from 2 strings containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*
- static Ref< Shader > Create (const std::string &name, const std::string &shaderFilePath)

  *Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*
- static Ref< Shader > Create (const std::string &shaderFilePath)

  *Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.*

## 9.42.1 Detailed Description

The Shader class is an abstract class that is used to create a shader for the application. Each renderer will have its own implementation of the shader.

Each implementation of the shader will have its own renderer ID. The renderer ID is used to identify the shader in the renderer. The implementation should also implement functions to set uniforms in the shader based on the name of the uniform.

**See also**

OpenGLShader

**Todo** : Add a way to automatically find the uniforms that need to be set in the shader.

## 9.42.2 Constructor & Destructor Documentation

### 9.42.2.1 ~Shader()

```
virtual Fracture::Shader::~Shader ( )  [inline], [virtual]
```

## 9.42.3 Member Function Documentation

### 9.42.3.1 Bind()

```
virtual void Fracture::Shader::Bind ( ) const  [pure virtual]
```

Function that binds to be used by the subsequent draw calls.

Implemented in Fracture::OpenGLShader.

### 9.42.3.2 Create() [1/3]

```
Ref< Shader > Fracture::Shader::Create (
            const std::string & name,
            const std::string & shaderFilePath )  [static]
```

Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.

The requirement for the shader source file is to be a glsl shader. The vertex and framgent shader parts of the shader source file should be contained within a _TYPE_VERTEX_SHADER and _TYPE_FRAGMENT_SHADER block respectively. you can use _TYPE_PIXEL_SHADER instead of _TYPE_FRAGMENT_SHADER.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name for the shader. |
| in | *const* | std::string& shaderFilePath: The path to the shader source file. |

**Returns**

A shared pointer to the shader

### 9.42.3.3 Create() [2/3]

```
Ref< Shader > Fracture::Shader::Create (
            const std::string & name,
            const std::string & vertex_source,
            const std::string fragment_source )  [static]
```

Function that creates a shader from 2 strings containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the shader. |
| in | *const* | std::string& vertex_source: The vertex shader source code. |
| in | *const* | std::string& fragment_source: The fragment shader source code. |

**Returns**

A shared pointer to the shader

#### 9.42.3.4 Create() [3/3]

```
Ref< Shader > Fracture::Shader::Create (
            const std::string & shaderFilePath )  [static]
```

Function that creates a shader from a file containing the vertex and fragment shader source code. The shader will be created based on the renderer API that is currently active.

The requirement for the shader source file is to be a glsl shader. The vertex and framgent shader parts of the shader source file should be contained within a _TYPE_VERTEX_SHADER and _TYPE_FRAGMENT_SHADER block respectively. you can use _TYPE_PIXEL_SHADER instead of _TYPE_FRAGMENT_SHADER. The name of the shader will be the name of the file.

If the name is repeated it will cause an error. In most situations it is better to use the other Create function that takes in a name and a filepath.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& shaderFilePath: The path to the shader source file. |

**Returns**

A shared pointer to the shader

#### 9.42.3.5 GetHandle()

```
virtual const uint32_t & Fracture::Shader::GetHandle ( ) const  [pure virtual]
```

Function to get the handle ID of the shader. Must be implemented by the platform specific shader class.

**Returns**

const uint32_t&: The handle ID of the shader.

Implemented in Fracture::OpenGLShader.

#### 9.42.3.6 GetName()

```
virtual const std::string & Fracture::Shader::GetName ( ) const  [pure virtual]
```

Function to get the name of the shader. Must be implemented by the platform specific shader class.

**Returns**

const std::string&: The name of the shader.

Implemented in Fracture::OpenGLShader.

#### 9.42.3.7 SetBool()

```
virtual void Fracture::Shader::SetBool (
            const std::string & name,
            bool value )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.8 SetFloat()

```
virtual void Fracture::Shader::SetFloat (
            const std::string & name,
            float value )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.9 SetFloat2()

```
virtual void Fracture::Shader::SetFloat2 (
            const std::string & name,
            const glm::vec2 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.10 SetFloat3()

```
virtual void Fracture::Shader::SetFloat3 (
            const std::string & name,
            const glm::vec3 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.11 SetFloat4()

```
virtual void Fracture::Shader::SetFloat4 (
            const std::string & name,
            const glm::vec4 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.12 SetInt()

```
virtual void Fracture::Shader::SetInt (
            const std::string & name,
            int value )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.13 SetInt2()

```
virtual void Fracture::Shader::SetInt2 (
            const std::string & name,
            const glm::ivec2 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.14 SetInt3()

```
virtual void Fracture::Shader::SetInt3 (
            const std::string & name,
            const glm::ivec3 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.15 SetInt4()

```
virtual void Fracture::Shader::SetInt4 (
            const std::string & name,
            const glm::ivec4 & values )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.16 SetMat3()

```
virtual void Fracture::Shader::SetMat3 (
            const std::string & name,
            const glm::mat3 & matrix )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.17 SetMat4()

```
virtual void Fracture::Shader::SetMat4 (
            const std::string & name,
            const glm::mat4 & matrix )  [pure virtual]
```

Implemented in Fracture::OpenGLShader.

#### 9.42.3.18 Unbind()

```
virtual void Fracture::Shader::Unbind ( ) const  [pure virtual]
```

Function that unbinds the shader.

Implemented in Fracture::OpenGLShader.

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

- Fracture/src/Fracture/Renderer/Shader.h
- Fracture/src/Fracture/Renderer/Shader.cpp

## 9.43 Fracture::ShaderLibrary Class Reference

The ShaderLibrary class is a singleton class that is used to store all the shaders that are created in the application.

```
#include <Shader.h>
```

### Static Public Member Functions

- static Scope< ShaderLibrary > & GetInstance ()

  *Function to get the instance of the shader library. This is a singleton class. It has a static member variable that is initialized once. This is done to be thread safe.*
- static void Add (const std::string &name, const Ref< Shader > &shader)

  *Function to add a shader to the library. The shader will be added to the library with the name of the shader as the key. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.*
- static void Add (const Ref< Shader > &shader)

  *Function to add a shader to the library. The the key will be the name of the shader. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.*
- static Ref< Shader > Load (const std::string &filepath)

  *Function loads a shader from a filepath, intializes it and adds it to the library. The shader will be added to the library with the name of the shader as the filename. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.*
- static Ref< Shader > Load (const std::string &name, const std::string &filepath)

  *Function loads a shader from a filepath, intializes it and adds it to the library. The shader will be added to the library with the name provided instead of the filename. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.*
- static Ref< Shader > Load (const std::string &name, const std::string &vertexSrc, const std::string &fragmentSrc)

  *Function loads a shader from 2 strings containing the vertex and fragment shader source code, intializes it and adds it to the library. The shader will be added to the library with the name provided. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.*
- static Ref< Shader > Get (const std::string &name)

  *Function to get a shader from the library. The shader will be retrieved from the library with the name provided. If the shader with the same name does not exist it will return a nullptr and cause a warning.*

**Private Member Functions**

- void InitLibrary ()
- void IAdd (const std::string &name, const Ref< Shader > &shader)
- void IAdd (const Ref< Shader > &shader)
- Ref< Shader > ILoad (const std::string &filepath)
- Ref< Shader > ILoad (const std::string &name, const std::string &filepath)
- Ref< Shader > ILoad (const std::string &name, const std::string &vertexSrc, const std::string &fragmentSrc)
- Ref< Shader > IGet (const std::string &name)

**Private Attributes**

- std::unordered_map< std::string, Ref< Shader > > m_Shaders

## 9.43.1 Detailed Description

The ShaderLibrary class is a singleton class that is used to store all the shaders that are created in the application.

The shaders are stored in a map with the name of the shader as the key and the shader reference as the value.

**Todo** : Add a way to make instances of existing shaders.

: Add some default shaders.

## 9.43.2 Member Function Documentation

### 9.43.2.1 Add() [1/2]

```
static void Fracture::ShaderLibrary::Add (
            const Ref< Shader > & shader )  [inline], [static]
```

Function to add a shader to the library. The the key will be the name of the shader. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.

**Parameters**

| | | |
|---|---|---|
| in | *const* | Ref<Shader>& shader: The shader reference to be added to the library. |

### 9.43.2.2 Add() [2/2]

```
static void Fracture::ShaderLibrary::Add (
            const std::string & name,
            const Ref< Shader > & shader )  [inline], [static]
```

Function to add a shader to the library. The shader will be added to the library with the name of the shader as the key. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the shader. |
| in | *const* | Ref<Shader>& shader: The shader reference to be added to the library. |

#### 9.43.2.3 Get()

```
static Ref< Shader > Fracture::ShaderLibrary::Get (
            const std::string & name )  [inline], [static]
```

Function to get a shader from the library. The shader will be retrieved from the library with the name provided. If the shader with the same name does not exist it will return a nullptr and cause a warning.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the shader to be retrieved. |

**Returns**

A shared pointer to the shader

#### 9.43.2.4 GetInstance()

```
static Scope< ShaderLibrary > & Fracture::ShaderLibrary::GetInstance ( )  [inline], [static]
```

Function to get the instance of the shader library. This is a singleton class. It has a static member variable that is initialized once. This is done to be thread safe.

**Returns**

Scope<ShaderLibrary>&: A reference to the shader library.

#### 9.43.2.5 IAdd() [1/2]

```
void Fracture::ShaderLibrary::IAdd (
            const Ref< Shader > & shader )  [private]
```

#### 9.43.2.6 IAdd() [2/2]

```
void Fracture::ShaderLibrary::IAdd (
            const std::string & name,
            const Ref< Shader > & shader )  [private]
```

#### 9.43.2.7 IGet()

```
Ref< Shader > Fracture::ShaderLibrary::IGet (
            const std::string & name )  [private]
```

#### 9.43.2.8 ILoad() [1/3]

```
Ref< Shader > Fracture::ShaderLibrary::ILoad (
            const std::string & filepath )  [private]
```

#### 9.43.2.9 ILoad() [2/3]

```
Ref< Shader > Fracture::ShaderLibrary::ILoad (
            const std::string & name,
            const std::string & filepath )  [private]
```

#### 9.43.2.10 ILoad() [3/3]

```
Ref< Shader > Fracture::ShaderLibrary::ILoad (
            const std::string & name,
            const std::string & vertexSrc,
            const std::string & fragmentSrc )  [private]
```

#### 9.43.2.11 InitLibrary()

```
void Fracture::ShaderLibrary::InitLibrary ( )  [private]
```

#### 9.43.2.12 Load() [1/3]

```
static Ref< Shader > Fracture::ShaderLibrary::Load (
            const std::string & filepath )  [inline], [static]
```

Function loads a shader from a filepath, intializes it and adds it to the library. The shader will be added to the library with the name of the shader as the filename. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& filepath: The path to the shader source file. |

**Returns**

A shared pointer to the shader

#### 9.43.2.13 Load() [2/3]

```
static Ref< Shader > Fracture::ShaderLibrary::Load (
            const std::string & name,
            const std::string & filepath )  [inline], [static]
```

Function loads a shader from a filepath, intializes it and adds it to the library. The shader will be added to the library with the name provided instead of the filename. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the shader. |
| in | *const* | std::string& filepath: The path to the shader source file. |

**Returns**

A shared pointer to the shader

#### 9.43.2.14 Load() [3/3]

```
static Ref< Shader > Fracture::ShaderLibrary::Load (
            const std::string & name,
            const std::string & vertexSrc,
            const std::string & fragmentSrc )  [inline], [static]
```

Function loads a shader from 2 strings containing the vertex and fragment shader source code, intializes it and adds it to the library. The shader will be added to the library with the name provided. If the shader with the same name already exists it will cause a warning but will not cause an error. It will not add the shader to the library.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& name: The name of the shader. |
| in | *const* | std::string& vertexSrc: The vertex shader source code. |
| in | *const* | std::string& fragmentSrc: The fragment shader source code. |

**Returns**

A shared pointer to the shader

### 9.43.3 Member Data Documentation

#### 9.43.3.1 m_Shaders

```
std::unordered_map<std::string, Ref<Shader> > Fracture::ShaderLibrary::m_Shaders  [private]
```

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

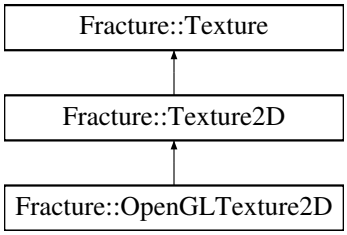
- Fracture/src/Fracture/Renderer/Shader.h
- Fracture/src/Fracture/Renderer/Shader.cpp

## 9.44 Fracture::Texture Class Reference

The Texture class is an abstract class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.

`#include <Texture.h>`

Inheritance diagram for Fracture::Texture:




**Public Member Functions**

- virtual ~Texture ()=default
- virtual uint32_t GetWidth () const =0

  *Function that returns the width of the texture.*
- virtual uint32_t GetHeight () const =0

  *Function that returns the height of the texture.*
- virtual uint32_t GetHandle () const =0

  *Function that returns the handle of the texture.*
- virtual void Bind (uint32_t slot=0) const =0

  *Binds the texture to the specified slot.*

## 9.44.1 Detailed Description

The Texture class is an abstract class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.

## 9.44.2 Constructor & Destructor Documentation

### 9.44.2.1 ~Texture()

```
virtual Fracture::Texture::~Texture ( )  [virtual], [default]
```

## 9.44.3 Member Function Documentation

### 9.44.3.1 Bind()

```
virtual void Fracture::Texture::Bind (
            uint32_t slot = 0 ) const  [pure virtual]
```

Binds the texture to the specified slot.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | slot: The slot to which the texture should be bound. |

Implemented in Fracture::OpenGLTexture2D.

#### 9.44.3.2 GetHandle()

```
virtual uint32_t Fracture::Texture::GetHandle ( ) const  [pure virtual]
```

Function that returns the handle of the texture.

**Returns**

uint32_t: The handle of the texture.

Implemented in Fracture::OpenGLTexture2D.

#### 9.44.3.3 GetHeight()

```
virtual uint32_t Fracture::Texture::GetHeight ( ) const  [pure virtual]
```

Function that returns the height of the texture.

**Returns**

uint32_t: The height of the texture.

Implemented in Fracture::OpenGLTexture2D.

#### 9.44.3.4 GetWidth()

```
virtual uint32_t Fracture::Texture::GetWidth ( ) const  [pure virtual]
```

Function that returns the width of the texture.

**Returns**

uint32_t: The width of the texture.

Implemented in Fracture::OpenGLTexture2D.

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

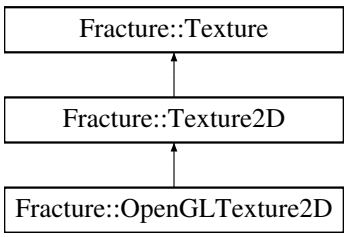
- Fracture/src/Fracture/Renderer/Texture.h

# 9.45 Fracture::Texture2D Class Reference

The Texture2D class is an abstract class that is used to store references to 2D textures needed for rendering. Each renderer will have its own implementation of the texture class.

`#include <Texture.h>`

Inheritance diagram for Fracture::Texture2D:




### Static Public Member Functions

- static Ref< Texture2D > Create (uint32_t width, uint32_t height, glm::vec4 color)

  *Function that creates a 2D texture from a given colour and a width and height. The texture will be 4 channel RGBA.*
- static Ref< Texture2D > Create (const std::string &path)

  *Function that creates a 2D texture from a given path to an image file.*

### Additional Inherited Members

## Public Member Functions inherited from Fracture::Texture

- virtual ~Texture ()=default
- virtual uint32_t GetWidth () const =0

  *Function that returns the width of the texture.*
- virtual uint32_t GetHeight () const =0

  *Function that returns the height of the texture.*
- virtual uint32_t GetHandle () const =0

  *Function that returns the handle of the texture.*
- virtual void Bind (uint32_t slot=0) const =0

  *Binds the texture to the specified slot.*

## 9.45.1 Detailed Description

The Texture2D class is an abstract class that is used to store references to 2D textures needed for rendering. Each renderer will have its own implementation of the texture class.

**See also**

OpenGLTexture

## 9.45.2 Member Function Documentation

### 9.45.2.1 Create() [1/2]

```
Ref< Texture2D > Fracture::Texture2D::Create (
            const std::string & path )  [static]
```

Function that creates a 2D texture from a given path to an image file.

We check the renderer api that is being used and create the appropriate 2D texture for that renderer.

**See also**

OpenGLTexture

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& path: The path to the image file. |

**Returns**

A shared pointer to the 2D texture.

### 9.45.2.2 Create() [2/2]

```
Ref< Texture2D > Fracture::Texture2D::Create (
            uint32_t width,
            uint32_t height,
            glm::vec4 color )  [static]
```

Function that creates a 2D texture from a given colour and a width and height. The texture will be 4 channel RGBA.

We check the renderer api that is being used and create the appropriate 2D texture for that renderer.

**See also**

OpenGLTexture

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | width: The width of the texture. |
| in | *uint32_t* | height: The height of the texture. |
| in | *glm::vec4* | color: The colour of the texture. |

**Returns**

A shared pointer to the 2D texture.

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

- Fracture/src/Fracture/Renderer/Texture.h
- Fracture/src/Fracture/Renderer/Texture.cpp

## 9.46 Fracture::Utils::Timestep Struct Reference

data structure used to store time in seconds

`#include <Helpers.h>`

### Public Member Functions

- Timestep (float time=0)
  *Constructor for the Timestep struct.*
- operator float () const
  *overload for the float operator to the stored time*
- float GetSeconds () const
  *Get the time in seconds.*
- float GetMilliseconds () const
  *Get the time in milliseconds.*
- float GetMicroseconds () const
  *Get the time in microseconds.*

### Private Attributes

- float m_Time

### 9.46.1 Detailed Description

data structure used to store time in seconds

### 9.46.2 Constructor & Destructor Documentation

#### 9.46.2.1 Timestep()

```
Fracture::Utils::Timestep::Timestep (
            float time = 0 )  [inline]
```

Constructor for the Timestep struct.

**Parameters**

| | | |
|---|---|---|
| in | *float* | time: The time to be stored in the timestep in seconds |

## 9.46.3 Member Function Documentation

### 9.46.3.1 GetMicroseconds()

```
float Fracture::Utils::Timestep::GetMicroseconds ( ) const  [inline]
```

Get the time in microseconds.

**Returns**

float: The time in microseconds

### 9.46.3.2 GetMilliseconds()

```
float Fracture::Utils::Timestep::GetMilliseconds ( ) const  [inline]
```

Get the time in milliseconds.

**Returns**

float: The time in milliseconds

### 9.46.3.3 GetSeconds()

```
float Fracture::Utils::Timestep::GetSeconds ( ) const  [inline]
```

Get the time in seconds.

**Returns**

float: The time in seconds

### 9.46.3.4 operator float()

```
Fracture::Utils::Timestep::operator float ( ) const  [inline]
```

overload for the float operator to the stored time

**Returns**

float: The stored time

## 9.46.4 Member Data Documentation

### 9.46.4.1 m_Time

```
float Fracture::Utils::Timestep::m_Time  [private]
```

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

- Fracture/src/Fracture/Utils/Helpers.h

# 9.47 Fracture::TransformComponent Class Reference

`#include <Component.h>`

### Public Member Functions

- TransformComponent ()=default

  *Constructor for the TransformComponent class.*
- TransformComponent (const TransformComponent &)=default

  *Copy Constructor for the TransformComponent class.*
- TransformComponent (const glm::vec3 &translation)

  *Constructor for the TransformComponent class that takes in a position vector.*
- const glm::vec3 & GetPosition ()

  *Getter for a const reference to the position vector of the transform.*
- const glm::vec3 & GetRotation ()

  *Getter for a const reference to the rotation vector of the transform.*
- const glm::vec3 & GetScale ()

  *Getter for a const reference to the scale vector of the transform.*
- void SetPosition (const glm::vec3 &position)

  *Setter for the position vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- void SetRotation (const glm::vec3 &rotation)

  *Setter for the rotation vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- void SetScale (const glm::vec3 &scale)

  *Setter for the scale vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- void Translate (const glm::vec3 &translation)

  *Function that will translate the transform by the given translation vector. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- void Rotate (const glm::vec3 &rotation)

  *Function that will add the given rotation to the current rotation. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- void Scale (const glm::vec3 &scale)

  *Function that will scale the transform by the given scale vector. This will set the isChanged flag to true so that the transform matrix can be recalculated.*
- glm::mat4 GetTransform ()

  *Function that will return the transform matrix of the transformComponent.*
- glm::mat4 GetTransformInverse ()

  *Function that will return the inverse transform matrix of the transformComponent.*

### Private Attributes

- glm::vec3 m_Position = { 0.0f, 0.0f, 0.0f }
- glm::vec3 m_Rotation = { 0.0f, 0.0f, 0.0f }

  *The position vector of the transform.*
- glm::vec3 m_Scale = { 1.0f, 1.0f, 1.0f }

  *The rotation vector of the transform.*
- bool isChanged = true

  *The scale vector of the transform.*
- glm::mat4 m_Transform = glm::mat4(1.0f)

  *A boolean flag that will be set to true if the transform is changed and needs to be recalculated.*
- glm::mat4 m_InverseTransform = glm::mat4(1.0f)

  *The cached transform matrix of the transform.*

### 9.47.1 Constructor & Destructor Documentation

#### 9.47.1.1 TransformComponent() [1/3]

```
Fracture::TransformComponent::TransformComponent ( )  [default]
```

Constructor for the TransformComponent class.

#### 9.47.1.2 TransformComponent() [2/3]

```
Fracture::TransformComponent::TransformComponent (
            const TransformComponent &  )  [default]
```

Copy Constructor for the TransformComponent class.

#### 9.47.1.3 TransformComponent() [3/3]

```
Fracture::TransformComponent::TransformComponent (
            const glm::vec3 & translation )  [inline]
```

Constructor for the TransformComponent class that takes in a position vector.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& translation: The position vector of the transform. |

### 9.47.2 Member Function Documentation

#### 9.47.2.1 GetPosition()

```
const glm::vec3 & Fracture::TransformComponent::GetPosition ( )  [inline]
```

Getter for a const reference to the position vector of the transform.

**Returns**

glm::vec3& The position vector of the transform.

#### 9.47.2.2 GetRotation()

```
const glm::vec3 & Fracture::TransformComponent::GetRotation ( )  [inline]
```

Getter for a const reference to the rotation vector of the transform.

**Returns**

glm::vec3& The rotation vector of the transform.

#### 9.47.2.3 GetScale()

```
const glm::vec3 & Fracture::TransformComponent::GetScale ( )  [inline]
```

Getter for a const reference to the scale vector of the transform.

**Returns**

glm::vec3& The scale vector of the transform.

#### 9.47.2.4 GetTransform()

```
glm::mat4 Fracture::TransformComponent::GetTransform ( )  [inline]
```

Function that will return the transform matrix of the transformComponent.

If the isChanged flag is true then the transform matrix will be recalculated and the isChanged flag will be set to false. If not it will return the cached transform matrix.

**Returns**

glm::mat4 The transform matrix of the transformComponent.

#### 9.47.2.5 GetTransformInverse()

```
glm::mat4 Fracture::TransformComponent::GetTransformInverse ( )  [inline]
```

Function that will return the inverse transform matrix of the transformComponent.

If the isChanged flag is true then the inverse transform matrix will be recalculated and the isChanged flag will be set to false. If not it will return the cached inverse transform matrix.

**Returns**

glm::mat4 The inverse transform matrix of the transformComponent.

#### 9.47.2.6 Rotate()

```
void Fracture::TransformComponent::Rotate (
            const glm::vec3 & rotation )  [inline]
```

Function that will add the given rotation to the current rotation. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& rotation: The rotation vector. |

#### 9.47.2.7 Scale()

```
void Fracture::TransformComponent::Scale (
            const glm::vec3 & scale )  [inline]
```

Function that will scale the transform by the given scale vector. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& scale: The scale vector. |

#### 9.47.2.8 SetPosition()

```
void Fracture::TransformComponent::SetPosition (
            const glm::vec3 & position )  [inline]
```

Setter for the position vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& position: The position vector of the transform. |

#### 9.47.2.9 SetRotation()

```
void Fracture::TransformComponent::SetRotation (
            const glm::vec3 & rotation )  [inline]
```

Setter for the rotation vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& rotation: The rotation vector of the transform. |

#### 9.47.2.10 SetScale()

```
void Fracture::TransformComponent::SetScale (
            const glm::vec3 & scale )  [inline]
```

Setter for the scale vector of the transform. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& scale: The scale vector of the transform. |

#### 9.47.2.11 Translate()

```
void Fracture::TransformComponent::Translate (
            const glm::vec3 & translation )  [inline]
```

Function that will translate the transform by the given translation vector. This will set the isChanged flag to true so that the transform matrix can be recalculated.

**Parameters**

| | | |
|---|---|---|
| in | *const* | glm::vec3& translation: The translation vector. |

## 9.47.3 Member Data Documentation

#### 9.47.3.1 isChanged

```
bool Fracture::TransformComponent::isChanged = true  [private]
```

The scale vector of the transform.

#### 9.47.3.2 m_InverseTransform

```
glm::mat4 Fracture::TransformComponent::m_InverseTransform = glm::mat4(1.0f)  [private]
```

The cached transform matrix of the transform.

#### 9.47.3.3 m_Position

```
glm::vec3 Fracture::TransformComponent::m_Position = { 0.0f, 0.0f, 0.0f }  [private]
```

#### 9.47.3.4 m_Rotation

```
glm::vec3 Fracture::TransformComponent::m_Rotation = { 0.0f, 0.0f, 0.0f }  [private]
```

The position vector of the transform.

#### 9.47.3.5 m_Scale

```
glm::vec3 Fracture::TransformComponent::m_Scale = { 1.0f, 1.0f, 1.0f }  [private]
```

The rotation vector of the transform.

#### 9.47.3.6 m_Transform

```
glm::mat4 Fracture::TransformComponent::m_Transform = glm::mat4(1.0f)  [private]
```

A boolean flag that will be set to true if the transform is changed and needs to be recalculated.

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

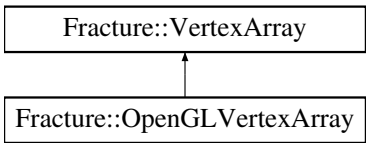
- Fracture/src/Fracture/Components/Component.h

## 9.48 Fracture::VertexArray Class Reference

The VertexArray class is an abstract class that is used to create a VertexArray object. Each renderer will have its own implementation of the VertexArray class.

```
#include <VertexArray.h>
```

Inheritance diagram for Fracture::VertexArray:




### Public Member Functions

- virtual ~VertexArray ()
- virtual void AddVertexBuffer (const Ref< VertexBuffer > &vertexBuffer)=0

  *Function that adds a VertexBuffer to the VertexArray. A vertex array can have multiple vertex buffers.*
- virtual void SetIndexBuffer (const Ref< IndexBuffer > &indexBuffer)=0

  *Function that sets the IndexBuffer of the VertexArray.*
- virtual void Bind () const =0

  *Function that binds the VertexArray. This function is called before drawing the VertexArray.*
- virtual void Unbind () const =0

  *Function that unbinds the VertexArray. This function is called after drawing the VertexArray.*
- virtual const Ref< IndexBuffer > & GetIndexBuffer () const =0

  *Function that returns the IndexBuffer of the VertexArray.*
- virtual const std::vector< Ref< VertexBuffer > > & GetVertexBuffers () const =0

  *Function that returns the VertexBuffers of the VertexArray.*

### Static Public Member Functions

- static Ref< VertexArray > Create ()

  *Function that creates a VertexArray. This function will create a VertexArray based on the current active renderer.*

### 9.48.1 Detailed Description

The VertexArray class is an abstract class that is used to create a VertexArray object. Each renderer will have its own implementation of the VertexArray class.

**See also**

OpenGLVertexArray

### 9.48.2 Constructor & Destructor Documentation

#### 9.48.2.1 ~VertexArray()

```
virtual Fracture::VertexArray::~VertexArray ( )  [inline], [virtual]
```

### 9.48.3 Member Function Documentation

#### 9.48.3.1 AddVertexBuffer()

```
virtual void Fracture::VertexArray::AddVertexBuffer (
            const Ref< VertexBuffer > & vertexBuffer )  [pure virtual]
```

Function that adds a VertexBuffer to the VertexArray. A vertex array can have multiple vertex buffers.

**Parameters**

| | | |
|---|---|---|
| in | *const* | Ref<VertexBuffer>& vertexBuffer: The VertexBuffer to be added to the VertexArray. |

Implemented in Fracture::OpenGLVertexArray.

#### 9.48.3.2 Bind()

```
virtual void Fracture::VertexArray::Bind ( ) const  [pure virtual]
```

Function that binds the VertexArray. This function is called before drawing the VertexArray.

Implemented in Fracture::OpenGLVertexArray.

#### 9.48.3.3 Create()

```
Ref< VertexArray > Fracture::VertexArray::Create ( )  [static]
```

Function that creates a VertexArray. This function will create a VertexArray based on the current active renderer.

We check the renderer api that is being used and create the appropriate VertexArray for that renderer.

**See also**

OpenGLVertexArray

**Returns**

A shared pointer to the VertexArray.

#### 9.48.3.4 GetIndexBuffer()

```
virtual const Ref< IndexBuffer > & Fracture::VertexArray::GetIndexBuffer ( ) const  [pure virtual]
```

Function that returns the IndexBuffer of the VertexArray.

**Returns**

A reference to the IndexBuffer of the VertexArray.

Implemented in Fracture::OpenGLVertexArray.

#### 9.48.3.5 GetVertexBuffers()

```
virtual const std::vector< Ref< VertexBuffer > > & Fracture::VertexArray::GetVertexBuffers ( ) const  [pure virtual]
```

Function that returns the VertexBuffers of the VertexArray.

**Returns**

const std::vector<Ref<VertexBuffer>>&: A reference to the vector of VertexBuffers of the VertexArray.

Implemented in Fracture::OpenGLVertexArray.

#### 9.48.3.6 SetIndexBuffer()

```
virtual void Fracture::VertexArray::SetIndexBuffer (
            const Ref< IndexBuffer > & indexBuffer )  [pure virtual]
```

Function that sets the IndexBuffer of the VertexArray.

**Parameters**

| | | |
|---|---|---|
| in | *const* | Ref<IndexBuffer>& indexBuffer: The IndexBuffer to be set to the VertexArray. |

Implemented in Fracture::OpenGLVertexArray.

#### 9.48.3.7 Unbind()

```
virtual void Fracture::VertexArray::Unbind ( ) const  [pure virtual]
```

Function that unbinds the VertexArray. This function is called after drawing the VertexArray.

Implemented in Fracture::OpenGLVertexArray.

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

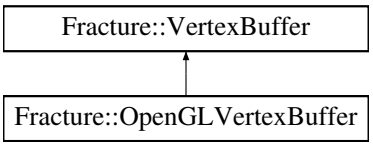
- Fracture/src/Fracture/Renderer/VertexArray.h
- Fracture/src/Fracture/Renderer/VertexArray.cpp

# 9.49 Fracture::VertexBuffer Class Reference

The VertexBuffer class is an abstract class that is used to store the vertex buffer. Each renderer will have its own implementation of the vertex buffer.

`#include <Buffer.h>`

Inheritance diagram for Fracture::VertexBuffer:




### Public Member Functions

- virtual ~VertexBuffer ()=default
- virtual void SetData (const void *data, uint32_t size)=0
- virtual void SetLayout (const BufferLayout &layout)=0
- virtual const BufferLayout & GetLayout () const =0
- virtual void Bind () const =0
- virtual void Unbind () const =0

### Static Public Member Functions

- static Ref< VertexBuffer > Create (float *vertices, uint32_t size)

  *Function that creates a vertex buffer. This function will create a vertex buffer based on the platform that the application is running on.*

## 9.49.1 Detailed Description

The VertexBuffer class is an abstract class that is used to store the vertex buffer. Each renderer will have its own implementation of the vertex buffer.

**See also**

OpenGLVertexBuffer

BufferLayout

## 9.49.2 Constructor & Destructor Documentation

### 9.49.2.1 ~VertexBuffer()

`virtual Fracture::VertexBuffer::~VertexBuffer ( ) [virtual], [default]`

### 9.49.3 Member Function Documentation

#### 9.49.3.1 Bind()

```
virtual void Fracture::VertexBuffer::Bind ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLVertexBuffer.

#### 9.49.3.2 Create()

```
Ref< VertexBuffer > Fracture::VertexBuffer::Create (
            float * vertices,
            uint32_t size )  [static]
```

Function that creates a vertex buffer. This function will create a vertex buffer based on the platform that the application is running on.

We check the renderer api that is being used and create the appropriate vertex buffer for that renderer.

**See also**

OpenGLVertexBuffer

**Parameters**

| | | |
|---|---|---|
| in | *float*∗ | vertices: The vertex data of the vertex buffer. |
| in | *uint32_t* | size: The size of the vertex buffer. |

**Returns**

A shared pointer to the vertex buffer.

#### 9.49.3.3 GetLayout()

```
virtual const BufferLayout & Fracture::VertexBuffer::GetLayout ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLVertexBuffer.

#### 9.49.3.4 SetData()

```
virtual void Fracture::VertexBuffer::SetData (
            const void * data,
            uint32_t size )  [pure virtual]
```

Implemented in Fracture::OpenGLVertexBuffer.

#### 9.49.3.5 SetLayout()

```
virtual void Fracture::VertexBuffer::SetLayout (
            const BufferLayout & layout )  [pure virtual]
```

Implemented in Fracture::OpenGLVertexBuffer.

#### 9.49.3.6 Unbind()

```
virtual void Fracture::VertexBuffer::Unbind ( ) const  [pure virtual]
```

Implemented in Fracture::OpenGLVertexBuffer.

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

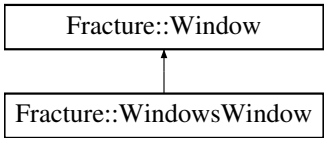
- Fracture/src/Fracture/Renderer/Buffer.h
- Fracture/src/Fracture/Renderer/Buffer.cpp

## 9.50 Fracture::Window Class Reference

Window interface representing a desktop system based Window. This is an abstract class.

`#include <Window.h>`

Inheritance diagram for Fracture::Window:




### Public Types

- using EventCallbackFn = std::function< void(Event &)>

### Public Member Functions

- virtual ~Window ()

  *Defines a type for the event callback function. This is a function that takes an event as a parameter and returns void.*
- virtual void OnUpdate ()=0

  *The function that will be called every frame by the application. Must be implemented by the platform specific window class.*
- virtual uint32_t GetWidth () const =0

  *Function that will return the width of the window. Must be implemented by the platform specific window class.*
- virtual uint32_t GetHeight () const =0

  *Function that will return the height of the window.*
- virtual void SetEventCallback (const EventCallbackFn &callback)=0

  *Function that must be implemented by the platform specific window class. This function will set the event callback function.*
- virtual void SetVSync (bool enabled)=0

  *Function that must be implemented by the platform specific window class. This function will set the VSync flag.*
- virtual bool IsVSync () const =0

  *Function that must be implemented by the platform specific window class. This function will return the VSync flag.*
- virtual void * GetNativeWindow () const =0

  *This is a function that will return a void pointer to the window object. This is used to get the window object from anywhere in the program.*

**Static Public Member Functions**

- static Window * Create (const WindowProperties &properties=WindowProperties())

  *Function that will create a window. This function will create a window based on the platform that the application is running on.*

## 9.50.1 Detailed Description

Window interface representing a desktop system based Window. This is an abstract class.

## 9.50.2 Member Typedef Documentation

### 9.50.2.1 EventCallbackFn

```
using Fracture::Window::EventCallbackFn = std::function<void(Event&)>
```

## 9.50.3 Constructor & Destructor Documentation

### 9.50.3.1 ~Window()

```
virtual Fracture::Window::~Window ( )  [inline], [virtual]
```

Defines a type for the event callback function. This is a function that takes an event as a parameter and returns void.

## 9.50.4 Member Function Documentation

### 9.50.4.1 Create()

```
Window * Fracture::Window::Create (
            const WindowProperties & props = WindowProperties() )  [static]
```

Function that will create a window. This function will create a window based on the platform that the application is running on.

This is a function that will create a window object. This function is defined here because the window class is a platform independent class.

**Todo** : Currently this only creates a WindowsWindow. In the future we will have to check the platform and create the appropriate window.

**See also**

WindowsWindow

**Parameters**

| | | |
|---|---|---|
| in | *const* | WindowProperties& properties: The properties of the window that you want to create. |
| in | *WindowProperties* | props: The properties of the window that is to be created. |

**Returns**

Window*: A pointer to the window object that was created.

#### 9.50.4.2 GetHeight()

```
virtual uint32_t Fracture::Window::GetHeight ( ) const  [pure virtual]
```

Function that will return the height of the window.

**Returns**

uint32_t: The height of the window.

Implemented in Fracture::WindowsWindow.

#### 9.50.4.3 GetNativeWindow()

```
virtual void * Fracture::Window::GetNativeWindow ( ) const  [pure virtual]
```

This is a function that will return a void pointer to the window object. This is used to get the window object from anywhere in the program.

The reason we return a void pointer is this could technically be any window not necessarily a GLFW window.

**Returns**

void*: A void pointer to the window object.

Implemented in Fracture::WindowsWindow.

#### 9.50.4.4 GetWidth()

```
virtual uint32_t Fracture::Window::GetWidth ( ) const  [pure virtual]
```

Function that will return the width of the window. Must be implemented by the platform specific window class.

**Returns**

uint32_t: The width of the window. Must be implemented by the platform specific window class.

Implemented in Fracture::WindowsWindow.

#### 9.50.4.5 IsVSync()

```
virtual bool Fracture::Window::IsVSync ( ) const  [pure virtual]
```

Function that must be implemented by the platform specific window class. This function will return the VSync flag.

Implemented in Fracture::WindowsWindow.

#### 9.50.4.6 OnUpdate()

```
virtual void Fracture::Window::OnUpdate ( )  [pure virtual]
```

The function that will be called every frame by the application. Must be implemented by the platform specific window class.

Implemented in Fracture::WindowsWindow.

#### 9.50.4.7 SetEventCallback()

```
virtual void Fracture::Window::SetEventCallback (
            const EventCallbackFn & callback )  [pure virtual]
```

Function that must be implemented by the platform specific window class. This function will set the event callback function.

**Parameters**

| | | |
|---|---|---|
| in | *const* | EventCallbackFn& callback: The event callback function. |

Implemented in Fracture::WindowsWindow.

#### 9.50.4.8 SetVSync()

```
virtual void Fracture::Window::SetVSync (
            bool enabled )  [pure virtual]
```

Function that must be implemented by the platform specific window class. This function will set the VSync flag.

**Parameters**

| | | |
|---|---|---|
| in | *bool* | enabled: The VSync flag |

Implemented in Fracture::WindowsWindow.

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

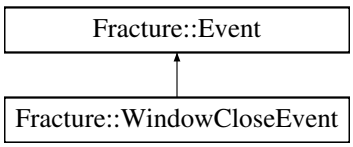
- Fracture/src/Fracture/Core/Window.h
- Fracture/src/Platform/Windows/WindowsWindow.cpp

# 9.51 Fracture::WindowCloseEvent Class Reference

Event class for holding information about window close events.

`#include <ApplicationEvent.h>`

Inheritance diagram for Fracture::WindowCloseEvent:




### Public Member Functions

- WindowCloseEvent ()

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- virtual std::string ToString () const

  *Virtual function, to get the string representation of the event. Default implementation is to return the name of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

## 9.51.1 Detailed Description

Event class for holding information about window close events.

## 9.51.2 Constructor & Destructor Documentation

### 9.51.2.1 WindowCloseEvent()

`Fracture::WindowCloseEvent::WindowCloseEvent ( ) [inline]`

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

- Fracture/src/Fracture/Events/ApplicationEvent.h

## 9.52 Fracture::WindowsWindow::WindowData Struct Reference

A unique pointer to the renderer context.

### Public Member Functions

- WindowData ()=default

  *Whether VSync is enabled or not.*

### Public Attributes

- std::string Title
- uint32_t Width

  *The title of the window.*
- uint32_t Height
- EventCallbackFn EventCallback

  *The width and height of the window.*
- bool VSync

  *The event callback function for the window.*

### 9.52.1 Detailed Description

A unique pointer to the renderer context.

The WindowData struct contains the data of the window that is provided to glfw to set as user data.

**See also**

Window

glfwSetWindowUserPointer

glfwGetWindowUserPointer

### 9.52.2 Constructor & Destructor Documentation

#### 9.52.2.1 WindowData()

```
Fracture::WindowsWindow::WindowData::WindowData ( )  [default]
```

Whether VSync is enabled or not.

### 9.52.3 Member Data Documentation

#### 9.52.3.1 EventCallback

```
EventCallbackFn Fracture::WindowsWindow::WindowData::EventCallback
```

The width and height of the window.

#### 9.52.3.2 Height

```
uint32_t Fracture::WindowsWindow::WindowData::Height
```

#### 9.52.3.3 Title

```
std::string Fracture::WindowsWindow::WindowData::Title
```

#### 9.52.3.4 VSync

```
bool Fracture::WindowsWindow::WindowData::VSync
```

The event callback function for the window.

#### 9.52.3.5 Width

```
uint32_t Fracture::WindowsWindow::WindowData::Width
```

The title of the window.

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

- Fracture/src/Platform/Windows/WindowsWindow.h

## 9.53 Fracture::WindowProperties Struct Reference

Stores the necessary information for a window.

```
#include <Window.h>
```

### Public Member Functions

- WindowProperties (const std::string &title="Fracture Engine", uint32_t width=1280, uint32_t height=720)

  *The height of the window.*

### Public Attributes

- std::string Title
- uint32_t Width

  *The title of the window.*
- uint32_t Height

  *The width of the window.*

### 9.53.1 Detailed Description

Stores the necessary information for a window.

### 9.53.2 Constructor & Destructor Documentation

#### 9.53.2.1 WindowProperties()

```
Fracture::WindowProperties::WindowProperties (
            const std::string & title = "Fracture Engine",
            uint32_t width = 1280,
            uint32_t height = 720 )  [inline]
```

The height of the window.

Constructor for the WindowProperties struct.

**Parameters**

| | | |
|---|---|---|
| in | *const* | std::string& title: The title of the window. |
| in | *uint32_t* | width: The width of the window. |
| in | *uint32_t* | height: The height of the window. |

### 9.53.3 Member Data Documentation

#### 9.53.3.1 Height

```
uint32_t Fracture::WindowProperties::Height
```

The width of the window.

#### 9.53.3.2 Title

```
std::string Fracture::WindowProperties::Title
```

#### 9.53.3.3 Width

```
uint32_t Fracture::WindowProperties::Width
```

The title of the window.

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

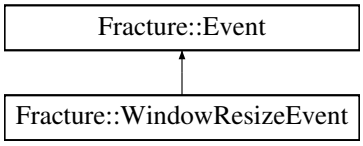
- Fracture/src/Fracture/Core/Window.h

# 9.54 Fracture::WindowResizeEvent Class Reference

Event class for holding information about window resize events.

`#include <ApplicationEvent.h>`

Inheritance diagram for Fracture::WindowResizeEvent:




### Public Member Functions

- WindowResizeEvent (unsigned int width, unsigned int height)

  *Constructor for WindowResizeEvent. Takes the new width and height of the window as parameters.*
- unsigned int GetWidth () const

  *Getter for the new width of the window.*
- unsigned int GetHeight () const

  *Getter for the new height of the window.*
- std::string ToString () const override

  *Converts the WindowResizeEvent data to a string to be serialized or for debugging purposes.*

## Public Member Functions inherited from Fracture::Event

- virtual EventType GetEventType () const =0

  *Pure Virtual function, to get the type of the event.*
- virtual const char * GetName () const =0

  *Pure Virtual function, to get the name of the event.*
- virtual int GetCategoryFlags () const =0

  *Pure Virtual function, to get the category flags of the event.*
- bool IsInCategory (EventCategory category)

  *Function to check if the event is in a certain category.*

### Private Attributes

- unsigned int m_Width
- unsigned int m_Height

  *The new width of the window.*

### Additional Inherited Members

## Public Attributes inherited from Fracture::Event

- bool Handled = false

### 9.54.1 Detailed Description

Event class for holding information about window resize events.

### 9.54.2 Constructor & Destructor Documentation

#### 9.54.2.1 WindowResizeEvent()

```
Fracture::WindowResizeEvent::WindowResizeEvent (
            unsigned int width,
            unsigned int height )  [inline]
```

Constructor for WindowResizeEvent. Takes the new width and height of the window as parameters.

**Parameters**

| | | |
|---|---|---|
| in | *uint32_t* | width: The new width of the window. |
| in | *uint32_t* | height: The new height of the window. |

### 9.54.3 Member Function Documentation

#### 9.54.3.1 GetHeight()

```
unsigned int Fracture::WindowResizeEvent::GetHeight ( ) const  [inline]
```

Getter for the new height of the window.

**Returns**

uint32_t The new height of the window.

#### 9.54.3.2 GetWidth()

```
unsigned int Fracture::WindowResizeEvent::GetWidth ( ) const  [inline]
```

Getter for the new width of the window.

**Returns**

uint32_t The new width of the window.

#### 9.54.3.3 ToString()

```
std::string Fracture::WindowResizeEvent::ToString ( ) const  [inline], [override], [virtual]
```

Converts the WindowResizeEvent data to a string to be serialized or for debugging purposes.

**Returns**

std::string The WindowResizeEvent data as a string.

Reimplemented from Fracture::Event.

### 9.54.4 Member Data Documentation

#### 9.54.4.1 m_Height

```
unsigned int Fracture::WindowResizeEvent::m_Height  [private]
```

The new width of the window.

#### 9.54.4.2 m_Width

```
unsigned int Fracture::WindowResizeEvent::m_Width  [private]
```

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

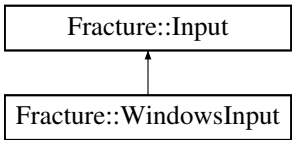
- Fracture/src/Fracture/Events/ApplicationEvent.h

## 9.55 Fracture::WindowsInput Class Reference

The Windows implementation of the Input class.

```
#include <WindowsInput.h>
```

Inheritance diagram for Fracture::WindowsInput:



#### Protected Member Functions

- virtual bool IsKeyPressedImpl (int keyCode) override

  *Implementation of the KeyPressed polling function.*
- virtual bool IsMouseButtonPressedImpl (int button) override

  *Implementation of the MouseButtonPressed polling function.*
- virtual float GetMouseXImpl () override

  *Implementation of the GetMouseX polling function.*
- virtual float GetMouseYImpl () override

  *Implementation of the GetMouseY polling function.*
- virtual std::pair< float, float > GetMousePositionImpl () override

  *Implementation of the GetMousePosition polling function.*

### Protected Member Functions inherited from Fracture::Input

- Input ()=default

  *protected constructor so that only the child classes can create an instance of this class.*

#### Additional Inherited Members

### Public Member Functions inherited from Fracture::Input

- Input (const Input &)=delete

  *Deleted copy constructor so that we can not copy this class since it is a singleton.*
- Input & operator= (const Input &)=delete

  *Deleted assignment operator so that we can not copy this class since it is a singleton.*

### Static Public Member Functions inherited from Fracture::Input

- static bool IsKeyPressed (int keyCode)

  *Static function that returns if a key is pressed or not.*
- static bool IsMouseButtonPressed (int button)

  *Static function that returns if a mouse button is pressed or not.*
- static float GetMouseX ()

  *Static function that returns the current x coordinate of the mouse.*
- static float GetMouseY ()

  *Static function that returns the current y coordinate of the mouse.*
- static std::pair< float, float > GetMousePosition ()

  *Static function that returns the current x and y coordinates of the mouse at once.*

## 9.55.1 Detailed Description

The Windows implementation of the Input class.

## 9.55.2 Member Function Documentation

### 9.55.2.1 GetMousePositionImpl()

```
std::pair< float, float > Fracture::WindowsInput::GetMousePositionImpl ( )  [override], [protected],
[virtual]
```

Implementation of the GetMousePosition polling function.

uses glfwGetCursorPos to get the mouse position

**Returns**

std::pair<float, float>: The x and y coordinates of the mouse cursor

Implements Fracture::Input.

### 9.55.2.2 GetMouseXImpl()

```
float Fracture::WindowsInput::GetMouseXImpl ( )  [override], [protected], [virtual]
```

Implementation of the GetMouseX polling function.

uses GetMousePositionImpl to get the x position and maintain the same code path.

**Returns**

float: The x coordinate of the mouse cursor

Implements Fracture::Input.

### 9.55.2.3 GetMouseYImpl()

```
float Fracture::WindowsInput::GetMouseYImpl ( )  [override], [protected], [virtual]
```

Implementation of the GetMouseY polling function.

uses GetMousePositionImpl to get the x position and maintain the same code path.

**Returns**

float: The y coordinate of the mouse cursor

Implements Fracture::Input.

### 9.55.2.4 IsKeyPressedImpl()

```
bool Fracture::WindowsInput::IsKeyPressedImpl (
            int keyCode )  [override], [protected], [virtual]
```

Implementation of the KeyPressed polling function.

uses glfwGetKey to poll the key

**Parameters**

| | | |
|---|---|---|
| in | *int* | keyCode: The key code of the key that is being polled |

**Returns**

bool: True if the key is pressed, false otherwise

Implements Fracture::Input.

#### 9.55.2.5 IsMouseButtonPressedImpl()

```
bool Fracture::WindowsInput::IsMouseButtonPressedImpl (
            int button )  [override], [protected], [virtual]
```

Implementation of the MouseButtonPressed polling function.

uses glfwGetMouseButton to poll the mouse button

**Parameters**

| | | |
|---|---|---|
| in | *int* | button: The mouse button code of the mouse button that is being polled |

**Returns**

bool: True if the mouse button is pressed, false otherwise

Implements Fracture::Input.

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

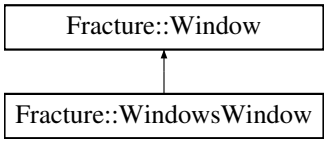
- Fracture/src/Platform/Windows/WindowsInput.h
- Fracture/src/Platform/Windows/WindowsInput.cpp

## 9.56 Fracture::WindowsWindow Class Reference

The WindowsWindow class is used to create a window for the application. It owns the renderer context.

```
#include <WindowsWindow.h>
```

Inheritance diagram for Fracture::WindowsWindow:



### Classes

- struct WindowData

  *A unique pointer to the renderer context.*

### Public Member Functions

- WindowsWindow (const WindowProperties &props)

  *Constructor for the WindowsWindow class.*
- virtual ~WindowsWindow ()

  *Destructor for the WindowsWindow class.*
- void OnUpdate () override

  *Function that updates the window.*
- uint32_t GetWidth () const override

  *Function that returns the width of the window.*
- uint32_t GetHeight () const override

  *Function that returns the height of the window.*
- void SetEventCallback (const EventCallbackFn &callback) override

  *Function that sets the event callback function for the window.*
- void SetVSync (bool enabled) override

  *Function that sets the VSync for the window.*
- bool IsVSync () const override

  *Function that returns whether VSync is enabled or not.*
- virtual void * GetNativeWindow () const override

  *Function that returns a pointer to the native window.*

## Public Member Functions inherited from Fracture::Window

- virtual ~Window ()

  *Defines a type for the event callback function. This is a function that takes an event as a parameter and returns void.*

### Private Member Functions

- virtual void Init (const WindowProperties &props)

  *Function that initializes the window.*
- virtual void Shutdown ()

  *Function that shuts down the window.*

### Private Attributes

- GLFWwindow * m_Window
- Scope< GraphicsContext > m_Context

  *A pointer to the GLFW window.*
- WindowData m_Data

### Additional Inherited Members

## Public Types inherited from Fracture::Window

- using EventCallbackFn = std::function< void(Event &)>

### Static Public Member Functions inherited from Fracture::Window

- static Window * Create (const WindowProperties &properties=WindowProperties())

  *Function that will create a window. This function will create a window based on the platform that the application is running on.*

## 9.56.1 Detailed Description

The WindowsWindow class is used to create a window for the application. It owns the renderer context.

**See also**

Window

## 9.56.2 Constructor & Destructor Documentation

### 9.56.2.1 WindowsWindow()

```
Fracture::WindowsWindow::WindowsWindow (
            const WindowProperties & props )
```

Constructor for the WindowsWindow class.

Calls the Init() function to initialize the window.

**See also**

Init

### 9.56.2.2 ~WindowsWindow()

```
Fracture::WindowsWindow::~WindowsWindow ( )  [virtual]
```

Destructor for the WindowsWindow class.

Calls the Shutdown() function to shutdown the window.

**See also**

Shutdown

## 9.56.3 Member Function Documentation

### 9.56.3.1 GetHeight()

```
uint32_t Fracture::WindowsWindow::GetHeight ( ) const  [inline], [override], [virtual]
```

Function that returns the height of the window.

**Returns**

uint32_t: The height of the window.

Implements Fracture::Window.

#### 9.56.3.2 GetNativeWindow()

```
virtual void * Fracture::WindowsWindow::GetNativeWindow ( ) const  [inline], [override], [virtual]
```

Function that returns a pointer to the native window.

**Returns**

void*: A pointer to the native window.

Implements Fracture::Window.

#### 9.56.3.3 GetWidth()

```
uint32_t Fracture::WindowsWindow::GetWidth ( ) const  [inline], [override], [virtual]
```

Function that returns the width of the window.

**Returns**

uint32_t: The width of the window.

Implements Fracture::Window.

#### 9.56.3.4 Init()

```
void Fracture::WindowsWindow::Init (
            const WindowProperties & props )  [private], [virtual]
```

Function that initializes the window.

Initializes the window using GLFW. Creates the window and the renderer context. Then sets up the event callback function for glfw.

**Parameters**

| | | |
|---|---|---|
| in | *const* | WindowProperties& props: The properties of the window. |

**See also**

WindowProperties

#### 9.56.3.5 IsVSync()

```
bool Fracture::WindowsWindow::IsVSync ( ) const  [override], [virtual]
```

Function that returns whether VSync is enabled or not.

**Returns**

bool: Whether VSync is enabled or not.

Implements Fracture::Window.

#### 9.56.3.6 OnUpdate()

```
void Fracture::WindowsWindow::OnUpdate ( )  [override], [virtual]
```

Function that updates the window.

Calls the glfwPollEvents function to poll for events. And then uses the context to swap the buffers.

Implements Fracture::Window.

#### 9.56.3.7 SetEventCallback()

```
void Fracture::WindowsWindow::SetEventCallback (
            const EventCallbackFn & callback )  [inline], [override], [virtual]
```

Function that sets the event callback function for the window.

**Parameters**

| | | |
|---|---|---|
| in | *const* | EventCallbackFn& callback: The callback function to set. |

**See also**

EventCallbackFn

Implements Fracture::Window.

#### 9.56.3.8 SetVSync()

```
void Fracture::WindowsWindow::SetVSync (
            bool enabled )  [override], [virtual]
```

Function that sets the VSync for the window.

**Parameters**

| | | |
|---|---|---|
| in | *bool* | enabled: Whether to enable or disable VSync. |

Implements Fracture::Window.

#### 9.56.3.9 Shutdown()

```
void Fracture::WindowsWindow::Shutdown ( )  [private], [virtual]
```

Function that shuts down the window.

Shuts down the window and the renderer context.

## 9.56.4 Member Data Documentation

### 9.56.4.1 m_Context

```
Scope<GraphicsContext> Fracture::WindowsWindow::m_Context  [private]
```

A pointer to the GLFW window.

### 9.56.4.2 m_Data

```
WindowData Fracture::WindowsWindow::m_Data  [private]
```

### 9.56.4.3 m_Window

```
GLFWwindow* Fracture::WindowsWindow::m_Window  [private]
```

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

- Fracture/src/Platform/Windows/WindowsWindow.h
- Fracture/src/Platform/Windows/WindowsWindow.cpp

# Chapter 10

# File Documentation

## 10.1 Fracture/src/Fracture.h File Reference

Main header file for the Fracture engine. Contains all the includes for the engine to be provided to the user.

```
#include "Fracture\Utils\Log.h"
#include "Fracture\Utils\Instrumentation.h"
#include "Fracture\Utils\Helpers.h"
#include "Fracture\Core\Application.h"
#include "Fracture\Core\Layer.h"
#include "imgui/imgui.h"
#include "Fracture\Input\Input.h"
#include "Fracture\Input\KeyCodes.h"
#include "Fracture\Input\MouseButtonCodes.h"
#include "Fracture\Events\Event.h"
#include "Fracture\Events\ApplicationEvent.h"
#include "Fracture\Events\KeyEvent.h"
#include "Fracture\Events\MouseEvent.h"
#include "Fracture\Renderer\Renderer.h"
#include "Fracture\Renderer\RenderCommand.h"
#include "Fracture\Renderer\Shader.h"
#include "Fracture\Renderer\Buffer.h"
#include "Fracture\Renderer\VertexArray.h"
#include "Fracture\Renderer\OrthographicCamera.h"
#include "Fracture\Renderer\OrthographicCameraController.h"
#include "Fracture\Renderer\Texture.h"
#include "Fracture\Components\Component.h"
```

### Macros

- #define MAX_SHADER_TYPE_COUNT 2

### 10.1.1 Detailed Description

Main header file for the Fracture engine. Contains all the includes for the engine to be provided to the user.

**Author**

Aditya Rajagopal

### 10.1.2 Macro Definition Documentation

#### 10.1.2.1 MAX_SHADER_TYPE_COUNT

```
#define MAX_SHADER_TYPE_COUNT 2
```

## 10.2 Fracture.h

Go to the documentation of this file.

```
00001 #pragma once
00009 // This is only for external use
00010
00011 // --- Utils ----------------------
00012 #include "Fracture\Utils\Log.h"
00013 #include "Fracture\Utils\Instrumentation.h"
00014 #include "Fracture\Utils\Helpers.h"
00015
00016 // For use by Fracture applications
00017 #include "Fracture\Core\Application.h"
00018 #include "Fracture\Core\Layer.h"
00019 #include "imgui/imgui.h"
00020
00021 // --- Input ----------------------
00022 #include "Fracture\Input\Input.h"
00023 #include "Fracture\Input\KeyCodes.h"
00024 #include "Fracture\Input\MouseButtonCodes.h"
00025
00026 // --- Events ---------------------
00027 #include "Fracture\Events\Event.h"
00028 #include "Fracture\Events\ApplicationEvent.h"
00029 #include "Fracture\Events\KeyEvent.h"
00030 #include "Fracture\Events\MouseEvent.h"
00031
00032 // --- Renderer ----------------------]
00033 #include "Fracture\Renderer\Renderer.h"
00034 #include "Fracture\Renderer\RenderCommand.h"
00035 #include "Fracture\Renderer\Shader.h"
00036 #include "Fracture\Renderer\Buffer.h"
00037 #include "Fracture\Renderer\VertexArray.h"
00038 #include "Fracture\Renderer\OrthographicCamera.h"
00039 #include "Fracture\Renderer\OrthographicCameraController.h"
00040 #include "Fracture\Renderer\Texture.h"
00041
00042 // --- Components ----------------------
00043 #include "Fracture\Components\Component.h"
00044
```

## 10.3 Fracture/src/Fracture/Components/Component.h File Reference

Contians all the components that can be attached to an entity.

```
#include <glm\glm.hpp>
#include <glm\gtc\matrix_transform.hpp>
#include <glm\gtx\quaternion.hpp>
```

#### Classes

- class Fracture::TransformComponent

#### Namespaces

- namespace Fracture

**Macros**

- #define GLM_ENABLE_EXPERIMENTAL

### 10.3.1 Detailed Description

Contians all the components that can be attached to an entity.

**Todo** : The system currently does not support ECS style component system. This will be implemented in the future.

**Author**

Aditya Rajagopal

### 10.3.2 Macro Definition Documentation

#### 10.3.2.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

## 10.4 Component.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include <glm\glm.hpp>
00012 #include <glm\gtc\matrix_transform.hpp>
00013
00014 #define GLM_ENABLE_EXPERIMENTAL
00015 #include <glm\gtx\quaternion.hpp>
00016
00017 namespace Fracture {
00018
00019     class TransformComponent
00020     {
00021     public:
00025         TransformComponent() = default;
00026
00030         TransformComponent(const TransformComponent&) = default;
00031
00037         TransformComponent(const glm::vec3& translation)
00038             : m_Position(translation) {}
00039
00045         const glm::vec3& GetPosition() { return m_Position; }
00046
00052         const glm::vec3& GetRotation() { return m_Rotation; }
00053
00059         const glm::vec3& GetScale() { return m_Scale; }
00060
00066         void SetPosition(const glm::vec3& position) { m_Position = position; isChanged = true; }
00067
00073         void SetRotation(const glm::vec3& rotation) { m_Rotation = rotation; isChanged = true; }
00074
00080         void SetScale(const glm::vec3& scale) { m_Scale = scale; isChanged = true; }
00081
00087         void Translate(const glm::vec3& translation) { m_Position += translation; isChanged = true; }
00088
00094         void Rotate(const glm::vec3& rotation) { m_Rotation += rotation; isChanged = true; }
00095
00101         void Scale(const glm::vec3& scale) { m_Scale += scale; isChanged = true; }
00102
00103
00111         glm::mat4 GetTransform()
00112         {
```

```
00113             if (isChanged)
00114             {
00115                 glm::mat4 rotation = glm::toMat4(glm::quat(m_Rotation));
00116                 glm::mat4 translation = glm::translate(glm::mat4(1.0f), m_Position);
00117                 glm::mat4 scale = glm::scale(glm::mat4(1.0f), m_Scale);
00118                 m_Transform = translation * rotation * scale;
00119                 isChanged = false;
00120             }
00121             return m_Transform;
00122         }
00123 
00131         glm::mat4 GetTransformInverse()
00132         {
00133             if (isChanged)
00134             {
00135                 glm::mat4 rotation = glm::toMat4(glm::quat(m_Rotation));
00136                 glm::mat4 translation = glm::translate(glm::mat4(1.0f), m_Position);
00137                 glm::mat4 scale = glm::scale(glm::mat4(1.0f), m_Scale);
00138                 m_InverseTransform = glm::inverse(translation * rotation * scale);
00139                 isChanged = false;
00140             }
00141             return m_InverseTransform;
00142         }
00143     private:
00144         glm::vec3 m_Position = { 0.0f, 0.0f, 0.0f };
00145         glm::vec3 m_Rotation = { 0.0f, 0.0f, 0.0f };
00146         glm::vec3 m_Scale = { 1.0f, 1.0f, 1.0f };
00147         bool isChanged = true;
00148 
00149         glm::mat4 m_Transform = glm::mat4(1.0f);
00150         glm::mat4 m_InverseTransform = glm::mat4(1.0f);
00151     };
00152 
00153 
00154 }
```

## 10.5 Fracture/src/Fracture/Core/Application.cpp File Reference

```
#include "frpch.h"
#include "Application.h"
#include "Fracture\Renderer\Shader.h"
#include "Fracture\Renderer\RenderCommand.h"
#include "Fracture\Renderer\Renderer.h"
#include "Fracture\Input\Input.h"
#include "Fracture\KeyCodes.h"
```

### Namespaces

- namespace Fracture

## 10.6 Fracture/src/Fracture/Core/Application.h File Reference

Application header file.

```
#include "Core.h"
#include "Window.h"
#include "Fracture\Events\Event.h"
#include "Fracture\Events\ApplicationEvent.h"
#include "Fracture\Events\MouseEvent.h"
#include "Fracture\Events\KeyEvent.h"
#include "Fracture\Core\LayerStack.h"
#include "Fracture\ImGui\ImGuiLayer.h"
```

### Classes

- class Fracture::Application

  *The Application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers.*

### Namespaces

- namespace Fracture

### Functions

- Application * Fracture::CreateApplication ()

## 10.6.1 Detailed Description

Application header file.

Contains the Application class and the CreateApplication function. The Application class is the base class for the engine. The application class is responsible for creating the window, running the main loop, and updating the layers.

The CreateApplication function is used to create the application class in the EntryPoint. This function is defined in the client application.

**See also**

EntryPoint.h

Window

LayerStack

ImGuiLayer

Event

**Author**

Aditya Rajagopal

# 10.7 Application.h

Go to the documentation of this file.

```
00001 #pragma once
00020 #include "Core.h"
00021 #include "Window.h"
00022
00023 #include "Fracture\Events\Event.h"
00024 #include "Fracture\Events\ApplicationEvent.h"
00025 #include "Fracture\Events\MouseEvent.h"
00026 #include "Fracture\Events\KeyEvent.h"
00027
00028 #include "Fracture\Core\LayerStack.h"
00029 #include "Fracture\ImGui\ImGuiLayer.h"
00030
00031 namespace Fracture {
00032
00033
00045     class FRACTURE_API Application
```

```
00046     {
00047     public:
00058         Application();
00059         virtual ~Application() = default;
00060
00074         void Run();
00075
00093         void OnEvent(Event& e);
00094
00106         void PushLayer(Layer* layer);
00107
00119         void PushOverlay(Layer* layer);
00120
00126         inline Window& GetWindow() { return *m_Window; }
00127
00128
00134         inline static Application& Get() { return *s_Instance; }
00135     private:
00144         bool OnWindowClose(WindowCloseEvent& e);
00145
00154         bool OnWindowResize(WindowResizeEvent& e);
00155     private:
00164         Ref<Window> m_Window;
00165
00167         LayerStack m_LayerStack;
00168         ImGuiLayer* m_ImGuiLayer;
00169
00170         bool m_Running = true;
00171         bool m_isMinimized = false;
00172
00173         long long m_LastFrameTime = 0;
00174     private:
00175         static Application* s_Instance;
00176     };
00177
00178     // To be defined in CLIENT
00179     Application* CreateApplication();
00180 } // namespace Fracture
00181
```

## 10.8 Fracture/src/Fracture/Core/Core.h File Reference

Core header file.

```
#include <memory>
```

### Namespaces

- namespace Fracture

### Macros

- #define FR_ASSERT(x, ...)
- #define FR_CORE_ASSERT(x, ...)
- #define BIT(x) (1 << x)
- #define FRACTURE_BIND_EVENT_FN(x) std::bind(&x, this, std::placeholders::_1)

### Typedefs

- template<typename T >
  using Fracture::Scope = std::unique_ptr< T >
- template<typename T >
  using Fracture::Ref = std::shared_ptr< T >

### Functions

- template<typename T , typename ... Args>
  constexpr Scope< T > Fracture::CreateScope (Args &&... args)
- template<typename T , typename ... Args>
  constexpr Ref< T > Fracture::CreateRef (Args &&... args)

## 10.8.1 Detailed Description

Core header file.

Contains the core macros and functions to be used throughout the engine.

**Author**

Aditya Rajagopal

## 10.8.2 Macro Definition Documentation

### 10.8.2.1 BIT

```
#define BIT(
              x ) (1 << x)
```

### 10.8.2.2 FR_ASSERT

```
#define FR_ASSERT(
              x,
              ...  )
```

### 10.8.2.3 FR_CORE_ASSERT

```
#define FR_CORE_ASSERT(
              x,
              ...  )
```

### 10.8.2.4 FRACTURE_BIND_EVENT_FN

```
#define FRACTURE_BIND_EVENT_FN(
              x ) std::bind(&x, this, std::placeholders::_1)
```

## 10.9 Core.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include <memory>
00013
00014 #ifdef FR_PLATFORM_WINDOWS
00015     #ifdef FR_DYNAMIC_LINK
00016         #ifdef FR_BUILD_DLL
00017             #define FRACTURE_API __declspec(dllexport)
00018         #else
00019             #define FRACTURE_API __declspec(dllimport)
00020         #endif
00021     #else
00022         #define FRACTURE_API
00023     #endif
00024 #else
00025     #error Fracture only supports Windows!
00026 #endif
00027
00028 #ifdef FR_DEBUG
00029     #define FR_ENABLE_ASSERTS
00030 #endif
00031
00032 #ifdef FR_ENABLE_ASSERTS
00033     #define FR_ASSERT(x, ...) { if(!(x)) { FR_ERROR("Assertion Failed: "); FR_ERROR(__VA_ARGS__);
      __debugbreak(); } }
00034     #define FR_CORE_ASSERT(x, ...) { if(!(x)) { FR_CORE_ERROR("Assertion Failed: ");
      FR_CORE_ERROR(__VA_ARGS__); __debugbreak(); } }
00035 #else
00036     #define FR_ASSERT(x, ...)
00037     #define FR_CORE_ASSERT(x, ...)
00038 #endif
00039
00040 #define BIT(x) (1 << x) // This is a bit shift operator. It shifts the bit 1 to the left x times. So
      BIT(0) = 00000001, BIT(1) = 00000010, BIT(2) = 00000100, etc.
00041
00042 #define FRACTURE_BIND_EVENT_FN(x) std::bind(&x, this, std::placeholders::_1)
00043
00044 namespace Fracture{
00045
00046     template<typename T>
00047     using Scope = std::unique_ptr<T>;
00048     template<typename T, typename ... Args>
00049     constexpr Scope<T> CreateScope(Args&& ... args)
00050     {
00051         return std::make_unique<T>(std::forward<Args>(args)...);
00052     }
00053
00054
00055     template<typename T>
00056     using Ref = std::shared_ptr<T>;
00057
00058     template<typename T, typename ... Args>
00059     constexpr Ref<T> CreateRef(Args&& ... args)
00060     {
00061         return std::make_shared<T>(std::forward<Args>(args)...);
00062     }
00063
00064 }
```

## 10.10 Fracture/src/Fracture/Core/Layer.cpp File Reference

```
#include "frpch.h"
#include "Layer.h"
```

### Namespaces

- namespace Fracture

## 10.11 Fracture/src/Fracture/Core/Layer.h File Reference

Layer header file. Contains the Layer class.

```
#include "Fracture\Core\Core.h"
#include "Fracture\Events\Event.h"
#include "Fracture\Utils\Helpers.h"
```

### Classes

- class Fracture::Layer

  *The Layer class is the base class for all layers in the engine. Layers are used to separate different parts of the application and set an order of execution.*

### Namespaces

- namespace Fracture

### 10.11.1 Detailed Description

Layer header file. Contains the Layer class.

**See also**

Layer

LayerStack

Application

**Author**

Aditya Rajagopal

## 10.12 Layer.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "Fracture\Core\Core.h"
00014 #include "Fracture\Events\Event.h"
00015
00016 #include "Fracture\Utils\Helpers.h"
00017
00018 namespace Fracture
00019 {
00025     class FRACTURE_API Layer
00026     {
00027     public:
00033         Layer(const std::string& name = "Layer");
00034         virtual ~Layer() = default;
00035
00039         virtual void OnAttach() {};
00040
00044         virtual void OnDetach() {};
00045
00051         virtual void OnUpdate(Utils::Timestep delta_time) {};
00052
00061         virtual void OnEvent(Event& event) {};
00062
00068         virtual void OnImGuiRender() {};
00069
00075         inline const std::string& GetName() const { return m_DebugName; }
00076     protected:
00077         std::string m_DebugName;
00078     };
00079
00080 } // namespace Fracture
00081
```

## 10.13 Fracture/src/Fracture/Core/LayerStack.cpp File Reference

```
#include "frpch.h"
#include "LayerStack.h"
```

### Namespaces

- namespace Fracture

## 10.14 Fracture/src/Fracture/Core/LayerStack.h File Reference

LayerStack header file Contains the LayerStack class that is used to store all the layers that are currently active.

```
#include "frpch.h"
#include "Fracture\Core\Core.h"
#include "Layer.h"
```

### Classes

- class Fracture::LayerStack

  *The LayerStack class is used to store all the layers that are currently active.*

### Namespaces

- namespace Fracture

### 10.14.1 Detailed Description

LayerStack header file Contains the LayerStack class that is used to store all the layers that are currently active.

**See also**

Layer

LayerStack

Application

**Author**

Aditya Rajagopal

## 10.15 LayerStack.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "frpch.h"
00014 #include "Fracture\Core\Core.h"
00015 #include "Layer.h"
00016
00017
00018 namespace Fracture {
00019
00029     class FRACTURE_API LayerStack
00030     {
00031     public:
00032         LayerStack();
00033
00039         ~LayerStack();
00040
00050         void PushLayer(Layer* layer);
00051
00061         void PushOverlay(Layer* layer);
00062
00074         void PopLayer(Layer* layer);
00075
00087         void PopOverlay(Layer* layer);
00088
00094         std::vector<Layer*>::iterator begin() { return m_Layers.begin(); }
00095
00101         std::vector<Layer*>::iterator end() { return m_Layers.end(); }
00102     private:
00103         std::vector<Layer*> m_Layers;
00104         uint32_t m_LayerInsertIndex = 0;
00105     };
00106 }
```

## 10.16 Fracture/src/Fracture/Core/Window.h File Reference

Window header file containing the Window class and the WindowProperties struct.

```
#include "frpch.h"
#include "Fracture\Core\Core.h"
#include "Fracture\Events\Event.h"
```

### Classes

- struct Fracture::WindowProperties

  *Stores the necessary information for a window.*
- class Fracture::Window

  *Window interface representing a desktop system based Window. This is an abstract class.*

### Namespaces

- namespace Fracture

### 10.16.1 Detailed Description

Window header file containing the Window class and the WindowProperties struct.

**See also**

Window

WindowProperties

Application

**Author**

Aditya Rajagopal

## 10.17 Window.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "frpch.h"
00014
00015 #include "Fracture\Core\Core.h"
00016 #include "Fracture\Events\Event.h"
00017
00018 namespace Fracture {
00019
00023     struct WindowProperties
00024     {
00025         std::string Title;
00026         uint32_t Width;
00027         uint32_t Height;
00028
00036         WindowProperties(const std::string& title = "Fracture Engine",
00037             uint32_t width = 1280,
00038             uint32_t height = 720)
00039             :Title(title), Width(width), Height(height)
00040         {
00041         }
00042     };
00043
00047     class FRACTURE_API Window
00048     {
00049     public:
00050         using EventCallbackFn = std::function<void(Event&)>;
00051
00052         virtual ~Window() {}
00053
00057         virtual void OnUpdate() = 0;
00058
00064         virtual uint32_t GetWidth() const = 0;
00065
00071         virtual uint32_t GetHeight() const = 0;
00072
00078         virtual void SetEventCallback(const EventCallbackFn& callback) = 0;
00079
00085         virtual void SetVSync(bool enabled) = 0;
00086
00090         virtual bool IsVSync() const = 0;
00091
00101         static Window* Create(const WindowProperties& properties = WindowProperties());
00102
00110         virtual void* GetNativeWindow() const = 0;
00111     };
00112
00113 }
00114
```

## 10.18 Fracture/src/Fracture/EntryPoint.h File Reference

Contains the main function of the application. This is the entry point of the engine.

### 10.18.1 Detailed Description

Contains the main function of the application. This is the entry point of the engine.

**See also**

Application

**Todo** : Add support for other platforms.

**Author**

: Aditya Rajagopal

## 10.19 EntryPoint.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #ifdef FR_PLATFORM_WINDOWS
00014
00015     extern Fracture::Application* Fracture::CreateApplication();
00016
00024     void main(int argc, char** argv)
00025     {
00026         FR_BEGIN_PROFILE_SESSION("Startup", "../Logs/FractureProfile-Startup.json");
00027         {
00028             FR_PROFILE_SCOPE("Log Init");
00029             Fracture::Log::Init();
00030         }
00031         FR_END_PROFILE_SESSION();
00032         FR_CORE_TRACE("Initialized Fracture Log!");
00033         FR_CORE_WARN("Initialized Fracture Log!");
00034         FR_WARN("Initialized Game Log with macros!");
00035
00036         auto app = Fracture::CreateApplication();
00037         app->Run();
00038         delete app;
00039     }
00040
00041 #endif
```

## 10.20 Fracture/src/Fracture/Events/ApplicationEvent.h File Reference

ApplicationEvent header file containing event definitions for windows and application events.

```
#include "Event.h"
```

**Classes**

- class Fracture::WindowResizeEvent

  *Event class for holding information about window resize events.*
- class Fracture::WindowCloseEvent

  *Event class for holding information about window close events.*
- class Fracture::AppTickEvent
- class Fracture::AppUpdateEvent
- class Fracture::AppRenderEvent

### Namespaces

- namespace Fracture

### 10.20.1 Detailed Description

ApplicationEvent header file containing event definitions for windows and application events.

**See also**

Event

**Author**

Aditya Rajagopal

## 10.21 ApplicationEvent.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include "Event.h"
00012
00013
00014 namespace Fracture {
00015
00019     class FRACTURE_API WindowResizeEvent : public Event
00020     {
00021     public:
00028         WindowResizeEvent(unsigned int width, unsigned int height) // Constructor
00029             : m_Width(width), m_Height(height) {}
00030
00036         inline unsigned int GetWidth() const { return m_Width; }
00037
00043         inline unsigned int GetHeight() const { return m_Height; }
00044
00045
00051         std::string ToString() const override
00052         {
00053             std::stringstream ss;
00054             ss << "WindowResizeEvent: W:" << m_Width << ", H:" << m_Height;
00055             return ss.str();
00056         }
00057
00058         EVENT_CLASS_CATEGORY(EventCategoryApplication)
00059         EVENT_CLASS_TYPE(WindowResize)
00060     private:
00061         unsigned int m_Width;
00062         unsigned int m_Height;
00063     };
00064
00068     class FRACTURE_API WindowCloseEvent : public Event
00069     {
00070     public:
00071         WindowCloseEvent() {} // Constructor
00072
00073         EVENT_CLASS_CATEGORY(EventCategoryApplication)
00074         EVENT_CLASS_TYPE(WindowClose)
00075     };
00076
00077     class FRACTURE_API AppTickEvent : public Event
00078     {
00079     public:
00080         AppTickEvent() {} // Constructor
00081
00082         EVENT_CLASS_CATEGORY(EventCategoryApplication)
00083         EVENT_CLASS_TYPE(AppTick)
00084     };
00085
00086     class FRACTURE_API AppUpdateEvent : public Event
00087     {
```

```
00088     public:
00089         AppUpdateEvent() {} // Constructor
00090 
00091         EVENT_CLASS_CATEGORY(EventCategoryApplication)
00092         EVENT_CLASS_TYPE(AppUpdate)
00093     };
00094 
00095     class FRACTURE_API AppRenderEvent : public Event
00096     {
00097     public:
00098         AppRenderEvent() {} // Constructor
00099 
00100         EVENT_CLASS_CATEGORY(EventCategoryApplication)
00101         EVENT_CLASS_TYPE(AppRender)
00102     };
00103 
00104 }
```

## 10.22 Fracture/src/Fracture/Events/Event.h File Reference

Contains classes for storing Keyboard events.

```
#include "frpch.h"
#include "Fracture\Core\Core.h"
```

### Classes

- class Fracture::Event

  *Base class for all events.*
- class Fracture::EventDispatcher

### Namespaces

- namespace Fracture

### Macros

- #define EVENT_CLASS_TYPE(type)
- #define EVENT_CLASS_CATEGORY(category) virtual int GetCategoryFlags() const override { return category; }

  *Macro that overrides the virtual function in the base class to return the category flags of the event.*

### Enumerations

- enum class Fracture::EventType {
  Fracture::None = 0 , Fracture::WindowClose , Fracture::WindowResize , Fracture::WindowFocus ,
  Fracture::WindowLostFocus , Fracture::WindowMoved , Fracture::AppTick , Fracture::AppUpdate ,
  Fracture::AppRender , Fracture::KeyPressed , Fracture::KeyReleased , Fracture::KeyTyped ,
  Fracture::MouseButtonPressed , Fracture::MouseButtonReleased , Fracture::MouseMoved , Fracture::MouseScrolled
  }

  *Enum class for the different types of events.*
- enum Fracture::EventCategory {
  Fracture::None = 0 , Fracture::EventCategoryApplication = BIT(0) , Fracture::EventCategoryInput = BIT(1) ,
  Fracture::EventCategoryKeyboard = BIT(2) ,
  Fracture::EventCategoryMouse = BIT(3) , Fracture::EventCategoryMouseButton = BIT(4) }

  *Enum class for the different categories of events. These are bit masks that can be combined.*

### Functions

- std::ostream & Fracture::operator<< (std::ostream &stream, const Event &event)

  *Overload of the << operator for events. It calls the ToString() function of the event and then pushes it to the stream.*

## 10.22.1 Detailed Description

Contains classes for storing Keyboard events.

**See also**

Event

**Author**

Aditya Rajagopal

## 10.22.2 Macro Definition Documentation

### 10.22.2.1 EVENT_CLASS_CATEGORY

```
#define EVENT_CLASS_CATEGORY(
              category ) virtual int GetCategoryFlags() const override { return category; }
```

Macro that overrides the virtual function in the base class to return the category flags of the event.

### 10.22.2.2 EVENT_CLASS_TYPE

```
#define EVENT_CLASS_TYPE(
              type )
```

**Value:**

```
static EventType GetStaticType() { return EventType::##type; }\
virtual EventType GetEventType() const override { return GetStaticType(); }\
virtual const char* GetName() const override { return #type; }
```

# 10.23 Event.h

Go to the documentation of this file.

```
00001 #pragma once
00002 /*
00003 * @file Event.h
00004 * @brief Event header file containing the Event class and the EventDispatcher class.
00005 *
00006 * @see ApplicationEvent.h
00007 * @see KeyEvent.h
00008 * @see MouseEvent.h
00009 *
00010 * @author Aditya Rajagopal
00011 */
00012
00013 #include "frpch.h"
00014 #include "Fracture\Core\Core.h"
00015
00016
```

```
00017 namespace Fracture {
00018
00019     //TODO: Buffer events and process during update stage in an events pass
00020
00024     enum class EventType
00025     {
00026         // These are implemented in the individual event classes
00027         None = 0,
00028         WindowClose, WindowResize, WindowFocus, WindowLostFocus, WindowMoved,
00029         AppTick, AppUpdate, AppRender,
00030         KeyPressed, KeyReleased, KeyTyped,
00031         MouseButtonPressed, MouseButtonReleased, MouseMoved, MouseScrolled
00032     };
00033
00037     enum EventCategory
00038     {
00039         None = 0,
00040         EventCategoryApplication    = BIT(0),
00041         EventCategoryInput            = BIT(1),
00042         EventCategoryKeyboard      = BIT(2),
00043         EventCategoryMouse            = BIT(3),
00044         EventCategoryMouseButton    = BIT(4),
00045     };
00046
00047 // Here ## is the token pasting operator (https://en.cppreference.com/w/cpp/preprocessor/replace)
00048 // This is the override of the virtual function in the base class
00049 // We get the string representation of the type of the event class (e.g. "WindowResize")
00050 // We need a GetStaticType() to get what type of event it is in a polymorphic way (e.g. to check if it
      is a KeyPressedEvent)
00051 // In dispatching the events we will use the GetStaticType() to check what type of event an incoming
      event is and check if it is the same as the type of the event we are trying to dispatch
00052 #define EVENT_CLASS_TYPE(type) static EventType GetStaticType() { return EventType::##type; }\
00053                                virtual EventType GetEventType() const override { return
      GetStaticType(); }\
00054                                virtual const char* GetName() const override { return #type; }
00055
00057 #define EVENT_CLASS_CATEGORY(category) virtual int GetCategoryFlags() const override { return
      category; }
00058
00062     class FRACTURE_API Event
00063     {
00064         friend class EventDispatcher; // The event dispatcher can access the protected members of the
      event class
00065     public:
00073         virtual EventType GetEventType() const = 0; //
00074
00080         virtual const char* GetName() const = 0;
00081
00087         virtual int GetCategoryFlags() const = 0;
00088
00094         virtual std::string ToString() const { return GetName(); }
00095
00101         inline bool IsInCategory(EventCategory category)
00102         {
00103             return GetCategoryFlags() & category; // This is a bitwise AND operation. It checks if the
      category is in the flags of the event
00104         }
00105
00106         bool Handled = false;
00107     };
00108
00109     class EventDispatcher
00110     {
00111     public:
00117         EventDispatcher(Event& event) : mEvent(event) {}
00118
00129         template<typename T, typename F>
00130         bool Dispatch(const F& func)
00131         {
00132             if (mEvent.GetEventType() == T::GetStaticType())
00133             {
00134                 // We convert the mEvent reference to a pointer of type T with (T*) and then
      dereference it with *
00135                 // since we have defined EventFn<T> as function that takes a reference of event of
      type T and returns a bool
00136                 mEvent.Handled = func(static_cast<T&>(mEvent)); // Call the function and cast the
      event to type T
00137                 return true; // Return true if the event is of type T
00138             }
00139             return false; // Return false if the event is not of type T
00140         }
00141     private:
00142         Event& mEvent;
00143     };
00144
00153     inline std::ostream& operator<<(std::ostream& stream, const Event& event)
00154     {
```

```
00155         return stream « event.ToString(); // This calls the ToString() function of the event
00156     }
00157
00158 }
```

## 10.24 Fracture/src/Fracture/Events/KeyEvent.h File Reference

```
#include "Event.h"
```

### Classes

- class Fracture::KeyEvent

  *the base class for KeyEvents*
- class Fracture::KeyPressedEvent

  *Event class for when a key is pressed.*
- class Fracture::KeyReleasedEvent

  *Event class for when a key is released.*
- class Fracture::KeyTypedEvent

  *Event class for when a key is typed.*

### Namespaces

- namespace Fracture

## 10.25 KeyEvent.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include "Event.h"
00012
00013
00014 namespace Fracture {
00015
00021     class FRACTURE_API KeyEvent : public Event
00022     {
00023     public:
00029         inline int GetKeyCode() const { return m_KeyCode; }
00030
00036         inline int GetKeyMods() const { return m_Mods; }
00037
00038         /*
00039         * @brief KeyEvent is both a keyboard event and an input event
00040         */
00041         EVENT_CLASS_CATEGORY(EventCategoryKeyboard | EventCategoryInput)
00042     protected: // Protected so that only the child classes can access the keycode
00049         KeyEvent(int keyCode, int mods)
00050             : m_KeyCode(keyCode), m_Mods(mods) {}
00051         int m_KeyCode;
00052         int m_Mods;
00053     };
00054
00061     class FRACTURE_API KeyPressedEvent : public KeyEvent
00062     {
00063     public:
00072         KeyPressedEvent(int keyCode, bool repeatCount, int mods)
00073             : KeyEvent(keyCode, mods), m_IsRepeated(repeatCount) {}
00074
00080         inline bool IsRepeated() const { return m_IsRepeated; }
00081
00087         std::string ToString() const override
00088         {
```

```
00089             std::stringstream ss;
00090             ss « "KeyPressedEvent: " « m_KeyCode « m_IsRepeated ? "(Repeated)" : "";
00091             return ss.str();
00092         }
00093
00094         EVENT_CLASS_TYPE(KeyPressed) // This macro is defined in Event.h and it overrides the virtual
      functions in the base class to return the type of the event and the name of the event.
00095     private:
00096         bool m_IsRepeated;
00097     };
00098
00099
00106     class FRACTURE_API KeyReleasedEvent : public KeyEvent
00107     {
00108     public:
00115         KeyReleasedEvent(int keyCode, int mods) // Constructor
00116             : KeyEvent(keyCode, mods) {}
00117
00121         std::string ToString() const override
00122         {
00123             std::stringstream ss;
00124             ss « "KeyReleasedEvent: " « m_KeyCode;
00125             return ss.str();
00126         }
00127
00128         EVENT_CLASS_TYPE(KeyReleased)
00129     };
00130
00137     class FRACTURE_API KeyTypedEvent : public KeyEvent
00138     {
00139     public:
00140         KeyTypedEvent(int keyCode) // Constructor
00141             : KeyEvent(keyCode, 0) {}
00142
00143         std::string ToString() const override
00144         {
00145             std::stringstream ss;
00146             ss « "KeyTypedEvent: " « m_KeyCode;
00147             return ss.str();
00148         }
00149
00150         EVENT_CLASS_TYPE(KeyTyped)
00151     };
00152
00153 }
```

## 10.26 Fracture/src/Fracture/Events/MouseEvent.h File Reference

Contains classes for storing Mouse events.

```
#include "Event.h"
```

### Classes

- class Fracture::MouseMovedEvent

  *Event for when the mouse is moved.*
- class Fracture::MouseScrolledEvent

  *Event for when the mouse is scrolled.*
- class Fracture::MouseButtonEvent

  *Base class for mouse button events.*
- class Fracture::MouseButtonPressedEvent

  *Event class for when a mouse button is pressed.*
- class Fracture::MouseButtonReleasedEvent

  *Event class for when a mouse button is released.*

### Namespaces

- namespace Fracture

### 10.26.1 Detailed Description

Contains classes for storing Mouse events.

**See also**

Event

**Author**

Aditya Rajagopal

## 10.27 MouseEvent.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include "Event.h"
00013
00014
00015 namespace Fracture {
00016
00022     class FRACTURE_API MouseMovedEvent : public Event
00023     {
00024     public:
00031         MouseMovedEvent(float x, float y) // Constructor
00032             : m_MouseX(x), m_MouseY(y) {}
00033
00039         inline float GetX() const { return m_MouseX; }
00040
00046         inline float GetY() const { return m_MouseY; }
00047
00048         /*
00049         * @brief Serialise the event data to string
00050         */
00051         std::string ToString() const override
00052         {
00053             std::stringstream ss;
00054             ss << "MouseMovedEvent: Position (" << m_MouseX << ", " << m_MouseY << ")";
00055             return ss.str();
00056         }
00057
00058         EVENT_CLASS_TYPE(MouseMoved)
00059         /*
00060         * @brief The category of the event is both mouse and
00061         */
00062         EVENT_CLASS_CATEGORY(EventCategoryMouse | EventCategoryInput)
00063     private:
00064         float m_MouseX; /* @brief x position of the event */
00065         float m_MouseY; /* @brief y position of the event */
00066     };
00067
00073     class FRACTURE_API MouseScrolledEvent : public Event
00074     {
00075     public:
00076         /*
00077         * @brief Constructor for the MouseScrolledEvent
00078         *
00079         * @details The offset of the mouse scroll is the distance the mouse wheel has moved in the x
      and y direction. We also consider the case where the mouse wheel is moved horizontally
00080         *
00081         * @param[in] float xOffset the offset of the mouse scroll in the x direction
00082         * @param[in] float yOffset the offset of the mouse scroll in the y direction
00083         */
00084         MouseScrolledEvent(float xOffset, float yOffset) // Constructor
00085             : m_XOffset(xOffset), m_YOffset(yOffset) {}
00086
```

```
00092         inline float GetXOffset() const { return m_XOffset; }
00093
00099         inline float GetYOffset() const { return m_YOffset; }
00100
00104         std::string ToString() const override
00105         {
00106             std::stringstream ss;
00107             ss << "MouseScrolledEvent: Offset(" << m_XOffset << ", " << m_YOffset << ")";
00108             return ss.str();
00109         }
00110
00111         EVENT_CLASS_TYPE(MouseScrolled)
00112         EVENT_CLASS_CATEGORY(EventCategoryMouse | EventCategoryInput)
00113     private:
00114         float m_XOffset;
00115         float m_YOffset;
00116     };
00117
00124     class FRACTURE_API MouseButtonEvent : public Event
00125     {
00126     public:
00132         inline int GetMouseButton() const { return m_Button; }
00133
00139         inline int GetMouseMod() const { return m_Mods; }
00140
00141         EVENT_CLASS_CATEGORY(EventCategoryMouse | EventCategoryInput)
00142     protected:
00149         MouseButtonEvent(int button, int mods)
00150             : m_Button(button), m_Mods(mods) {}
00151         int m_Button;
00152         int m_Mods;
00153     };
00154
00161     class FRACTURE_API MouseButtonPressedEvent : public MouseButtonEvent
00162     {
00163     public:
00164         MouseButtonPressedEvent(int button, int mods)
00165             : MouseButtonEvent(button, mods) {}
00166
00167         std::string ToString() const override
00168         {
00169             std::stringstream ss;
00170             ss << "MouseButtonPressedEvent: " << m_Button;
00171             return ss.str();
00172         }
00173
00174         EVENT_CLASS_TYPE(MouseButtonPressed)
00175     };
00176
00183     class FRACTURE_API MouseButtonReleasedEvent : public MouseButtonEvent
00184     {
00185     public:
00186         MouseButtonReleasedEvent(int button, int mods)
00187             : MouseButtonEvent(button, mods) {}
00188
00189         std::string ToString() const override
00190         {
00191             std::stringstream ss;
00192             ss << "MouseButtonReleasedEvent: " << m_Button;
00193             return ss.str();
00194         }
00195
00196         EVENT_CLASS_TYPE(MouseButtonReleased)
00197     };
00198
00199 }
```

## 10.28 Fracture/src/Fracture/ImGui/ImGuiBuild.cpp File Reference

```
#include "frpch.h"
#include <misc/cpp/imgui_stdlib.cpp>
#include <backends\imgui_impl_opengl3.cpp>
#include <backends\imgui_impl_glfw.cpp>
```

### Macros

- #define IMGUI_IMPL_OPENGL_LOADER_GLAD

### 10.28.1 Macro Definition Documentation

#### 10.28.1.1 IMGUI_IMPL_OPENGL_LOADER_GLAD

```
#define IMGUI_IMPL_OPENGL_LOADER_GLAD
```

## 10.29 Fracture/src/Fracture/ImGui/ImGuiLayer.cpp File Reference

```
#include "frpch.h"
#include "ImGuiLayer.h"
#include "imgui.h"
#include "Fracture\Core\Core.h"
#include "Fracture\Core\Application.h"
#include "backends\imgui_impl_glfw.h"
#include "backends\imgui_impl_opengl3.h"
#include <glad\glad.h>
#include <GLFW\glfw3.h>
```

#### Namespaces

- namespace Fracture

#### Macros

- #define IMGUI_IMPL_API

### 10.29.1 Macro Definition Documentation

#### 10.29.1.1 IMGUI_IMPL_API

```
#define IMGUI_IMPL_API
```

## 10.30 Fracture/src/Fracture/ImGui/ImGuiLayer.h File Reference

```
#include "Fracture/Core/Layer.h"
#include "Fracture\Events\KeyEvent.h"
#include "Fracture\Events\MouseEvent.h"
#include "Fracture\Events\ApplicationEvent.h"
```

#### Classes

- class Fracture::ImGuiLayer

### Namespaces

- namespace Fracture

## 10.31 ImGuiLayer.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Fracture/Core/Layer.h"
00004 #include "Fracture\Events\KeyEvent.h"
00005 #include "Fracture\Events\MouseEvent.h"
00006 #include "Fracture\Events\ApplicationEvent.h"
00007
00008 namespace Fracture {
00009
00010     class FRACTURE_API ImGuiLayer : public Layer
00011     {
00012     public:
00013         ImGuiLayer();
00014         ~ImGuiLayer() = default;
00015
00016         virtual void OnAttach() override;
00017         virtual void OnDetach() override;
00018         virtual void OnImGuiRender() override;
00019
00020         void Begin();
00021         void End();
00022     private:
00023         float m_Time = 0.0f;
00024     };
00025
00026 }
```

## 10.32 Fracture/src/Fracture/Input/Input.h File Reference

Input header file conatins the singleton class that will be implemented per platform to handle polling inputs.

`#include "Fracture\Core\Core.h"`

### Classes

- class Fracture::Input

  *The base class for Input polling. This class will be implemented per platform.*

### Namespaces

- namespace Fracture

### 10.32.1 Detailed Description

Input header file conatins the singleton class that will be implemented per platform to handle polling inputs.

**See also**

WindowsInput

KeyCodes.h

MouseButtonCodes.h

**Author**

Aditya Rajagopal

## 10.33 Input.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "Fracture\Core\Core.h"
00014
00015 namespace Fracture {
00016
00024     class FRACTURE_API Input
00025     {
00026     protected:
00030         Input() = default;
00031     public:
00035         Input(const Input&) = delete;
00036
00040         Input& operator=(const Input&) = delete;
00041
00049         inline static bool IsKeyPressed(int keyCode) { return s_Instance->IsKeyPressedImpl(keyCode); }
00050
00058         inline static bool IsMouseButtonPressed(int button) { return
      s_Instance->IsMouseButtonPressedImpl(button); }
00059
00065         inline static float GetMouseX() { return s_Instance->GetMouseXImpl(); }
00066
00072         inline static float GetMouseY() { return s_Instance->GetMouseYImpl(); }
00073
00079         inline static std::pair<float, float> GetMousePosition() { return
      s_Instance->GetMousePositionImpl(); }
00080     protected:
00081         // These are the backend pure virtual functions that will be implemented per platform.
00082         virtual bool IsKeyPressedImpl(int keyCode) = 0;
00083         virtual bool IsMouseButtonPressedImpl(int button) = 0;
00084         virtual float GetMouseXImpl() = 0;
00085         virtual float GetMouseYImpl() = 0;
00086         virtual std::pair<float, float> GetMousePositionImpl() = 0;
00087     private:
00088         static Scope<Input> s_Instance;
00089     };
00090
00091 }
```

## 10.34 Fracture/src/Fracture/Input/KeyCodes.h File Reference

KeyCodes header file.

### Macros

- #define FR_KEY_SPACE 32
- #define FR_KEY_APOSTROPHE 39 /* ' */
- #define FR_KEY_COMMA 44 /* , */
- #define FR_KEY_MINUS 45 /* - */
- #define FR_KEY_PERIOD 46 /* . */
- #define FR_KEY_SLASH 47 /* / */
- #define FR_KEY_0 48
- #define FR_KEY_1 49
- #define FR_KEY_2 50
- #define FR_KEY_3 51
- #define FR_KEY_4 52
- #define FR_KEY_5 53
- #define FR_KEY_6 54
- #define FR_KEY_7 55
- #define FR_KEY_8 56
- #define FR_KEY_9 57
- #define FR_KEY_SEMICOLON 59 /* ; */
- #define FR_KEY_EQUAL 61 /* = */

- #define FR_KEY_A 65
- #define FR_KEY_B 66
- #define FR_KEY_C 67
- #define FR_KEY_D 68
- #define FR_KEY_E 69
- #define FR_KEY_F 70
- #define FR_KEY_G 71
- #define FR_KEY_H 72
- #define FR_KEY_I 73
- #define FR_KEY_J 74
- #define FR_KEY_K 75
- #define FR_KEY_L 76
- #define FR_KEY_M 77
- #define FR_KEY_N 78
- #define FR_KEY_O 79
- #define FR_KEY_P 80
- #define FR_KEY_Q 81
- #define FR_KEY_R 82
- #define FR_KEY_S 83
- #define FR_KEY_T 84
- #define FR_KEY_U 85
- #define FR_KEY_V 86
- #define FR_KEY_W 87
- #define FR_KEY_X 88
- #define FR_KEY_Y 89
- #define FR_KEY_Z 90
- #define FR_KEY_LEFT_BRACKET 91 /* [ */
- #define FR_KEY_BACKSLASH 92 /* \ */
- #define FR_KEY_RIGHT_BRACKET 93 /* ] */
- #define FR_KEY_GRAVE_ACCENT 96 /* ` */
- #define FR_KEY_WORLD_1 161 /* non-US #1 */
- #define FR_KEY_WORLD_2 162 /* non-US #2 */
- #define FR_KEY_ESCAPE 256
- #define FR_KEY_ENTER 257
- #define FR_KEY_TAB 258
- #define FR_KEY_BACKSPACE 259
- #define FR_KEY_INSERT 260
- #define FR_KEY_DELETE 261
- #define FR_KEY_RIGHT 262
- #define FR_KEY_LEFT 263
- #define FR_KEY_DOWN 264
- #define FR_KEY_UP 265
- #define FR_KEY_PAGE_UP 266
- #define FR_KEY_PAGE_DOWN 267
- #define FR_KEY_HOME 268
- #define FR_KEY_END 269
- #define FR_KEY_CAPS_LOCK 280
- #define FR_KEY_SCROLL_LOCK 281
- #define FR_KEY_NUM_LOCK 282
- #define FR_KEY_PRINT_SCREEN 283
- #define FR_KEY_PAUSE 284
- #define FR_KEY_F1 290
- #define FR_KEY_F2 291
- #define FR_KEY_F3 292
- #define FR_KEY_F4 293

- #define FR_KEY_F5 294
- #define FR_KEY_F6 295
- #define FR_KEY_F7 296
- #define FR_KEY_F8 297
- #define FR_KEY_F9 298
- #define FR_KEY_F10 299
- #define FR_KEY_F11 300
- #define FR_KEY_F12 301
- #define FR_KEY_F13 302
- #define FR_KEY_F14 303
- #define FR_KEY_F15 304
- #define FR_KEY_F16 305
- #define FR_KEY_F17 306
- #define FR_KEY_F18 307
- #define FR_KEY_F19 308
- #define FR_KEY_F20 309
- #define FR_KEY_F21 310
- #define FR_KEY_F22 311
- #define FR_KEY_F23 312
- #define FR_KEY_F24 313
- #define FR_KEY_F25 314
- #define FR_KEY_KP_0 320
- #define FR_KEY_KP_1 321
- #define FR_KEY_KP_2 322
- #define FR_KEY_KP_3 323
- #define FR_KEY_KP_4 324
- #define FR_KEY_KP_5 325
- #define FR_KEY_KP_6 326
- #define FR_KEY_KP_7 327
- #define FR_KEY_KP_8 328
- #define FR_KEY_KP_9 329
- #define FR_KEY_KP_DECIMAL 330
- #define FR_KEY_KP_DIVIDE 331
- #define FR_KEY_KP_MULTIPLY 332
- #define FR_KEY_KP_SUBTRACT 333
- #define FR_KEY_KP_ADD 334
- #define FR_KEY_KP_ENTER 335
- #define FR_KEY_KP_EQUAL 336
- #define FR_KEY_LEFT_SHIFT 340
- #define FR_KEY_LEFT_CONTROL 341
- #define FR_KEY_LEFT_ALT 342
- #define FR_KEY_RIGHT_SHIFT 344
- #define FR_KEY_RIGHT_CONTROL 345
- #define FR_KEY_RIGHT_ALT 346
- #define FR_KEY_MENU 348
- #define FR_KEY_LEFT_SUPER 343
- #define FR_KEY_LEFT_WINDOWS 343
- #define FR_KEY_RIGHT_SUPER 347
- #define FR_KEY_RIGHT_WINDOWS 347
- #define FR_MOD_SHIFT 0x0001

  *If this bit is set one or more Shift keys were held down.*
- #define FR_MOD_CONTROL 0x0002

  *If this bit is set one or more Control keys were held down.*
- #define FR_MOD_ALT 0x0004

*If this bit is set one or more Alt keys were held down.*

- #define FR_MOD_SUPER 0x0008

  *If this bit is set one or more Super keys were held down.*

- #define FR_MOD_CAPS_LOCK 0x0010

  *If this bit is set the Caps Lock key is enabled.*

- #define FR_MOD_NUM_LOCK 0x0020

  *If this bit is set the Num Lock key is enabled.*

## 10.34.1 Detailed Description

KeyCodes header file.

Contains the key codes for the keyboard.

**See also**

https://www.glfw.org/docs/latest/group__keys.html

## 10.34.2 Macro Definition Documentation

### 10.34.2.1 FR_MOD_ALT

```
#define FR_MOD_ALT 0x0004
```

If this bit is set one or more Alt keys were held down.

If this bit is set one or more Alt keys were held down.

### 10.34.2.2 FR_MOD_CAPS_LOCK

```
#define FR_MOD_CAPS_LOCK 0x0010
```

If this bit is set the Caps Lock key is enabled.

If this bit is set the Caps Lock key is enabled and the FR_LOCK_KEY_MODS input mode is set.

### 10.34.2.3 FR_MOD_CONTROL

```
#define FR_MOD_CONTROL 0x0002
```

If this bit is set one or more Control keys were held down.

If this bit is set one or more Control keys were held down.

### 10.34.2.4 FR_MOD_NUM_LOCK

```
#define FR_MOD_NUM_LOCK 0x0020
```

If this bit is set the Num Lock key is enabled.

If this bit is set the Num Lock key is enabled and the FR_LOCK_KEY_MODS input mode is set.

#### 10.34.2.5 FR_MOD_SHIFT

```
#define FR_MOD_SHIFT 0x0001
```

If this bit is set one or more Shift keys were held down.

If this bit is set one or more Shift keys were held down.

#### 10.34.2.6 FR_MOD_SUPER

```
#define FR_MOD_SUPER 0x0008
```

If this bit is set one or more Super keys were held down.

If this bit is set one or more Super keys were held down.

## 10.35 KeyCodes.h

Go to the documentation of this file.

```
00001 #pragma once
00017 #define FR_KEY_SPACE              32
00018 #define FR_KEY_APOSTROPHE         39  /* ' */
00019 #define FR_KEY_COMMA              44  /* , */
00020 #define FR_KEY_MINUS              45  /* - */
00021 #define FR_KEY_PERIOD             46  /* . */
00022 #define FR_KEY_SLASH              47  /* / */
00023 #define FR_KEY_0                  48
00024 #define FR_KEY_1                  49
00025 #define FR_KEY_2                  50
00026 #define FR_KEY_3                  51
00027 #define FR_KEY_4                  52
00028 #define FR_KEY_5                  53
00029 #define FR_KEY_6                  54
00030 #define FR_KEY_7                  55
00031 #define FR_KEY_8                  56
00032 #define FR_KEY_9                  57
00033 #define FR_KEY_SEMICOLON          59  /* ; */
00034 #define FR_KEY_EQUAL              61  /* = */
00035 #define FR_KEY_A                  65
00036 #define FR_KEY_B                  66
00037 #define FR_KEY_C                  67
00038 #define FR_KEY_D                  68
00039 #define FR_KEY_E                  69
00040 #define FR_KEY_F                  70
00041 #define FR_KEY_G                  71
00042 #define FR_KEY_H                  72
00043 #define FR_KEY_I                  73
00044 #define FR_KEY_J                  74
00045 #define FR_KEY_K                  75
00046 #define FR_KEY_L                  76
00047 #define FR_KEY_M                  77
00048 #define FR_KEY_N                  78
00049 #define FR_KEY_O                  79
00050 #define FR_KEY_P                  80
00051 #define FR_KEY_Q                  81
00052 #define FR_KEY_R                  82
00053 #define FR_KEY_S                  83
00054 #define FR_KEY_T                  84
00055 #define FR_KEY_U                  85
00056 #define FR_KEY_V                  86
00057 #define FR_KEY_W                  87
00058 #define FR_KEY_X                  88
00059 #define FR_KEY_Y                  89
00060 #define FR_KEY_Z                  90
00061 #define FR_KEY_LEFT_BRACKET       91  /* [ */
00062 #define FR_KEY_BACKSLASH          92  /* \ */
00063 #define FR_KEY_RIGHT_BRACKET      93  /* ] */
00064 #define FR_KEY_GRAVE_ACCENT       96  /* ` */
00065 #define FR_KEY_WORLD_1            161 /* non-US #1 */
00066 #define FR_KEY_WORLD_2            162 /* non-US #2 */
00067
```

```
00068 /* Function keys */
00069 #define FR_KEY_ESCAPE            256
00070 #define FR_KEY_ENTER             257
00071 #define FR_KEY_TAB               258
00072 #define FR_KEY_BACKSPACE         259
00073 #define FR_KEY_INSERT            260
00074 #define FR_KEY_DELETE            261
00075 #define FR_KEY_RIGHT             262
00076 #define FR_KEY_LEFT              263
00077 #define FR_KEY_DOWN              264
00078 #define FR_KEY_UP                265
00079 #define FR_KEY_PAGE_UP           266
00080 #define FR_KEY_PAGE_DOWN         267
00081 #define FR_KEY_HOME              268
00082 #define FR_KEY_END               269
00083 #define FR_KEY_CAPS_LOCK         280
00084 #define FR_KEY_SCROLL_LOCK       281
00085 #define FR_KEY_NUM_LOCK          282
00086 #define FR_KEY_PRINT_SCREEN      283
00087 #define FR_KEY_PAUSE             284
00088 #define FR_KEY_F1                290
00089 #define FR_KEY_F2                291
00090 #define FR_KEY_F3                292
00091 #define FR_KEY_F4                293
00092 #define FR_KEY_F5                294
00093 #define FR_KEY_F6                295
00094 #define FR_KEY_F7                296
00095 #define FR_KEY_F8                297
00096 #define FR_KEY_F9                298
00097 #define FR_KEY_F10               299
00098 #define FR_KEY_F11               300
00099 #define FR_KEY_F12               301
00100 #define FR_KEY_F13               302
00101 #define FR_KEY_F14               303
00102 #define FR_KEY_F15               304
00103 #define FR_KEY_F16               305
00104 #define FR_KEY_F17               306
00105 #define FR_KEY_F18               307
00106 #define FR_KEY_F19               308
00107 #define FR_KEY_F20               309
00108 #define FR_KEY_F21               310
00109 #define FR_KEY_F22               311
00110 #define FR_KEY_F23               312
00111 #define FR_KEY_F24               313
00112 #define FR_KEY_F25               314
00113 #define FR_KEY_KP_0              320
00114 #define FR_KEY_KP_1              321
00115 #define FR_KEY_KP_2              322
00116 #define FR_KEY_KP_3              323
00117 #define FR_KEY_KP_4              324
00118 #define FR_KEY_KP_5              325
00119 #define FR_KEY_KP_6              326
00120 #define FR_KEY_KP_7              327
00121 #define FR_KEY_KP_8              328
00122 #define FR_KEY_KP_9              329
00123 #define FR_KEY_KP_DECIMAL        330
00124 #define FR_KEY_KP_DIVIDE         331
00125 #define FR_KEY_KP_MULTIPLY       332
00126 #define FR_KEY_KP_SUBTRACT       333
00127 #define FR_KEY_KP_ADD            334
00128 #define FR_KEY_KP_ENTER          335
00129 #define FR_KEY_KP_EQUAL          336
00130 #define FR_KEY_LEFT_SHIFT        340
00131 #define FR_KEY_LEFT_CONTROL      341
00132 #define FR_KEY_LEFT_ALT          342
00133 #define FR_KEY_RIGHT_SHIFT       344
00134 #define FR_KEY_RIGHT_CONTROL     345
00135 #define FR_KEY_RIGHT_ALT         346
00136 #define FR_KEY_MENU              348
00137
00138 #define FR_KEY_LEFT_SUPER        343
00139 #define FR_KEY_LEFT_WINDOWS      343
00140 #define FR_KEY_RIGHT_SUPER       347
00141 #define FR_KEY_RIGHT_WINDOWS     347
00148 #define FR_MOD_SHIFT           0x0001
00153 #define FR_MOD_CONTROL         0x0002
00158 #define FR_MOD_ALT             0x0004
00163 #define FR_MOD_SUPER           0x0008
00169 #define FR_MOD_CAPS_LOCK       0x0010
00175 #define FR_MOD_NUM_LOCK        0x0020
```

## 10.36 Fracture/src/Fracture/Input/MouseButtonCodes.h File Reference

MouseButtonCodes header file.

### Macros

- #define FR_MOUSE_BUTTON_1 0
- #define FR_MOUSE_BUTTON_2 1
- #define FR_MOUSE_BUTTON_3 2
- #define FR_MOUSE_BUTTON_4 3
- #define FR_MOUSE_BUTTON_5 4
- #define FR_MOUSE_BUTTON_6 5
- #define FR_MOUSE_BUTTON_7 6
- #define FR_MOUSE_BUTTON_8 7
- #define FR_MOUSE_BUTTON_LAST FR_MOUSE_BUTTON_8
- #define FR_MOUSE_BUTTON_LEFT FR_MOUSE_BUTTON_1
- #define FR_MOUSE_BUTTON_RIGHT FR_MOUSE_BUTTON_2
- #define FR_MOUSE_BUTTON_MIDDLE FR_MOUSE_BUTTON_3

### 10.36.1 Detailed Description

MouseButtonCodes header file.

Contains the mouse button codes.

**See also**

https://www.glfw.org/docs/latest/group__buttons.html

## 10.37 MouseButtonCodes.h

Go to the documentation of this file.

```
00001 #pragma once
00017 #define FR_MOUSE_BUTTON_1         0
00018 #define FR_MOUSE_BUTTON_2         1
00019 #define FR_MOUSE_BUTTON_3         2
00020 #define FR_MOUSE_BUTTON_4         3
00021 #define FR_MOUSE_BUTTON_5         4
00022 #define FR_MOUSE_BUTTON_6         5
00023 #define FR_MOUSE_BUTTON_7         6
00024 #define FR_MOUSE_BUTTON_8         7
00025 #define FR_MOUSE_BUTTON_LAST      FR_MOUSE_BUTTON_8
00026 #define FR_MOUSE_BUTTON_LEFT      FR_MOUSE_BUTTON_1
00027 #define FR_MOUSE_BUTTON_RIGHT     FR_MOUSE_BUTTON_2
00028 #define FR_MOUSE_BUTTON_MIDDLE    FR_MOUSE_BUTTON_3
```

## 10.38 Fracture/src/Fracture/Renderer/Buffer.cpp File Reference

```
#include "frpch.h"
#include "Buffer.h"
#include "Renderer.h"
#include "Platform/OpenGL/OpenGLBuffer.h"
```

### Namespaces

- namespace Fracture

## 10.39 Fracture/src/Fracture/Renderer/Buffer.h File Reference

Contains the Buffer class that is used to store the vertex and index buffers.

```
#include <string>
#include <vector>
#include "Fracture/Core/Core.h"
```

### Classes

- struct Fracture::BufferElement

  *The BufferElement struct is used to store the elements of the vertex buffer layout.*
- class Fracture::BufferLayout

  *The BufferLayout class is used to store the layout of the vertex buffer. Each vertex buffer has a buffer layout.*
- class Fracture::VertexBuffer

  *The VertexBuffer class is an abstract class that is used to store the vertex buffer. Each renderer will have its own implementation of the vertex buffer.*
- class Fracture::IndexBuffer

  *The IndexBuffer class is an abstract class that is used to store the index buffer. Each renderer will have its own implementation of the index buffer.*

### Namespaces

- namespace Fracture

### Enumerations

- enum class Fracture::ShaderDataType {
  Fracture::None = 0 , Fracture::Float , Fracture::Float2 , Fracture::Float3 ,
  Fracture::Float4 , Fracture::Mat3 , Fracture::Mat4 , Fracture::Int ,
  Fracture::Int2 , Fracture::Int3 , Fracture::Int4 , Fracture::Bool }

  *The ShaderDataType enum is used to store the data type of the vertex buffer layout.*

### Functions

- static uint32_t Fracture::ShaderDataTypeSize (ShaderDataType type)

  *The ShaderDataTypeSize function is used to return the size of the data type of the vertex buffer layout.*

### 10.39.1 Detailed Description

Contains the Buffer class that is used to store the vertex and index buffers.

**See also**

OpenGLVertexBuffer

OpenGLIndexBuffer

**Author**

Aditya Rajagopal

## 10.40 Buffer.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include <string>
00013 #include <vector>
00014
00015 #include "Fracture/Core/Core.h"
00016
00017 namespace Fracture {
00018
00022     enum class ShaderDataType
00023     {
00024         None = 0, Float, Float2, Float3, Float4, Mat3, Mat4, Int, Int2, Int3, Int4, Bool
00025     };
00026
00030     static uint32_t ShaderDataTypeSize(ShaderDataType type)
00031     {
00032         switch (type)
00033         {
00034             case ShaderDataType::Float2: return       4 * 2;
00035             case ShaderDataType::Float: return     4;
00036             case ShaderDataType::Float3: return       4 * 3;
00037             case ShaderDataType::Float4: return       4 * 4;
00038             case ShaderDataType::Mat3: return       4 * 3 * 3;
00039             case ShaderDataType::Mat4: return       4 * 4 * 4;
00040             case ShaderDataType::Int: return     4;
00041             case ShaderDataType::Int2: return       4 * 2;
00042             case ShaderDataType::Int3: return       4 * 3;
00043             case ShaderDataType::Int4: return       4 * 4;
00044             case ShaderDataType::Bool: return       4;
00045         }
00046
00047         FR_CORE_ASSERT(false, "Unknown ShaderDataType!");
00048         return 0;
00049     }
00050
00056     struct BufferElement
00057     {
00058         std::string Name;
00059         uint32_t Offset;
00060         uint32_t Size;
00061         ShaderDataType Type;
00062         bool Normalized;
00063
00067         BufferElement() = default;
00068
00076         BufferElement(ShaderDataType type, const std::string& name, bool normalized = false)
00077             :Name(name), Type(type), Size(ShaderDataTypeSize(type)), Offset(0), Normalized(normalized)
00078         {}
00079
00083         uint32_t GetElementCount() const
00084         {
00085             switch (Type)
00086             {
00087                 case ShaderDataType::Float2: return       2;
00088                 case ShaderDataType::Float: return     1;
00089                 case ShaderDataType::Float3: return       3;
00090                 case ShaderDataType::Float4: return       4;
00091                 case ShaderDataType::Mat3: return       3 * 3;
```

```
00092                 case ShaderDataType::Mat4: return       4 * 4;
00093                 case ShaderDataType::Int: return     1;
00094                 case ShaderDataType::Int2: return       2;
00095                 case ShaderDataType::Int3: return       3;
00096                 case ShaderDataType::Int4: return       4;
00097                 case ShaderDataType::Bool: return       1;
00098             }
00099 
00100             FR_CORE_ASSERT(false, "Unknown ShaderDataType!");
00101             return 0;
00102         }
00103     };
00104 
00110     class BufferLayout {
00111     public:
00115         BufferLayout() {}
00116 
00122         BufferLayout(const std::initializer_list<BufferElement>& elements):
00123             m_Elements(elements)
00124         {
00125             // When you want to initialise buffer layout with a list of elements you need an
    initializer list of
00126             // initializer list of elements. There are 2 implicit conversions here. First, the
    initializer list of
00127             // elements is converted to a vector of elements. Second, the initializer list of vectors
    is converted
00128             // to a vector of vector of elements. If we want to provide an api like
00129             // BufferLayout layout = { { ShaderDataType::Float3, "a_Position" }, {
    ShaderDataType::Float2, "a_TexCoord" } };
00130             // we need to provide a constructor that takes an initializer list of elements.
00131             CalculateOffsetsAndStride();
00132         }
00133 
00139         std::vector<BufferElement>::iterator begin() { return m_Elements.begin(); }
00140 
00146         std::vector<BufferElement>::iterator end() { return m_Elements.end(); }
00147 
00153         std::vector<BufferElement>::const_iterator begin() const { return m_Elements.begin(); }
00154 
00160         std::vector<BufferElement>::const_iterator end() const { return m_Elements.end(); }
00161 
00165         inline uint32_t GetStride() const { return m_Stride; }
00166 
00172         inline const std::vector<BufferElement>& GetElements() const { return m_Elements; }
00173     private:
00179         void CalculateOffsetsAndStride()
00180         {
00181             uint32_t offset = 0;
00182             m_Stride = 0;
00183             for (auto& element : m_Elements)
00184             {
00185                 element.Offset = offset;
00186                 offset += element.Size;
00187                 m_Stride += element.Size;
00188             }
00189         }
00190     private:
00191         std::vector<BufferElement> m_Elements;
00192         uint32_t m_Stride = 0;
00193 
00194     };
00195 
00202     class VertexBuffer {
00203     public:
00204         virtual ~VertexBuffer() = default;
00205 
00206         virtual void SetData(const void* data, uint32_t size) = 0;
00207 
00208         virtual void SetLayout(const BufferLayout& layout) = 0;
00209         virtual const BufferLayout& GetLayout() const = 0;
00210 
00211         virtual void Bind() const = 0;
00212         virtual void Unbind() const = 0;
00213 
00214 
00227         static Ref<VertexBuffer> Create(float* vertices, uint32_t size);
00228 
00229     };
00230 
00236     class IndexBuffer {
00237     public:
00238         virtual ~IndexBuffer() = default;
00239 
00240         virtual void SetData(const void* data, uint32_t size) = 0;
00241 
00242         virtual void Bind() const = 0;
00243         virtual void Unbind() const = 0;
```

```
00244
00245         virtual uint32_t GetCount() const = 0;
00246
00261         static Ref<IndexBuffer> Create(uint32_t* indices, uint32_t size);
00262     };
00263 }
```

## 10.41 Fracture/src/Fracture/Renderer/GraphicsContext.h File Reference

Contains the GraphicsContext class that is used to create a graphics context for the application per renderer.

### Classes

- class Fracture::GraphicsContext

  *The GraphicsContext class is an abstract class that is used to create a graphics context for the application. Each renderer will have its own implementation of the graphics context.*

### Namespaces

- namespace Fracture

### 10.41.1 Detailed Description

Contains the GraphicsContext class that is used to create a graphics context for the application per renderer.

**See also**

OpenGLContext

Renderer

**Todo** : Add VulkanContext

**Author**

Aditya Rajagopal

## 10.42 GraphicsContext.h

Go to the documentation of this file.

```
00001 #pragma once
00014 namespace Fracture {
00015
00021     class FRACTURE_API GraphicsContext
00022     {
00023     public:
00027         virtual void Init() = 0;
00028
00032         virtual void SwapBuffers() = 0;
00033     };
00034
00035 }
```

## 10.43 Fracture/src/Fracture/Renderer/OrthographicCamera.cpp File Reference

```
#include "frpch.h"
#include "OrthographicCamera.h"
```

### Namespaces

- namespace Fracture

## 10.44 Fracture/src/Fracture/Renderer/OrthographicCamera.h File Reference

Contains the OrthographicCamera class that is provided by the Fracture engine. It is intended that other cameras will be created by the user.

```
#include <Fracture\Components\Component.h>
#include <glm\glm.hpp>
```

### Classes

- class Fracture::OrthographicCamera

### Namespaces

- namespace Fracture

### 10.44.1 Detailed Description

Contains the OrthographicCamera class that is provided by the Fracture engine. It is intended that other cameras will be created by the user.

**Author**

Aditya Rajagopal

## 10.45 OrthographicCamera.h

Go to the documentation of this file.

```
00001 #pragma once
00009 #include <Fracture\Components\Component.h>
00010 #include <glm\glm.hpp>
00011
00012 namespace Fracture {
00013
00014     class OrthographicCamera
00015     {
00016     public:
00028         OrthographicCamera(float left, float right, float bottom, float top);
00029
00042         OrthographicCamera(float left, float right, float bottom, float top, float nearval, float
      farval);
00043
00047         ~OrthographicCamera();
00048
00054         void SetProjection(float left, float right, float bottom, float top, float nearval = -1, float
      farval = 1);
00055
00061         const glm::mat4& GetProjectionMatrix() const { return m_ProjectionMatrix; }
00062
00068         const glm::mat4& GetViewMatrix() { return m_ViewMatrix; }
00069
00075         const glm::mat4& GetViewProjectionMatrix() { return m_ViewProjectionMatrix; }
00076
00082         void SetProjectionMatrix(const glm::mat4& projection) { m_ProjectionMatrix = projection;
      m_ViewProjectionMatrix = m_ProjectionMatrix * m_ViewMatrix; }
00083
00089         void SetViewMatrix(const glm::mat4& view) { m_ViewMatrix = view; m_ViewProjectionMatrix =
      m_ProjectionMatrix * m_ViewMatrix; }
00090     private:
00091         glm::mat4 m_ProjectionMatrix;
00092         glm::mat4 m_ViewMatrix;
00093         glm::mat4 m_ViewProjectionMatrix;
00094     };
00095
00096 }
00097
```

## 10.46 Fracture/src/Fracture/Renderer/OrthographicCamera↩ Controller.cpp File Reference

```
#include "frpch.h"
#include "OrthographicCameraController.h"
#include "Fracture\Core\Core.h"
#include "Fracture/Input/Input.h"
#include "Fracture/Input/KeyCodes.h"
#include "Fracture/Input/MouseButtonCodes.h"
```

### Namespaces

- namespace Fracture

## 10.47 Fracture/src/Fracture/Renderer/OrthographicCameraController.h File Reference

OrthographicCameraController header file containing the OrthographicCameraController class. This class is used to control the orthographic camera.

```
#include <Fracture\Components\Component.h>
#include <Fracture\Renderer\OrthographicCamera.h>
#include <Fracture\Utils\Helpers.h>
#include <Fracture\Events\ApplicationEvent.h>
#include <Fracture\Events\MouseEvent.h>
#include <Fracture\Events\KeyEvent.h>
#include <glm\glm.hpp>
```

**Classes**

- class Fracture::OrthographicCameraController

  *The OrthographicCameraController class is used to control the orthographic camera.*

**Namespaces**

- namespace Fracture

### 10.47.1 Detailed Description

OrthographicCameraController header file containing the OrthographicCameraController class. This class is used to control the orthographic camera.

**See also**

OrthographicCamera

**Author**

Aditya Rajagopal

## 10.48 OrthographicCameraController.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include <Fracture\Components\Component.h>
00013 #include <Fracture\Renderer\OrthographicCamera.h>
00014 #include <Fracture\Utils\Helpers.h>
00015 #include <Fracture\Components\Component.h>
00016
00017 #include <Fracture\Events\ApplicationEvent.h>
00018 #include <Fracture\Events\MouseEvent.h>
00019 #include <Fracture\Events\KeyEvent.h>
00020
00021
00022 #include <glm\glm.hpp>
00023
00024 namespace Fracture {
00025
00031     class OrthographicCameraController
00032     {
00033     public:
00040         OrthographicCameraController(float aspectRatio, float enableRotation = false);
00041
00047         void OnUpdate(Utils::Timestep ts);
00048
00054         void OnEvent(Event& e);
00055
00061         OrthographicCamera& GetCamera() { return m_Camera; }
```

```
00062
00068         const OrthographicCamera& GetCamera() const { return m_Camera; }
00069
00075         float GetAspectRatio() const { return m_AspectRatio; }
00076
00082         const glm::vec3& GetPosition() { return m_CameraTransform.GetPosition(); }
00083
00091         void SetPosition(const glm::vec3& position) { m_CameraTransform.SetPosition(position);
      isChanged = true; }
00092
00100         void Translate(const glm::vec3& translation) { m_CameraTransform.Translate(translation);
      isChanged = true; }
00101
00107         const float& GetRotation() { return m_CameraTransform.GetRotation().z; }
00108
00116         void SetRotation(float rotation) {
00117             if (m_EnableRotation)
00118             {
00119                 m_CameraTransform.SetRotation({ 0.0, 0.0, rotation });
00120                 isChanged = true;
00121             }
00122         }
00123
00131         void Rotate(float rotation) {
00132             if (m_EnableRotation)
00133             {
00134                 m_CameraTransform.Rotate({ 0.0, 0.0, rotation });
00135                 isChanged = true;
00136             }
00137         }
00138
00144         float& GetZoomLevel() { return m_ZoomLevel; }
00145
00153         void Zoom(float zoom);
00154
00162         void SetZoom(float zoom);
00163
00169         const TransformComponent& GetCameraTransform() { return m_CameraTransform; }
00170
00178         void SetCameraTransform(const TransformComponent& transform) { m_CameraTransform = transform;
      isChanged = true; }
00179
00185         void SetCameraZoomSpeed(float speed) { m_cameraZoomSpeed = speed; }
00186
00192         void ToggleRotation(bool enable) { m_EnableRotation = enable; }
00193
00199         void SetMaxZoom(float zoom) { m_MaxZoom = zoom; }
00200
00206         void SetMinZoom(float zoom) { m_MinZoom = zoom; }
00207
00213         float& GetCameraZoomSpeed() { return m_cameraZoomSpeed; }
00214
00220         bool& GetRotationEnabled() { return m_EnableRotation; }
00221
00227         float& GetMaxZoom() { return m_MaxZoom; }
00228
00234         float& GetMinZoom() { return m_MinZoom; }
00235
00236     private:
00247         bool OnMouseScrolledEvent(MouseScrolledEvent& e);
00248
00256         bool OnWindowResizedEvent(WindowResizeEvent& e);
00257
00267         bool OnMouseButtonDownEvent(MouseButtonPressedEvent& e);
00268
00278         bool OnMouseButtonUpEvent(MouseButtonReleasedEvent& e);
00279     private:
00280         Utils::Timestep m_LastFrameTime;
00281
00282         float m_AspectRatio;
00283         float m_ZoomLevel = 1.0f;
00284         bool m_EnableRotation;
00285         TransformComponent m_CameraTransform;
00286
00287         OrthographicCamera m_Camera;
00288
00289         glm::vec2 m_InitialMousePosition = { 0.0f, 0.0f };
00290         glm::vec3 m_InitialCameraPosition = { 0.0f, 0.0f, 0.0f };
00291
00292         float m_MiddleMouseScale = 0.005f;
00293         float m_cameraTranslationSpeed = 1.0f;
00294         float m_cameraRotationSpeed = 1.0f;
00295         float m_cameraZoomSpeed = 40.0f;
00296         float m_MaxZoom = 100.0f;
00297         float m_MinZoom = 0.25f;
00298
00299         bool isChanged = true;
```

```
00300         bool m_canMoveMiddleMouse = false;
00301     };
00302
00303 }
```

## 10.49 Fracture/src/Fracture/Renderer/RenderCommand.cpp File Reference

```
#include "frpch.h"
#include "RenderCommand.h"
#include "Platform/OpenGL/OpenGLRendererAPI.h"
```

### Namespaces

- namespace Fracture

## 10.50 Fracture/src/Fracture/Renderer/RenderCommand.h File Reference

Contains the RenderCommand class that is used to send commands to the renderer.

```
#include "Fracture\Renderer\RendererAPI.h"
```

### Classes

- class Fracture::RenderCommand

  *The RenderCommand class is used to send commands to the renderer. It is a thin wrapper around the RendererAPI class.*

### Namespaces

- namespace Fracture

### 10.50.1 Detailed Description

Contains the RenderCommand class that is used to send commands to the renderer.

The commands are sent to the renderer API that is currently active. The renderer API is created per renderer and inherits from the RendererAPI class that defines the expected interface.

**See also**

RendererAPI

OpenGLRendererAPI

**Author**

Aditya Rajagopal

## 10.51 RenderCommand.h

Go to the documentation of this file.

```
00001 #pragma once
00014 #include "Fracture\Renderer\RendererAPI.h"
00015
00016 namespace Fracture{
00017
00021     class RenderCommand
00022     {
00023     public:
00024
00032         inline static Scope<RendererAPI>& GetRendererAPI()
00033         {
00034             static Scope<RendererAPI> s_RendererAPI = CreateRendererAPI();
00035             return s_RendererAPI;
00036         }
00037
00046         inline static void DrawIndexed(uint32_t indexCount)
00047         {
00048             GetRendererAPI()->DrawIndexed(indexCount);
00049         }
00050
00058         inline static void SetClearColor(const glm::vec4& color)
00059         {
00060             GetRendererAPI()->SetClearColor(color);
00061         }
00062
00075         inline static void SetViewport(uint32_t x, uint32_t y, uint32_t width, uint32_t height)
00076         {
00077             GetRendererAPI()->SetViewport(x, y, width, height);
00078         }
00079
00085         inline static void Clear()
00086         {
00087             GetRendererAPI()->Clear();
00088         }
00089     private:
00097         static Scope<RendererAPI> CreateRendererAPI();
00098     };
00099
00100 }
```

## 10.52 Fracture/src/Fracture/Renderer/Renderer.cpp File Reference

```
#include "frpch.h"
#include "Renderer.h"
```

### Namespaces

- namespace Fracture

## 10.53 Fracture/src/Fracture/Renderer/Renderer.h File Reference

Contains the Renderer class. It provides an interface to render a scene.

```
#include "Fracture/Core/Core.h"
#include "Fracture/Renderer/RenderCommand.h"
#include "Fracture/Renderer/RendererAPI.h"
#include "Fracture\Renderer\VertexArray.h"
#include "Fracture\Renderer\Shader.h"
#include "Fracture\Renderer\OrthographicCamera.h"
#include <glm/glm.hpp>
```

### Classes

- class Fracture::Renderer

  *The Renderer class is used to render a scene. It provides an interface to render a scene.*
- struct Fracture::Renderer::SceneData

  *This is a temporary structure that is used to store all the data that is needed to render the current scene.*

### Namespaces

- namespace Fracture

## 10.53.1 Detailed Description

Contains the Renderer class. It provides an interface to render a scene.

**See also**

Renderer

RendererCommand

RendererAPI

OpenGLRendererAPI

**Todo** : Add VulkanRendererAPI

**Author**

Aditya rajagopal

# 10.54 Renderer.h

Go to the documentation of this file.

```
00001 #pragma once
00016 #include "Fracture/Core/Core.h"
00017 #include "Fracture/Renderer/RenderCommand.h"
00018 #include "Fracture/Renderer/RendererAPI.h"
00019
00020 #include "Fracture\Renderer\VertexArray.h"
00021 #include "Fracture\Renderer\Shader.h"
00022 #include "Fracture\Renderer\OrthographicCamera.h"
00023
00024 #include <glm/glm.hpp>
00025
00026
00027 namespace Fracture
00028 {
00032     class Renderer
00033     {
00034     public:
00038         static void Init();
00039
00047         static void BeginScene(OrthographicCamera& camera);
00048
00052         static void EndScene();
00053
00060         static void OnWindowResize(uint32_t width, uint32_t height);
00061
00062
00076         static void Submit(const Ref<VertexArray>& vertexArray, const Ref<Shader>& shader, const
      glm::mat4& transform);
00077
00083         inline static RendererAPI::API GetAPI() { return RendererAPI::GetAPI(); }
00084     private:
00090         struct SceneData
00091         {
00092             glm::mat4 ViewProjectionMatrix;
00093             uint32_t CurrentBoundShader = 0;
00094         };
00095
00096         static Scope<SceneData> s_SceneData;
00097     };
00098 }
```

## 10.55 Fracture/src/Fracture/Renderer/RendererAPI.cpp File Reference

```
#include "frpch.h"
#include "RendererAPI.h"
```

### Namespaces

- namespace Fracture

## 10.56 Fracture/src/Fracture/Renderer/RendererAPI.h File Reference

Provides an interface for the RendererAPI that needs to be implemented by each renderer.

```
#include <glm/glm.hpp>
#include "VertexArray.h"
```

### Classes

- class Fracture::RendererAPI

  *The RendererAPI class provides an interface for the RendererAPI that needs to be implemented by each renderer.*

### Namespaces

- namespace Fracture

### 10.56.1 Detailed Description

Provides an interface for the RendererAPI that needs to be implemented by each renderer.

**See also**

OpenGLRendererAPI

RenderCommand

**Author**

Aditya Rajagopal

## 10.57 RendererAPI.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include <glm/glm.hpp>
00013
00014 #include "VertexArray.h"
00015
00016 namespace Fracture {
00017
00024     class RendererAPI
00025     {
00026     public:
00030         enum class API
00031         {
00032             None = 0, OpenGL = 1
00033         };
00034     public:
00038         virtual void Init() = 0;
00044         virtual void SetClearColor(const glm::vec4& color) = 0;
00045
00054         virtual void SetViewport(uint32_t x, uint32_t y, uint32_t width, uint32_t height) = 0;
00055
00059         virtual void Clear() = 0;
00060
00068         virtual void DrawIndexed(uint32_t indexCount = 0) = 0;
00069
00073         virtual bool IsInitialized() const = 0;
00074
00082         inline static API GetAPI()
00083         {
00084             static API s_API = API::OpenGL;
00085             return s_API;
00086         }
00087     };
00088
00089 }
00090
```

## 10.58 Fracture/src/Fracture/Renderer/Shader.cpp File Reference

```
#include "frpch.h"
#include "Shader.h"
#include "Renderer.h"
#include "Platform/OpenGL/OpenGLShader.h"
```

### Namespaces

- namespace Fracture

## 10.59 Fracture/src/Fracture/Renderer/Shader.h File Reference

Contains the Shader and ShaderLibrary class.

```
#include <string>
#include <unordered_map>
#include <glm\glm.hpp>
```

### Classes

- class Fracture::Shader

  *The Shader class is an abstract class that is used to create a shader for the application. Each renderer will have its own implementation of the shader.*
- class Fracture::ShaderLibrary

  *The ShaderLibrary class is a singleton class that is used to store all the shaders that are created in the application.*

### Namespaces

- namespace Fracture

### Macros

- #define MAX_SHADER_TYPE_COUNT 2

## 10.59.1 Detailed Description

Contains the Shader and ShaderLibrary class.

**See also**

OpenGLShader

**Author**

Aditya Rajagopal

## 10.59.2 Macro Definition Documentation

### 10.59.2.1 MAX_SHADER_TYPE_COUNT

```
#define MAX_SHADER_TYPE_COUNT 2
```

# 10.60 Shader.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include <string>
00012 #include <unordered_map>
00013
00014 #include <glm\glm.hpp>
00015
00016 namespace Fracture
00017 {
00027     class Shader
00028     {
00029     public:
00030         virtual ~Shader() {};
00031
00033         virtual void Bind() const = 0;
00035         virtual void Unbind() const = 0;
00036
00037         // The following functions set the uniforms in the shader based on the name of the uniform.
00038         virtual void SetInt(const std::string& name, int value) = 0;
00039         virtual void SetInt2(const std::string& name, const glm::ivec2& values) = 0;
00040         virtual void SetInt3(const std::string& name, const glm::ivec3& values) = 0;
00041         virtual void SetInt4(const std::string& name, const glm::ivec4& values) = 0;
00042
00043         virtual void SetFloat(const std::string& name, float value) = 0;
00044         virtual void SetFloat2(const std::string& name, const glm::vec2& values) = 0;
00045         virtual void SetFloat3(const std::string& name, const glm::vec3& values) = 0;
00046         virtual void SetFloat4(const std::string& name, const glm::vec4& values) = 0;
00047
00048         virtual void SetMat3(const std::string& name, const glm::mat3& matrix) = 0;
00049         virtual void SetMat4(const std::string& name, const glm::mat4& matrix) = 0;
00050
00051         virtual void SetBool(const std::string& name, bool value) = 0;
00052
00062         static Ref<Shader> Create(const std::string& name, const std::string& vertex_source, const
      std::string fragment_source);
00063
00075         static Ref<Shader> Create(const std::string& name, const std::string& shaderFilePath);
00076
00089         static Ref<Shader> Create(const std::string& shaderFilePath);
00090
00096         virtual const std::string& GetName() const = 0;
00097
00103         virtual const uint32_t& GetHandle() const = 0;
00104     };
00105
00106
00115     class ShaderLibrary
00116     {
00117     public:
00118
00124         static Scope<ShaderLibrary>& GetInstance()
00125         {
00126             static Scope<ShaderLibrary> instance;
00127             if (instance == nullptr)
00128             {
00129                 instance = CreateScope<ShaderLibrary>();
00130                 instance->InitLibrary();
00131             }
00132             return instance;
00133         }
00134
00141         static void Add(const std::string& name, const Ref<Shader>& shader) {
      GetInstance()->IAdd(name, shader); }
00142
00148         static void Add(const Ref<Shader>& shader) { GetInstance()->IAdd(shader); }
00149
00157         static Ref<Shader> Load(const std::string& filepath) { return GetInstance()->ILoad(filepath);
      }
00158
00167         static Ref<Shader> Load(const std::string& name, const std::string& filepath) { return
      GetInstance()->ILoad(name, filepath); }
00168
00178         static Ref<Shader> Load(const std::string& name, const std::string& vertexSrc, const
      std::string& fragmentSrc) { return GetInstance()->ILoad(name, vertexSrc, fragmentSrc); }
00179
00187         static Ref<Shader> Get(const std::string& name) { return GetInstance()->IGet(name); }
00188     private:
00189         void InitLibrary();
00190         void IAdd(const std::string& name, const Ref<Shader>& shader); // add a shader to the library
00191         void IAdd(const Ref<Shader>& shader); // add a shader to the library
00192         Ref<Shader> ILoad(const std::string& filepath); // load a shader from a file
00193         Ref<Shader> ILoad(const std::string& name, const std::string& filepath); // load a shader from
      a file
```

```
00194         Ref<Shader> ILoad(const std::string& name, const std::string& vertexSrc, const std::string&
      fragmentSrc); // load shaders from string sources
00195         Ref<Shader> IGet(const std::string& name); // get a shader from the library
00196     private:
00197         std::unordered_map<std::string, Ref<Shader>> m_Shaders;
00198     };
00199
00200 }
00201
00202 #define MAX_SHADER_TYPE_COUNT 2
```

## 10.61 Fracture/src/Fracture/Renderer/Texture.cpp File Reference

```
#include "frpch.h"
#include "Texture.h"
#include "Fracture/Renderer/Renderer.h"
#include "Platform/OpenGL/OpenGLTexture.h"
```

### Namespaces

- namespace Fracture

## 10.62 Fracture/src/Fracture/Renderer/Texture.h File Reference

Contains the Texture class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.

```
#include "Fracture\Core\Core.h"
#include <string>
#include <glm\glm.hpp>
```

### Classes

- class Fracture::Texture

  *The Texture class is an abstract class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.*
- class Fracture::Texture2D

  *The Texture2D class is an abstract class that is used to store references to 2D textures needed for rendering. Each renderer will have its own implementation of the texture class.*

### Namespaces

- namespace Fracture

### 10.62.1 Detailed Description

Contains the Texture class that is used to store references to textures needed for rendering. Each renderer will have its own implementation of the texture class.

**See also**

OpenGLTexture

Renderer

**Author**

Aditya Rajagopal

## 10.63 Texture.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include "Fracture\Core\Core.h"
00013
00014 #include <string>
00015 #include <glm\glm.hpp>
00016
00017 namespace Fracture {
00018
00022     class Texture
00023     {
00024     public:
00025         virtual ~Texture() = default;
00026
00032         virtual uint32_t GetWidth() const = 0;
00033
00039         virtual uint32_t GetHeight() const = 0;
00040
00046         virtual uint32_t GetHandle() const = 0;
00047
00053         virtual void Bind(uint32_t slot = 0) const = 0;
00054     };
00055
00063     class Texture2D : public Texture
00064     {
00065     public:
00079         static Ref<Texture2D> Create(uint32_t width, uint32_t height, glm::vec4 color);
00080
00092         static Ref<Texture2D> Create(const std::string& path);
00093     };
00094
00095 }
00096
```

## 10.64 Fracture/src/Fracture/Renderer/VertexArray.cpp File Reference

```
#include "frpch.h"
#include "VertexArray.h"
#include "Fracture/Renderer/Renderer.h"
#include "Platform/OpenGL/OpenGLVertexArray.h"
```

**Namespaces**

- namespace Fracture

## 10.65 Fracture/src/Fracture/Renderer/VertexArray.h File Reference

Contains the VertexArray class that is used to create a VertexArray object.

```
#include "Fracture/Renderer/Buffer.h"
```

### Classes

- class Fracture::VertexArray

  *The VertexArray class is an abstract class that is used to create a VertexArray object. Each renderer will have its own implementation of the VertexArray class.*

### Namespaces

- namespace Fracture

### 10.65.1 Detailed Description

Contains the VertexArray class that is used to create a VertexArray object.

**See also**

VertexBuffer

IndexBuffer

**Author**

Aditya Rajagopal

## 10.66 VertexArray.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include "Fracture/Renderer/Buffer.h"
00013
00014 namespace Fracture {
00015
00021     class VertexArray {
00022     public:
00023         virtual ~VertexArray() {};
00024
00030         virtual void AddVertexBuffer(const Ref<VertexBuffer>& vertexBuffer) = 0;
00031
00037         virtual void SetIndexBuffer(const Ref<IndexBuffer>& indexBuffer) = 0;
00038
00042         virtual void Bind() const = 0;
00043
00047         virtual void Unbind() const = 0;
00048
00054         virtual const Ref<IndexBuffer>& GetIndexBuffer() const = 0;
00055
00061         virtual const std::vector<Ref<VertexBuffer>>& GetVertexBuffers() const = 0;
00062
00072         static Ref<VertexArray> Create();
00073
00074     };
00075 }
```

## 10.67 Fracture/src/Fracture/Utils/Helpers.cpp File Reference

```
#include "frpch.h"
#include "Helpers.h"
```

### Namespaces

- namespace Fracture
- namespace Fracture::Utils

### Functions

- std::string Fracture::Utils::ReadFile (const std::string &filePath)

  *Reads a file and returns the contents as a string.*

## 10.68 Fracture/src/Fracture/Utils/Helpers.h File Reference

Contains helper functions for the engine to be used internally and by the client application.

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

### Classes

- struct Fracture::Utils::Timestep

  *data structure used to store time in seconds*

### Namespaces

- namespace Fracture
- namespace Fracture::Utils

### Functions

- std::string Fracture::Utils::ReadFile (const std::string &filePath)

  *Reads a file and returns the contents as a string.*

### 10.68.1 Detailed Description

Contains helper functions for the engine to be used internally and by the client application.

**Author**

Aditya Rajagaopl

## 10.69 Helpers.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include <fstream>
00012 #include <string>
00013
00014 namespace  Fracture {
00015
00016     namespace Utils {
00017
00025         std::string ReadFile(const std::string& filePath);
00026
00027
00031         struct Timestep
00032         {
00038             Timestep(float time = 0)
00039                 :m_Time(time)
00040             {
00041             }
00042
00048             operator float() const { return m_Time; }
00049
00055             float GetSeconds() const { return m_Time; }
00056
00062             float GetMilliseconds() const { return m_Time * 1000.0f; }
00063
00069             float GetMicroseconds() const { return m_Time * 1000.0f * 1000.0f; }
00070         private:
00071             float m_Time;
00072         };
00073
00074     }
00075
00076 }
```

## 10.70 Fracture/src/Fracture/Utils/Instrumentation.h File Reference

```
#include <string>
#include <chrono>
#include <algorithm>
#include <fstream>
#include <thread>
```

### Classes

- struct Fracture::Utils::ProfileResult
- struct Fracture::Utils::InstrumentationSession
- class Fracture::Utils::Instrumentor
- class Fracture::Utils::InstrumentationTimer

### Namespaces

- namespace Fracture
- namespace Fracture::Utils

### Macros

- #define FR_PROFILE_SCOPE(name)
- #define FR_PROFILE_FUNCTION()
- #define FR_BEGIN_PROFILE_SESSION(name, filepath)
- #define FR_END_PROFILE_SESSION()

### 10.70.1 Macro Definition Documentation

#### 10.70.1.1 FR_BEGIN_PROFILE_SESSION

```
#define FR_BEGIN_PROFILE_SESSION(
              name,
              filepath )
```

#### 10.70.1.2 FR_END_PROFILE_SESSION

```
#define FR_END_PROFILE_SESSION( )
```

#### 10.70.1.3 FR_PROFILE_FUNCTION

```
#define FR_PROFILE_FUNCTION( )
```

#### 10.70.1.4 FR_PROFILE_SCOPE

```
#define FR_PROFILE_SCOPE(
              name )
```

## 10.71 Instrumentation.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 //
00004 // Basic instrumentation profiler by Cherno
00005
00006 // Usage: include this header file somewhere in your code (eg. precompiled header), and then use like:
00007 //
00008 // Instrumentor::Get().BeginSession("Session Name");        // Begin session
00009 // {
00010 //     InstrumentationTimer timer("Profiled Scope Name");   // Place code like this in scopes you'd
      like to include in profiling
00011 //     // Code
00012 // }
00013 // Instrumentor::Get().EndSession();                        // End Session
00014 //
00015
00016 #include <string>
00017 #include <chrono>
00018 #include <algorithm>
00019 #include <fstream>
00020
00021 #include <thread>
00022
00023 namespace Fracture {
00024     namespace Utils {
00025         struct ProfileResult
00026         {
00027             std::string Name;
00028             long long Start, End;
00029             uint32_t ThreadID;
00030         };
00031
00032         struct InstrumentationSession
00033         {
00034             std::string Name;
00035         };
00036
00037         class Instrumentor
```

```
00038         {
00039         private:
00040             InstrumentationSession* m_CurrentSession;
00041             std::ofstream m_OutputStream;
00042             int m_ProfileCount;
00043         public:
00044             Instrumentor()
00045                 :m_CurrentSession(nullptr), m_ProfileCount(0)
00046             {
00047             }
00048 
00049             void BeginSession(const std::string& name, const std::string& filepath =
      "../Logs/results.json")
00050             {
00051                 m_OutputStream.open(filepath);
00052                 WriteHeader();
00053                 m_CurrentSession = new InstrumentationSession{ name };
00054             }
00055 
00056             void EndSession()
00057             {
00058                 WriteFooter();
00059                 m_OutputStream.close();
00060                 delete m_CurrentSession;
00061                 m_CurrentSession = nullptr;
00062                 m_ProfileCount = 0;
00063             }
00064 
00065             void WriteProfile(const ProfileResult& result)
00066             {
00067                 if (m_ProfileCount++ > 0)
00068                     m_OutputStream << ",";
00069 
00070                 std::string name = result.Name;
00071                 std::replace(name.begin(), name.end(), '"', '\'');
00072 
00073                 m_OutputStream << "{";
00074                 m_OutputStream << "\"cat\":\"function\",";
00075                 m_OutputStream << "\"dur\":" << (result.End - result.Start) << ',';
00076                 m_OutputStream << "\"name\":\"" << name << "\",";
00077                 m_OutputStream << "\"ph\":\"X\",";
00078                 m_OutputStream << "\"pid\":0,";
00079                 m_OutputStream << "\"tid\":" << result.ThreadID << ",";
00080                 m_OutputStream << "\"ts\":" << result.Start;
00081                 m_OutputStream << "}";
00082 
00083                 m_OutputStream.flush();
00084             }
00085 
00086             void WriteHeader()
00087             {
00088                 m_OutputStream << "{\"otherData\": {},\"traceEvents\":[";
00089                 m_OutputStream.flush();
00090             }
00091 
00092             void WriteFooter()
00093             {
00094                 m_OutputStream << "]}";
00095                 m_OutputStream.flush();
00096             }
00097 
00098             static Instrumentor& Get()
00099             {
00100                 static Instrumentor instance;
00101                 return instance;
00102             }
00103         };
00104 
00105         class InstrumentationTimer
00106         {
00107         public:
00108             InstrumentationTimer(const char* name)
00109                 :m_Name(name), m_Stopped(false)
00110             {
00111                 m_StartTimepoint = std::chrono::high_resolution_clock::now();
00112             }
00113 
00114             ~InstrumentationTimer()
00115             {
00116                 if (!m_Stopped)
00117                     Stop();
00118             }
00119 
00120             void Stop()
00121             {
00122                 auto endTimepoint = std::chrono::high_resolution_clock::now();
00123 
```

```
00124                 long long start =
      std::chrono::time_point_cast<std::chrono::microseconds>(m_StartTimepoint).time_since_epoch().count();
00125                 long long end =
      std::chrono::time_point_cast<std::chrono::microseconds>(endTimepoint).time_since_epoch().count();
00126
00127                 uint32_t threadID = std::hash<std::thread::id>{}(std::this_thread::get_id());
00128                 Instrumentor::Get().WriteProfile({ m_Name, start, end, threadID });
00129
00130                 m_Stopped = true;
00131             }
00132         private:
00133             const char* m_Name;
00134             std::chrono::time_point<std::chrono::high_resolution_clock> m_StartTimepoint;
00135             bool m_Stopped;
00136         };
00137     }
00138 }
00139
00140 //#ifndef FR_DIST
00141 //#define FR_PROFILE_SCOPE(name) ::Fracture::Utils::InstrumentationTimer timer##__LINE__(name)
00142 //#define FR_PROFILE_FUNCTION() FR_PROFILE_SCOPE(__FUNCSIG__)
00143 //#define FR_BEGIN_PROFILE_SESSION(name, filepath)
      ::Fracture::Utils::Instrumentor::Get().BeginSession(name, filepath)
00144 //#define FR_END_PROFILE_SESSION() ::Fracture::Utils::Instrumentor::Get().EndSession()
00145 //#else
00146 #define FR_PROFILE_SCOPE(name)
00147 #define FR_PROFILE_FUNCTION()
00148 #define FR_BEGIN_PROFILE_SESSION(name, filepath)
00149 #define FR_END_PROFILE_SESSION()
00150 //#endif
```

## 10.72 Fracture/src/Fracture/Utils/Log.cpp File Reference

```
#include "frpch.h"
#include <spdlog/sinks/stdout_color_sinks.h>
```

### Namespaces

- namespace Fracture

## 10.73 Fracture/src/Fracture/Utils/Log.h File Reference

Contains the Log class that is used to log messages to the console.

```
#include "Fracture\Core\Core.h"
#include "spdlog/spdlog.h"
#include "spdlog/fmt/ostr.h"
```

### Classes

- class Fracture::Log

  *The Log class is used to log messages to the console.*

### Namespaces

- namespace Fracture

**Macros**

- #define FR_CORE_CRITICAL(...) ::Fracture::Log::GetCoreLogger()->critical(__VA_ARGS__)
- #define FR_CORE_ERROR(...) ::Fracture::Log::GetCoreLogger()->error(__VA_ARGS__)
- #define FR_CORE_WARN(...) ::Fracture::Log::GetCoreLogger()->warn(__VA_ARGS__)
- #define FR_CORE_INFO(...) ::Fracture::Log::GetCoreLogger()->info(__VA_ARGS__)
- #define FR_CORE_TRACE(...) ::Fracture::Log::GetCoreLogger()->trace(__VA_ARGS__)
- #define FR_CRITICAL(...) ::Fracture::Log::GetClientLogger()->critical(__VA_ARGS__)
- #define FR_ERROR(...) ::Fracture::Log::GetClientLogger()->error(__VA_ARGS__)
- #define FR_WARN(...) ::Fracture::Log::GetClientLogger()->warn(__VA_ARGS__)
- #define FR_INFO(...) ::Fracture::Log::GetClientLogger()->info(__VA_ARGS__)
- #define FR_TRACE(...) ::Fracture::Log::GetClientLogger()->trace(__VA_ARGS__)

## 10.73.1 Detailed Description

Contains the Log class that is used to log messages to the console.

**See also**

spdlog

**Todo** : Add logging to file

**Author**

Aditya Rajagopal

## 10.73.2 Macro Definition Documentation

### 10.73.2.1 FR_CORE_CRITICAL

```
#define FR_CORE_CRITICAL(
              ... ) ::Fracture::Log::GetCoreLogger()->critical(__VA_ARGS__)
```

### 10.73.2.2 FR_CORE_ERROR

```
#define FR_CORE_ERROR(
              ... ) ::Fracture::Log::GetCoreLogger()->error(__VA_ARGS__)
```

### 10.73.2.3 FR_CORE_INFO

```
#define FR_CORE_INFO(
              ... ) ::Fracture::Log::GetCoreLogger()->info(__VA_ARGS__)
```

### 10.73.2.4 FR_CORE_TRACE

```
#define FR_CORE_TRACE(
              ... ) ::Fracture::Log::GetCoreLogger()->trace(__VA_ARGS__)
```

#### 10.73.2.5 FR_CORE_WARN

```
#define FR_CORE_WARN(
              ...  ) ::Fracture::Log::GetCoreLogger()->warn(__VA_ARGS__)
```

#### 10.73.2.6 FR_CRITICAL

```
#define FR_CRITICAL(
              ...  ) ::Fracture::Log::GetClientLogger()->critical(__VA_ARGS__)
```

#### 10.73.2.7 FR_ERROR

```
#define FR_ERROR(
              ...  ) ::Fracture::Log::GetClientLogger()->error(__VA_ARGS__)
```

#### 10.73.2.8 FR_INFO

```
#define FR_INFO(
              ...  ) ::Fracture::Log::GetClientLogger()->info(__VA_ARGS__)
```

#### 10.73.2.9 FR_TRACE

```
#define FR_TRACE(
              ...  ) ::Fracture::Log::GetClientLogger()->trace(__VA_ARGS__)
```

#### 10.73.2.10 FR_WARN

```
#define FR_WARN(
              ...  ) ::Fracture::Log::GetClientLogger()->warn(__VA_ARGS__)
```

## 10.74 Log.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "Fracture\Core\Core.h"
00014 #include "spdlog/spdlog.h"
00015 #include "spdlog/fmt/ostr.h"
00016
00017 namespace Fracture {
00018
00022     class FRACTURE_API Log
00023     {
00024     public:
00028         static void Init();
00029
00035         inline static Ref<spdlog::logger>& GetCoreLogger() { return s_CoreLogger; }
00036
00042         inline static Ref<spdlog::logger>& GetClientLogger() { return s_ClientLogger; }
00043     private:
00044         static Ref<spdlog::logger> s_ClientLogger;
00045         static Ref<spdlog::logger> s_CoreLogger;
00046     };
00047 }
```

```
00048
00049 // Internal engine logging macros
00050 // the ::Fracture:: ensures it is accessed from the global namespace
00051 #define FR_CORE_CRITICAL(...)    ::Fracture::Log::GetCoreLogger()->critical(__VA_ARGS__)
00052 #define FR_CORE_ERROR(...)       ::Fracture::Log::GetCoreLogger()->error(__VA_ARGS__)
00053 #define FR_CORE_WARN(...)        ::Fracture::Log::GetCoreLogger()->warn(__VA_ARGS__)
00054 #define FR_CORE_INFO(...)        ::Fracture::Log::GetCoreLogger()->info(__VA_ARGS__)
00055 #define FR_CORE_TRACE(...)       ::Fracture::Log::GetCoreLogger()->trace(__VA_ARGS__)
00056
00057 // Client logging macros
00058 #define FR_CRITICAL(...)         ::Fracture::Log::GetClientLogger()->critical(__VA_ARGS__)
00059 #define FR_ERROR(...)            ::Fracture::Log::GetClientLogger()->error(__VA_ARGS__)
00060 #define FR_WARN(...)             ::Fracture::Log::GetClientLogger()->warn(__VA_ARGS__)
00061 #define FR_INFO(...)             ::Fracture::Log::GetClientLogger()->info(__VA_ARGS__)
00062 #define FR_TRACE(...)            ::Fracture::Log::GetClientLogger()->trace(__VA_ARGS__)
```

## 10.75 Fracture/src/frpch.cpp File Reference

```
#include "frpch.h"
```

## 10.76 Fracture/src/frpch.h File Reference

Precompiled header file for Fracture engine.

```
#include <memory>
#include <utility>
#include <algorithm>
#include <functional>
#include <thread>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <chrono>
#include <sstream>
#include <string>
#include <vector>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include "Fracture\Utils\Log.h"
#include "Fracture\Utils\Instrumentation.h"
#include "Fracture\Utils\Helpers.h"
```

### 10.76.1 Detailed Description

Precompiled header file for Fracture engine.

**See also**

https://docs.microsoft.com/en-us/cpp/build/reference/creating-precompiled-header-fi

**Author**

Aditya Rajagopal

## 10.77 frpch.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include <memory>
00013 #include <utility>
00014 #include <algorithm>
00015 #include <functional>
00016 #include <thread>
00017
00018 #include <iostream>
00019 #include <fstream>
00020 #include <filesystem>
00021 #include <chrono>
00022 #include <sstream>
00023 #include <string>
00024 #include <vector>
00025 #include <set>
00026 #include <unordered_map>
00027 #include <unordered_set>
00028
00029
00030 #ifdef FR_PLATFORM_WINDOWS
00031     #include <Windows.h>
00032 #endif
00033
00034 #include "Fracture\Utils\Log.h"
00035 #include "Fracture\Utils\Instrumentation.h"
00036 #include "Fracture\Utils\Helpers.h"
```

## 10.78 Fracture/src/Platform/OpenGL/OpenGLBuffer.cpp File Reference

```
#include "frpch.h"
#include "OpenGLBuffer.h"
#include <glad/glad.h>
```

### Namespaces

- namespace Fracture

## 10.79 Fracture/src/Platform/OpenGL/OpenGLBuffer.h File Reference

Contains the OpenGLBuffer class that implements the VertexBuffer and IndexBuffer classes for OpenGL.

```
#include "Fracture/Renderer/Buffer.h"
```

### Classes

- class Fracture::OpenGLVertexBuffer

  *The OpenGLVertexBuffer class is an implementation of the VertexBuffer class for OpenGL.*
- class Fracture::OpenGLIndexBuffer

### Namespaces

- namespace Fracture

### 10.79.1 Detailed Description

Contains the OpenGLBuffer class that implements the VertexBuffer and IndexBuffer classes for OpenGL.

**See also**

VertexBuffer

IndexBuffer

**Author**

Aditya Rajagopal

## 10.80 OpenGLBuffer.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include "Fracture/Renderer/Buffer.h"
00013
00014
00015 namespace Fracture {
00016
00022     class OpenGLVertexBuffer : public VertexBuffer
00023     {
00024     public:
00033         OpenGLVertexBuffer(float* vertices, uint32_t size);
00034
00038         ~OpenGLVertexBuffer();
00039
00051         virtual void SetData(const void* data, uint32_t size) override;
00052
00060         virtual void SetLayout(const BufferLayout& layout) override { m_Layout = layout; }
00061
00067         virtual const BufferLayout& GetLayout() const override { return m_Layout; }
00068
00072         virtual void Bind() const override;
00073
00077         virtual void Unbind() const override;
00078
00079     private:
00080         uint32_t m_RendererID;
00081         BufferLayout m_Layout;
00082     };
00083
00084
00085     class OpenGLIndexBuffer : public IndexBuffer
00086     {
00087     public:
00088         OpenGLIndexBuffer(uint32_t* indices, uint32_t count);
00089         ~OpenGLIndexBuffer();
00090
00091         virtual void SetData(const void* data, uint32_t size) override;
00092
00093         virtual void Bind() const override;
00094         virtual void Unbind() const override;
00095
00096         virtual uint32_t GetCount() const override { return m_Count; }
00097
00098     private:
00099         uint32_t m_RendererID;
00100         uint32_t m_Count;
00101     };
00102
00103 }
```

## 10.81 Fracture/src/Platform/OpenGL/OpenGLContext.cpp File Reference

```
#include "frpch.h"
#include "OpenGLContext.h"
#include <GLFW/glfw3.h>
#include <glad/glad.h>
```

### Namespaces

- namespace Fracture

## 10.82 Fracture/src/Platform/OpenGL/OpenGLContext.h File Reference

Contains the OpenGLContext class that is used to create a graphics context for the OpenGL renderer.

```
#include "Fracture\Renderer\GraphicsContext.h"
```

### Classes

- class Fracture::OpenGLContext

  *The OpenGLContext class is an implementation of the GraphicsContext class for the OpenGL renderer.*

### Namespaces

- namespace Fracture

### 10.82.1 Detailed Description

Contains the OpenGLContext class that is used to create a graphics context for the OpenGL renderer.

**See also**

GraphicsContext

**Author**

Aditya Rajagopal

## 10.83 OpenGLContext.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "Fracture\Renderer\GraphicsContext.h"
00014
00015 struct GLFWwindow; // Forward declaration
00016
00017
00018 namespace Fracture {
00019
00025     class OpenGLContext : public GraphicsContext
00026     {
00027     public:
00037         OpenGLContext(GLFWwindow* windowHandle);
00038
00046         virtual void Init() override;
00047
00053         virtual void SwapBuffers() override;
00054     private:
00055         GLFWwindow* m_WindowHandle;
00056     };
00057
00058 }
```

## 10.84 Fracture/src/Platform/OpenGL/OpenGLRendererAPI.cpp File Reference

```
#include "frpch.h"
#include "OpenGLRendererAPI.h"
#include <glad/glad.h>
```

### Namespaces

- namespace Fracture

## 10.85 Fracture/src/Platform/OpenGL/OpenGLRendererAPI.h File Reference

Implementation of the RendererAPI for OpenGL.

```
#include "Fracture/Renderer/RendererAPI.h"
```

### Classes

- class Fracture::OpenGLRendererAPI

  *Implementation of the RendererAPI for OpenGL.*

### Namespaces

- namespace Fracture

### 10.85.1 Detailed Description

Implementation of the RendererAPI for OpenGL.

**See also**

RendererAPI

**Author**

Aditya Rajagopal

## 10.86 OpenGLRendererAPI.h

Go to the documentation of this file.

```
00001 #pragma once
00012 #include "Fracture/Renderer/RendererAPI.h"
00013
00014 namespace Fracture {
00015
00019     class OpenGLRendererAPI : public RendererAPI
00020     {
00021     public:
00031         OpenGLRendererAPI();
00032         ~OpenGLRendererAPI();
00033
00041         virtual void Init() override;
00042
00050         virtual void SetClearColor(const glm::vec4& color) override;
00051
00062         virtual void SetViewport(uint32_t x, uint32_t y, uint32_t width, uint32_t height) override;
00063
00069         virtual void Clear() override;
00070
00079         virtual void DrawIndexed(uint32_t indexCount = 0) override;
00080
00086         virtual bool IsInitialized() const override { return m_IsInitialized; }
00087     private:
00088         bool m_IsInitialized = false;
00089     };
00090
00091 }
```

## 10.87 Fracture/src/Platform/OpenGL/OpenGLShader.cpp File Reference

```
#include "frpch.h"
#include "OpenGLShader.h"
#include "Fracture\Renderer\Shader.h"
#include "glm\gtc\type_ptr.hpp"
```

### Namespaces

- namespace Fracture

### Functions

- static GLenum Fracture::ShaderTypeFromString (const std::string &type)
- static std::string Fracture::ShaderTypeToString (GLenum type)
- static void Fracture::CompileShaders (GLuint handle, const std::string &source, GLenum type)

## 10.88 Fracture/src/Platform/OpenGL/OpenGLShader.h File Reference

Contains the OpenGL implementation of the Shader class.

```
#include <string.h>
#include "Fracture/Core/Core.h"
#include "Fracture/Renderer/Shader.h"
#include "glad/glad.h"
```

### Classes

- class Fracture::OpenGLShader

  *The OpenGLShader class is an implementation of the Shader class. It is used to create a shader program for the OpenGL renderer.*

### Namespaces

- namespace Fracture

## 10.88.1 Detailed Description

Contains the OpenGL implementation of the Shader class.

**See also**

Shader

**Author**

Aditya Rajagopal

# 10.89 OpenGLShader.h

Go to the documentation of this file.

```
00001 #pragma once
00010 #include <string.h>
00011
00012 #include "Fracture/Core/Core.h"
00013 #include "Fracture/Renderer/Shader.h"
00014
00015 #include "glad/glad.h"
00016
00017
00018 namespace Fracture
00019 {
00023     class OpenGLShader : public Shader
00024     {
00025     public:
00035         OpenGLShader(const std::string& name, const std::string& vertex_source, const std::string
    fragment_source);
00036
00043         OpenGLShader(const std::string& name, const std::string& shaderFilePath);
00044
00048         ~OpenGLShader();
00049
00050         virtual void Bind() const override;
00051         virtual void Unbind() const override;
00052
00053         virtual void SetInt(const std::string& name, int value) override;
00054         virtual void SetInt2(const std::string& name, const glm::ivec2& values) override;
00055         virtual void SetInt3(const std::string& name, const glm::ivec3& values) override;
00056         virtual void SetInt4(const std::string& name, const glm::ivec4& values) override;
00057
00058
00059         virtual void SetFloat(const std::string& name, float value) override;
00060         virtual void SetFloat2(const std::string& name, const glm::vec2& values) override;
00061         virtual void SetFloat3(const std::string& name, const glm::vec3& values) override;
00062         virtual void SetFloat4(const std::string& name, const glm::vec4& values) override;
00063
00064         virtual void SetMat3(const std::string& name, const glm::mat3& matrix) override;
00065         virtual void SetMat4(const std::string& name, const glm::mat4& matrix) override;
00066         virtual void SetBool(const std::string& name, bool value) override;
00067
00068         void UploadUniformInt(const std::string& name, int value);
```

```
00069         void UploadUniformInt2(const std::string& name, const glm::ivec2& values);
00070         void UploadUniformInt3(const std::string& name, const glm::ivec3& values);
00071         void UploadUniformInt4(const std::string& name, const glm::ivec4& values);
00072 
00073         void UploadUniformFloat(const std::string& name, float value);
00074         void UploadUniformFloat2(const std::string& name, const glm::vec2& values);
00075         void UploadUniformFloat3(const std::string& name, const glm::vec3& values);
00076         void UploadUniformFloat4(const std::string& name, const glm::vec4& values);
00077 
00078         void UploadUniformMat3(const std::string& name, const glm::mat3& matrix);
00079         void UploadUniformMat4(const std::string& name, const glm::mat4& matrix);
00080 
00081         void UploadUniformBool(const std::string& name, bool value);
00082 
00083         virtual const std::string& GetName() const override { return m_Name; }
00084         virtual const uint32_t& GetHandle() const override { return m_RendererID; }
00085     private:
00096         int32_t GetUniformLocation(const std::string& name);
00097 
00105         std::unordered_map<GLenum, std::string> PreProcess(const std::string& source);
00106 
00112         void Compile(const std::unordered_map<GLenum, std::string>& shaderSources);
00113     private:
00114         uint32_t m_RendererID;
00115         std::string m_Name;
00116         std::unordered_map<std::string, int32_t> m_UniformLocationCache;
00117     };
00118 
00119 }
```

## 10.90 Fracture/src/Platform/OpenGL/OpenGLTexture.cpp File Reference

```
#include "frpch.h"
#include "OpenGLTexture.h"
#include <glad/glad.h>
#include <stb_image.h>
```

### Namespaces

- namespace Fracture

## 10.91 Fracture/src/Platform/OpenGL/OpenGLTexture.h File Reference

contains the OpenGL implementation of the Texture class

```
#include "Fracture/Core/Core.h"
#include "Fracture/Renderer/Texture.h"
```

### Classes

- class Fracture::OpenGLTexture2D

  *OpenGL implementation of the Texture2D class.*

### Namespaces

- namespace Fracture

### 10.91.1 Detailed Description

contains the OpenGL implementation of the Texture class

**See also**

Texture

**Author**

Aditya Rajagopal

## 10.92 OpenGLTexture.h

Go to the documentation of this file.

```
00001 #pragma once
00010 #include "Fracture/Core/Core.h"
00011 #include "Fracture/Renderer/Texture.h"
00012
00013 namespace Fracture
00014 {
00018     class OpenGLTexture2D : public Texture2D
00019     {
00020     public:
00028         OpenGLTexture2D(const std::string& path);
00029
00039         OpenGLTexture2D(uint32_t width, uint32_t height, glm::vec4 color);
00040
00041         virtual ~OpenGLTexture2D();
00042
00048         virtual uint32_t GetWidth() const override { return m_Width; }
00049
00055         virtual uint32_t GetHeight() const override { return m_Height; }
00056
00062         virtual uint32_t GetHandle() const override { return m_RendererID; }
00063
00069         virtual void Bind(uint32_t slot = 0) const override;
00070     private:
00071         std::string m_Path;
00072         uint32_t m_Width, m_Height;
00073         uint32_t m_RendererID;
00074     };
00075
00076 }
```

## 10.93 Fracture/src/Platform/OpenGL/OpenGLVertexArray.cpp File Reference

```
#include "frpch.h"
#include "OpenGLVertexArray.h"
#include <glad/glad.h>
```

**Namespaces**

- namespace Fracture

**Functions**

- static GLenum Fracture::ShaderDataTypeToOpenGLBaseType (ShaderDataType type)

# 10.94 Fracture/src/Platform/OpenGL/OpenGLVertexArray.h File Reference

VertexArray implementation for OpenGL.

```
#include "Fracture\Renderer\VertexArray.h"
```

### Classes

- class Fracture::OpenGLVertexArray

  *OpenGLVertexArray class that implements the VertexArray class for OpenGL.*

### Namespaces

- namespace Fracture

## 10.94.1 Detailed Description

VertexArray implementation for OpenGL.

**See also**

VertexArray

**Author**

Aditya Rajagopal

# 10.95 OpenGLVertexArray.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include "Fracture\Renderer\VertexArray.h"
00012
00013
00014 namespace Fracture {
00015
00021     class OpenGLVertexArray: public VertexArray
00022     {
00023     public:
00027         OpenGLVertexArray();
00028
00032         ~OpenGLVertexArray();
00033
00043         virtual void AddVertexBuffer(const Ref<VertexBuffer>& vertexBuffer) override;
00044
00050         virtual void SetIndexBuffer(const Ref<IndexBuffer>& indexBuffer) override;
00051
00057         virtual const Ref<IndexBuffer>& GetIndexBuffer() const override { return m_IndexBuffer; }
00058
00064         virtual const std::vector<Ref<VertexBuffer»& GetVertexBuffers() const override { return
      m_VertexBuffers; }
00065
00069         virtual void Bind() const override;
00070
00074         virtual void Unbind() const override;
00075     private:
00076         uint32_t m_RendererID;
00077         std::vector<Ref<VertexBuffer» m_VertexBuffers;
00078         Ref<IndexBuffer> m_IndexBuffer;
00079         uint32_t m_VertexBufferIndex = 0;
00080     };
00081
00082 }
```

## 10.96 Fracture/src/Platform/Windows/WindowsInput.cpp File Reference

```
#include "frpch.h"
#include "WindowsInput.h"
#include "Fracture\Core\Application.h"
#include <GLFW\glfw3.h>
```

### Namespaces

- namespace Fracture

## 10.97 Fracture/src/Platform/Windows/WindowsInput.h File Reference

Contains the Windows implementation of the Input class.

```
#include "Fracture\Input\Input.h"
```

### Classes

- class Fracture::WindowsInput

  *The Windows implementation of the Input class.*

### Namespaces

- namespace Fracture

### 10.97.1 Detailed Description

Contains the Windows implementation of the Input class.

**See also**

Input

**Author**

Aditya Rajagopal

## 10.98 WindowsInput.h

Go to the documentation of this file.

```
00001 #pragma once
00011 #include "Fracture\Input\Input.h"
00012
00013 namespace Fracture {
00014
00018     class WindowsInput : public Input
00019     {
00020     protected:
00030         virtual bool IsKeyPressedImpl(int keyCode) override;
00031
00041         virtual bool IsMouseButtonPressedImpl(int button) override;
00042
00050         virtual float GetMouseXImpl() override;
00051
00059         virtual float GetMouseYImpl() override;
00060
00068         virtual std::pair<float, float> GetMousePositionImpl() override;
00069     };
00070
00071 }
```

## 10.99 Fracture/src/Platform/Windows/WindowsWindow.cpp File Reference

```
#include "frpch.h"
#include "WindowsWindow.h"
#include "Fracture\Events\ApplicationEvent.h"
#include "Fracture\Events\KeyEvent.h"
#include "Fracture\Events\MouseEvent.h"
#include "Platform\OpenGL\OpenGLContext.h"
```

### Namespaces

- namespace Fracture

### Functions

- static void Fracture::GLFWErrorCallback (int error, const char *description)

### Variables

- static uint8_t Fracture::s_GLFWWindowCount = 0

## 10.100 Fracture/src/Platform/Windows/WindowsWindow.h File Reference

Contains the WindowsWindow class that is used to create a window for the application.

```
#include "Fracture/Core/Window.h"
#include "Fracture/Renderer/GraphicsContext.h"
#include <GLFW/glfw3.h>
```

### Classes

- class Fracture::WindowsWindow

  *The WindowsWindow class is used to create a window for the application. It owns the renderer context.*
- struct Fracture::WindowsWindow::WindowData

  *A unique pointer to the renderer context.*

### Namespaces

- namespace Fracture

## 10.100.1 Detailed Description

Contains the WindowsWindow class that is used to create a window for the application.

**See also**

Window

**Author**

Aditya Rajagopal

# 10.101 WindowsWindow.h

Go to the documentation of this file.

```
00001 #pragma once
00013 #include "Fracture/Core/Window.h"
00014 #include "Fracture/Renderer/GraphicsContext.h"
00015
00016 #include <GLFW/glfw3.h>
00017
00018 namespace Fracture {
00019
00025     class WindowsWindow : public Window
00026     {
00027     public:
00035         WindowsWindow(const WindowProperties& props);
00036
00044         virtual ~WindowsWindow();
00045
00051         void OnUpdate() override;
00052
00058         inline uint32_t GetWidth() const override { return m_Data.Width; }
00059
00065         inline uint32_t GetHeight() const override { return m_Data.Height; }
00066
00074         inline void SetEventCallback(const EventCallbackFn& callback) override { m_Data.EventCallback
      = callback; }
00075
00081         void SetVSync(bool enabled) override;
00082
00088         bool IsVSync() const override;
00089
00095         inline virtual void* GetNativeWindow() const override { return m_Window; }
00096     private:
00106         virtual void Init(const WindowProperties& props);
00107
00113         virtual void Shutdown();
00114     private:
00115         GLFWwindow* m_Window;
00116         Scope<GraphicsContext> m_Context;
00117
00127         struct WindowData
```

```
00128         {
00129             std::string Title;
00130             uint32_t Width, Height;
00131             EventCallbackFn EventCallback;
00132             bool VSync;
00133 
00134             WindowData() = default;
00135         };
00136 
00137         WindowData m_Data;
00138     };
00139 
00140 }
00141 
```

# Index