vengine

0.1.0

Generated by Doxygen 1.11.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 ven::Buffer Class Reference

Class for buffer.

`#include <Buffer.hpp>`

Public Member Functions

- **Buffer** (Device &device, VkDeviceSize instanceSize, uint32_t instanceCount, VkBufferUsageFlags usage↩ Flags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize minOffsetAlignment=1)
- **Buffer** (const Buffer &)=delete
- Buffer & **operator=** (const Buffer &)=delete
- VkResult map (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0)

  Map a memory range of this buffer. If successful, mapped points to the specified buffer range.
- void unmap ()

  Unmap a mapped memory range.
- void writeToBuffer (const void *data, VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Copies the specified data to the mapped buffer. Default value writes whole buffer range.
- VkResult flush (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Flush a memory range of the buffer to make it visible to the device.
- VkDescriptorBufferInfo descriptorInfo (const VkDeviceSize size=VK_WHOLE_SIZE, const VkDeviceSize offset=0) const

  Create a buffer info descriptor.
- VkResult invalidate (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Invalidate a memory range of the buffer to make it visible to the host.
- void writeToIndex (const void *data, const VkDeviceSize index) const
- VkResult flushIndex (const VkDeviceSize index) const
- VkDescriptorBufferInfo descriptorInfoForIndex (const VkDeviceSize index) const
- VkResult invalidateIndex (const VkDeviceSize index) const
- VkBuffer **getBuffer** () const
- void * **getMappedMemory** () const
- uint32_t **getInstanceCount** () const
- VkDeviceSize **getInstanceSize** () const
- VkDeviceSize **getAlignmentSize** () const
- VkBufferUsageFlags **getUsageFlags** () const
- VkMemoryPropertyFlags **getMemoryPropertyFlags** () const
- VkDeviceSize **getBufferSize** () const

3.1.1 Detailed Description

Class for buffer.

3.1.2 Member Function Documentation

3.1.2.1 descriptorInfo()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfo (
            const VkDeviceSize size = VK_WHOLE_SIZE,
            const VkDeviceSize offset = 0) const  [inline], [nodiscard]
```

Create a buffer info descriptor.

Parameters

size	(Optional) Size of the memory range of the descriptor
offset	(Optional) Byte offset from beginning

Returns

VkDescriptorBufferInfo of specified offset and range

3.1.2.2 descriptorInfoForIndex()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfoForIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Create a buffer info descriptor

Parameters

index	Specifies the region given by index ∗ alignmentSize

Returns

VkDescriptorBufferInfo for instance at index

3.1.2.3 flush()

```
VkResult ven::Buffer::flush (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const  [nodiscard]
```

Flush a memory range of the buffer to make it visible to the device.

Note

Only required for non-coherent memory

Parameters

size	(Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the flush call

3.1.2.4 flushIndex()

```
VkResult ven::Buffer::flushIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Flush the memory range at index ∗ alignmentSize of the buffer to make it visible to the device

Parameters

index	Used in offset calculation

3.1.2.5 invalidate()

```
VkResult ven::Buffer::invalidate (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const  [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host.

Note

Only required for non-coherent memory

Parameters

size	(Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to invalidate the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the invalidate call

3.1.2.6 invalidateIndex()

```
VkResult ven::Buffer::invalidateIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host

Note

Only required for non-coherent memory

Parameters

index	Specifies the region to invalidate: index ∗ alignmentSize

Returns

VkResult of the invalidate call

3.1.2.7 map()

```
VkResult ven::Buffer::map (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0)
```

Map a memory range of this buffer. If successful, mapped points to the specified buffer range.

Parameters

size	(Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the buffer mapping call

3.1.2.8 unmap()

```
void ven::Buffer::unmap ()
```

Unmap a mapped memory range.

Note

Does not return a result as vkUnmapMemory can't fail

3.1.2.9 writeToBuffer()

```
void ven::Buffer::writeToBuffer (
            const void * data,
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const
```

Copies the specified data to the mapped buffer. Default value writes whole buffer range.

Parameters

data	Pointer to the data to copy
size	(Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the complete buffer range.
offset	(Optional) Byte offset from beginning of mapped region

3.1.2.10 writeToIndex()

```
void ven::Buffer::writeToIndex (
            const void * data,
            const VkDeviceSize index) const  [inline]
```

Copies "instanceSize" bytes of data to the mapped buffer at an offset of index ∗ alignmentSize

Parameters

data	Pointer to the data to copy
index	Used in offset calculation

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

- include/VEngine/Buffer.hpp

3.2 ven::DescriptorPool::Builder Class Reference

Public Member Functions

- **Builder** (Device &device)
- Builder & **addPoolSize** (VkDescriptorType descriptorType, uint32_t count)
- Builder & **setPoolFlags** (VkDescriptorPoolCreateFlags flags)
- Builder & **setMaxSets** (uint32_t count)
- std::unique_ptr< DescriptorPool > **build** () const

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

- include/VEngine/Descriptors.hpp

3.3 ven::DescriptorSetLayout::Builder Class Reference

Public Member Functions

- **Builder** (Device &device)
- Builder & **addBinding** (uint32_t binding, VkDescriptorType descriptorType, VkShaderStageFlags stageFlags, uint32_t count=1)
- std::unique_ptr< DescriptorSetLayout > **build** () const

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

- include/VEngine/Descriptors.hpp

3.4 ven::Model::Builder Struct Reference

Public Member Functions

- void **loadModel** (const std::string &filename)

Public Attributes

- std::vector< Vertex > **vertices**
- std::vector< uint32_t > **indices**

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

- include/VEngine/Model.hpp

3.5 ven::Camera Class Reference

Public Member Functions

- void **setOrthographicProjection** (float left, float right, float top, float bottom, float near, float far)
- void **setPerspectiveProjection** (float fovy, float aspect, float near, float far)
- void **setViewDirection** (glm::vec3 position, glm::vec3 direction, glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void **setViewTarget** (glm::vec3 position, glm::vec3 target, glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void **setViewYXZ** (glm::vec3 position, glm::vec3 rotation)
- const glm::mat4 & **getProjection** () const
- const glm::mat4 & **getView** () const
- const glm::mat4 & **getInverseView** () const

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

- include/VEngine/Camera.hpp

3.6 myLib::Clock Class Reference

Class for time management.

`#include <Clock.hpp>`

Public Member Functions

- void **restart** ()

  Restart the clock.
- void **pause** ()

  Pause the clock.
- void **resume** ()

  Resume the clock.
- Time getElapsedTime () const

  Get the elapsed time since the last restart.

3.6.1 Detailed Description

Class for time management.

3.6.2 Member Function Documentation

3.6.2.1 getElapsedTime()

```
Time myLib::Clock::getElapsedTime () const  [nodiscard]
```

Get the elapsed time since the last restart.

Returns

Time The elapsed time

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

- lib/local/static/myLib/include/myLib/Clock/Clock.hpp

3.7 ven::DescriptorPool Class Reference

Class for descriptor pool.

```
#include <Descriptors.hpp>
```

Classes

- class Builder

Public Member Functions

- **DescriptorPool** (Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags, const std::vector< VkDescriptorPoolSize > &poolSizes)
- **DescriptorPool** (const DescriptorPool &)=delete
- DescriptorPool & **operator=** (const DescriptorPool &)=delete
- bool **allocateDescriptor** (VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet &descriptor) const
- void **freeDescriptors** (const std::vector< VkDescriptorSet > &descriptors) const
- void **resetPool** () const

Friends

- class **DescriptorWriter**

3.7.1 Detailed Description

Class for descriptor pool.

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

- include/VEngine/Descriptors.hpp

3.8 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

`#include <Descriptors.hpp>`

Classes

- class Builder

Public Member Functions

- **DescriptorSetLayout** (Device &device, const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > &bindings)
- **DescriptorSetLayout** (const DescriptorSetLayout &)=delete
- DescriptorSetLayout & **operator=** (const DescriptorSetLayout &)=delete
- VkDescriptorSetLayout **getDescriptorSetLayout** () const

Friends

- class **DescriptorWriter**

3.8.1 Detailed Description

Class for descriptor set layout.

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

- include/VEngine/Descriptors.hpp

3.9 ven::DescriptorWriter Class Reference

Class for descriptor writer.

`#include <Descriptors.hpp>`

Public Member Functions

- **DescriptorWriter** (DescriptorSetLayout &setLayout, DescriptorPool &pool)
- DescriptorWriter & **writeBuffer** (uint32_t binding, const VkDescriptorBufferInfo *bufferInfo)
- DescriptorWriter & **writeImage** (uint32_t binding, const VkDescriptorImageInfo *imageInfo)
- bool **build** (VkDescriptorSet &set)
- void **overwrite** (const VkDescriptorSet &set)

3.9.1 Detailed Description

Class for descriptor writer.

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

- include/VEngine/Descriptors.hpp

3.10 ven::Device Class Reference

Public Member Functions

- **Device** (Window &window)
- **Device** (const Device &)=delete
- Device & **operator=** (const Device &)=delete
- **Device** (Device &&)=delete
- Device & **operator=** (Device &&)=delete
- VkCommandPool **getCommandPool** () const
- VkDevice **device** () const
- VkSurfaceKHR **surface** () const
- VkQueue **graphicsQueue** () const
- VkQueue **presentQueue** () const
- SwapChainSupportDetails **getSwapChainSupport** () const
- uint32_t **findMemoryType** (uint32_t typeFilter, VkMemoryPropertyFlags propertiesp) const
- QueueFamilyIndices **findPhysicalQueueFamilies** () const
- VkFormat **findSupportedFormat** (const std::vector< VkFormat > &candidates, VkImageTiling tiling, Vk↩ FormatFeatureFlags features) const
- void **createBuffer** (VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
- VkCommandBuffer **beginSingleTimeCommands** () const
- void **endSingleTimeCommands** (VkCommandBuffer commandBuffer) const
- void **copyBuffer** (VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const
- void **copyBufferToImage** (VkBuffer buffer, VkImage image, uint32_t width, uint32_t height, uint32_t layer↩ Count) const
- void **createImageWithInfo** (const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags properties, Vk↩ Image &image, VkDeviceMemory &imageMemory) const
- VkPhysicalDevice **getPhysicalDevice** () const
- VkQueue **getGraphicsQueue** () const

Public Attributes

- const bool **enableValidationLayers** = true
- VkPhysicalDeviceProperties **m_properties**

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

- include/VEngine/Device.hpp

3.11 ven::Engine Class Reference

Public Member Functions

- **Engine** (uint32_t=DEFAULT_WIDTH, uint32_t=DEFAULT_HEIGHT, const std::string &title=DEFAULT_↩ TITLE.data())
- **Engine** (const Engine &)=delete
- Engine **operator=** (const Engine &)=delete
- Window & **getWindow** ()
- void **mainLoop** ()

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

- include/VEngine/Engine.hpp

3.12 ven::FrameCounter Class Reference

Public Member Functions

- void **update** (const float deltaTime)
- float **getFps** () const
- float **getFrameTime** () const

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

- include/VEngine/FrameCounter.hpp

3.13 ven::FrameInfo Struct Reference

Public Attributes

- int **frameIndex**
- float **frameTime**
- VkCommandBuffer **commandBuffer**
- Camera & **camera**
- VkDescriptorSet **globalDescriptorSet**
- Object::Map & **objects**

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

- include/VEngine/FrameInfo.hpp

3.14 ven::GlobalUbo Struct Reference

Public Attributes

- glm::mat4 **projection** {1.F}
- glm::mat4 **view** {1.F}
- glm::mat4 **inverseView** {1.F}
- glm::vec4 **ambientLightColor** {1.F, 1.F, 1.F, .02F}
- std::array< PointLight, MAX_LIGHTS > **pointLights**
- int **numLights**

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

- include/VEngine/FrameInfo.hpp

3.15 ven::KeyboardController Class Reference

Classes

- struct KeyMappings

Public Member Functions

- void **moveInPlaneXZ** (GLFWwindow *window, float dt, Object &object) const

Public Attributes

- KeyMappings **m_keys** {}
- float **m_moveSpeed** {3.F}
- float **m_lookSpeed** {1.5F}

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

- include/VEngine/KeyboardController.hpp

3.16 ven::KeyboardController::KeyMappings Struct Reference

Public Attributes

- int **moveLeft** = GLFW_KEY_A
- int **moveRight** = GLFW_KEY_D
- int **moveForward** = GLFW_KEY_W
- int **moveBackward** = GLFW_KEY_S
- int **moveUp** = GLFW_KEY_SPACE
- int **moveDown** = GLFW_KEY_LEFT_SHIFT
- int **lookLeft** = GLFW_KEY_LEFT
- int **lookRight** = GLFW_KEY_RIGHT
- int **lookUp** = GLFW_KEY_UP
- int **lookDown** = GLFW_KEY_DOWN

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

- include/VEngine/KeyboardController.hpp

3.17 ven::Model Class Reference

Classes

- struct Builder
- struct Vertex

Public Member Functions

- **Model** (Device &device, const Builder &builder)
- **Model** (const Model &)=delete
- void **operator=** (const Model &)=delete
- void **bind** (VkCommandBuffer commandBuffer) const
- void **draw** (VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static std::unique_ptr< Model > **createModelFromFile** (Device &device, const std::string &filename)

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

- include/VEngine/Model.hpp

3.18 ven::Object Class Reference

Public Types

- using **Map** = std::unordered_map<id_t, Object>

Public Member Functions

- **Object** (const Object &)=delete
- Object & **operator=** (const Object &)=delete
- **Object** (Object &&)=default
- Object & **operator=** (Object &&)=default
- id_t **getId** () const

Static Public Member Functions

- static Object **createObject** ()
- static Object **makePointLight** (float intensity=10.F, float radius=0.1F, glm::vec3 color=glm::vec3(1.F))

Public Attributes

- std::shared_ptr< Model > **model** {}
- glm::vec3 **color** {}
- Transform3DComponent **transform3D** {}
- std::unique_ptr< PointLightComponent > **pointLight** = nullptr

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

- include/VEngine/Object.hpp

3.19 ven::PipelineConfigInfo Struct Reference

Public Member Functions

- **PipelineConfigInfo** (const PipelineConfigInfo &)=delete
- PipelineConfigInfo & **operator=** (const PipelineConfigInfo &)=delete

Public Attributes

- std::vector< VkVertexInputBindingDescription > **bindingDescriptions**
- std::vector< VkVertexInputAttributeDescription > **attributeDescriptions**
- VkPipelineInputAssemblyStateCreateInfo **inputAssemblyInfo** {}
- VkPipelineRasterizationStateCreateInfo **rasterizationInfo** {}
- VkPipelineMultisampleStateCreateInfo **multisampleInfo** {}
- VkPipelineColorBlendAttachmentState **colorBlendAttachment** {}
- VkPipelineColorBlendStateCreateInfo **colorBlendInfo** {}
- VkPipelineDepthStencilStateCreateInfo **depthStencilInfo** {}
- std::vector< VkDynamicState > **dynamicStateEnables**
- VkPipelineDynamicStateCreateInfo **dynamicStateInfo** {}
- VkPipelineLayout **pipelineLayout** = nullptr
- VkRenderPass **renderPass** = nullptr
- uint32_t **subpass** = 0

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

- include/VEngine/Shaders.hpp

3.20 ven::PointLight Struct Reference

Public Attributes

- glm::vec4 **position** {}
- glm::vec4 **color** {}

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

- include/VEngine/FrameInfo.hpp

3.21 ven::PointLightComponent Struct Reference

Public Attributes

- float **lightIntensity** = 1.0F

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

- include/VEngine/Object.hpp

3.22 ven::PointLightSystem Class Reference

Class for point light system.

`#include <PointLightSystem.hpp>`

Public Member Functions

- **PointLightSystem** (Device &device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout)
- **PointLightSystem** (const PointLightSystem &)=delete
- PointLightSystem & **operator=** (const PointLightSystem &)=delete
- void **render** (const FrameInfo &frameInfo) const

Static Public Member Functions

- static void **update** (const FrameInfo &frameInfo, GlobalUbo &ubo)

3.22.1 Detailed Description

Class for point light system.

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

- include/VEngine/System/PointLightSystem.hpp

3.23 ven::QueueFamilyIndices Struct Reference

Public Member Functions

- bool **isComplete** () const

Public Attributes

- uint32_t **graphicsFamily** {}
- uint32_t **presentFamily** {}
- bool **graphicsFamilyHasValue** = false
- bool **presentFamilyHasValue** = false

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

- include/VEngine/Device.hpp

3.24 myLib::Random Class Reference

Class for random number generation.

`#include <Random.hpp>`

Static Public Member Functions

- static int randomInt (int min, int max)

  Generate a random integer between min and max.
- static int **randomInt** ()
- static float randomFloat (float min, float max)
- static float **randomFloat** ()

3.24.1 Detailed Description

Class for random number generation.

3.24.2 Member Function Documentation

3.24.2.1 randomFloat()

```
static float myLib::Random::randomFloat (
            float min,
            float max)  [static]
```

Parameters

min	The minimum value
max	The maximum value

Returns

float The random float

3.24.2.2 randomInt()

```
static int myLib::Random::randomInt (
            int min,
            int max)  [static]
```

Generate a random integer between min and max.

Parameters

min	The minimum value
max	The maximum value

Returns

int The random integer

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

- lib/local/static/myLib/include/myLib/Random.hpp

3.25 ven::Renderer Class Reference

Public Member Functions

- **Renderer** (Window &window, Device &device)
- **Renderer** (const Renderer &)=delete
- Renderer & **operator=** (const Renderer &)=delete
- VkRenderPass **getSwapChainRenderPass** () const
- float **getAspectRatio** () const
- bool **isFrameInProgress** () const
- VkCommandBuffer **getCurrentCommandBuffer** () const
- int **getFrameIndex** () const
- VkCommandBuffer **beginFrame** ()
- void **endFrame** ()
- void **beginSwapChainRenderPass** (VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void **endSwapChainRenderPass** (VkCommandBuffer commandBuffer)

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

- include/VEngine/Renderer.hpp

3.26 ven::RenderSystem Class Reference

Class for render system.

```
#include <RenderSystem.hpp>
```

Public Member Functions

- **RenderSystem** (Device &device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout)
- **RenderSystem** (const RenderSystem &)=delete
- RenderSystem & **operator=** (const RenderSystem &)=delete
- void **renderObjects** (const FrameInfo &frameInfo) const

3.26.1 Detailed Description

Class for render system.

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

- include/VEngine/System/RenderSystem.hpp

3.27 ven::Shaders Class Reference

Public Member Functions

- **Shaders** (Device &device, const std::string &vertFilepath, const std::string &fragFilepath, const PipelineConfigInfo &configInfo)
- **Shaders** (const Shaders &)=delete
- Shaders & **operator=** (const Shaders &)=delete
- void **bind** (const VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void **defaultPipelineConfigInfo** (PipelineConfigInfo &configInfo)

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

- include/VEngine/Shaders.hpp

3.28 ven::SimplePushConstantData Struct Reference

Public Attributes

- glm::mat4 **modelMatrix** {1.F}
- glm::mat4 **normalMatrix** {1.F}

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

- include/VEngine/System/RenderSystem.hpp

3.29 ven::SwapChain Class Reference

Public Member Functions

- **SwapChain** (Device &deviceRef, const VkExtent2D windowExtentRef)
- **SwapChain** (Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr< SwapChain > previous)
- **SwapChain** (const SwapChain &)=delete
- SwapChain & **operator=** (const SwapChain &)=delete
- VkFramebuffer **getFrameBuffer** (const unsigned long index) const
- VkRenderPass **getRenderPass** () const
- VkImageView **getImageView** (const int index) const
- size_t **imageCount** () const
- VkFormat **getSwapChainImageFormat** () const
- VkExtent2D **getSwapChainExtent** () const
- uint32_t **width** () const
- uint32_t **height** () const
- float **extentAspectRatio** () const
- VkFormat **findDepthFormat** () const
- VkResult **acquireNextImage** (uint32_t *imageIndex) const
- VkResult **submitCommandBuffers** (const VkCommandBuffer *buffers, const uint32_t *imageIndex)
- bool **compareSwapFormats** (const SwapChain &swapChainp) const

Static Public Attributes

- static constexpr int **MAX_FRAMES_IN_FLIGHT** = 2

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

- include/VEngine/SwapChain.hpp

3.30 ven::SwapChainSupportDetails Struct Reference

Public Attributes

- VkSurfaceCapabilitiesKHR **capabilities**
- std::vector< VkSurfaceFormatKHR > **formats**
- std::vector< VkPresentModeKHR > **presentModes**

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

- include/VEngine/Device.hpp

3.31 myLib::Time Class Reference

Class used for time management.

`#include <Time.hpp>`

Public Member Functions

- **Time** (const double seconds)

  Construct a new Time object.
- int asSeconds () const

  Transform the time to seconds.
- int asMilliseconds () const

  Transform the time to milliseconds.
- int asMicroseconds () const

  Transform the time to microseconds.

3.31.1 Detailed Description

Class used for time management.

3.31.2 Member Function Documentation

3.31.2.1 asMicroseconds()

```
int myLib::Time::asMicroseconds () const  [inline], [nodiscard]
```

Transform the time to microseconds.

Returns

int The time in microseconds

3.31.2.2 asMilliseconds()

```
int myLib::Time::asMilliseconds () const  [inline], [nodiscard]
```

Transform the time to milliseconds.

Returns

int The time in milliseconds

3.31.2.3 asSeconds()

```
int myLib::Time::asSeconds () const  [inline], [nodiscard]
```

Transform the time to seconds.

Returns

int The time in seconds

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

- lib/local/static/myLib/include/myLib/Clock/Time.hpp

3.32 ven::Transform3DComponent Struct Reference

Public Member Functions

- glm::mat4 **mat4** () const
- glm::mat3 **normalMatrix** () const

Public Attributes

- glm::vec3 **translation** {}
- glm::vec3 **scale** {1.F, 1.F, 1.F}
- glm::vec3 **rotation** {}

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

- include/VEngine/Object.hpp

3.33 ven::Model::Vertex Struct Reference

Public Member Functions

- bool **operator==** (const Vertex &other) const

Static Public Member Functions

- static std::vector< VkVertexInputBindingDescription > **getBindingDescriptions** ()
- static std::vector< VkVertexInputAttributeDescription > **getAttributeDescriptions** ()

Public Attributes

- glm::vec3 **position** {}
- glm::vec3 **color** {}
- glm::vec3 **normal** {}
- glm::vec2 **uv** {}

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

- include/VEngine/Model.hpp

3.34 ven::Window Class Reference

Public Member Functions

- **Window** (const uint32_t width, const uint32_t height, const std::string &title)
- GLFWwindow * **createWindow** (uint32_t width, uint32_t height, const std::string &title)
- void **createWindowSurface** (VkInstance instance, VkSurfaceKHR *surface) const
- GLFWwindow * **getGLFWindow** () const
- VkExtent2D **getExtent** () const
- bool **wasWindowResized** () const
- void **resetWindowResizedFlag** ()

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

- include/VEngine/Window.hpp

Chapter 4

File Documentation

4.1 include/VEngine/Buffer.hpp File Reference

This file contains the Buffer class.

```
#include "VEngine/Device.hpp"
```

Classes

- class ven::Buffer

  Class for buffer.

4.1.1 Detailed Description

This file contains the Buffer class.

4.2 Buffer.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include "VEngine/Device.hpp"
00010
00011 namespace ven {
00012
00017     class Buffer {
00018
00019         public:
00020
00021             Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
      VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
      minOffsetAlignment = 1);
00022             ~Buffer();
00023
00024             Buffer(const Buffer&) = delete;
00025             Buffer& operator=(const Buffer&) = delete;
00026
00035             VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00036
```

```
00042             void unmap();
00043
00051             void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
      offset = 0) const;
00052
00063             [[nodiscard]] VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0)
      const;
00064
00073             [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size =
      VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset,
      size, }; }
00074
00086             [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset =
      0) const;
00087
00095             void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
      m_instanceSize, index * m_alignmentSize); }
00096
00102             [[nodiscard]] VkResult flushIndex(const VkDeviceSize index) const { return
      flush(m_alignmentSize, index * m_alignmentSize); }
00103
00112             [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index)
      const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }
00113
00123             [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return
      invalidate(m_alignmentSize, index * m_alignmentSize); }
00124
00125             [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
00126             [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
00127             [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
00128             [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }
00129             [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_instanceSize; }
00130             [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00131             [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
      m_memoryPropertyFlags; }
00132             [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }
00133
00134         private:
00144             static VkDeviceSize getAlignment(VkDeviceSize instanceSize, VkDeviceSize
      minOffsetAlignment);
00145
00146             Device& m_device;
00147             void* m_mapped = nullptr;
00148             VkBuffer m_buffer = VK_NULL_HANDLE;
00149             VkDeviceMemory m_memory = VK_NULL_HANDLE;
00150
00151             VkDeviceSize m_bufferSize;
00152             VkDeviceSize m_instanceSize;
00153             uint32_t m_instanceCount;
00154             VkDeviceSize m_alignmentSize;
00155             VkBufferUsageFlags m_usageFlags;
00156             VkMemoryPropertyFlags m_memoryPropertyFlags;
00157
00158     }; // class Buffer
00159
00160 } // namespace ven
```

4.3 include/VEngine/Camera.hpp File Reference

This file contains the Camera class.

```
#include <glm/glm.hpp>
```

Classes

- class ven::Camera

4.3.1 Detailed Description

This file contains the Camera class.

This file contains the KeyboardController class.

4.4 Camera.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #define GLM_FORCE_RADIANS
00010 #define GLM_FORCE_DEPTH_ZERO_TO_ONE
00011 #include <glm/glm.hpp>
00012
00013 namespace ven {
00014
00016
00017     class Camera {
00018
00019         public:
00020
00021             void setOrthographicProjection(float left, float right, float top, float bottom, float
      near, float far);
00022             void setPerspectiveProjection(float fovy, float aspect, float near, float far);
00023             void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up =
      glm::vec3{0.F, -1.F, 0.F});
00024             void setViewTarget(glm::vec3 position, glm::vec3 target, glm::vec3 up = glm::vec3{0.F,
      -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00025             void setViewYXZ(glm::vec3 position, glm::vec3 rotation);
00026
00027             [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00028             [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00029             [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00030
00031         private:
00032
00033             glm::mat4 m_projectionMatrix{1.F};
00034             glm::mat4 m_viewMatrix{1.F};
00035             glm::mat4 m_inverseViewMatrix{1.F};
00036
00037     }; // class Camera
00038
00039 } // namespace ven
```

4.5 include/VEngine/Constant.hpp File Reference

This file contains the constant values used in the project.

Typedefs

- using ven::return_type_t

4.5.1 Detailed Description

This file contains the constant values used in the project.

4.5.2 Typedef Documentation

4.5.2.1 return_type_t

```
using ven::return_type_t
```

Initial value:

```
 enum ReturnType : uint8_t {
        VEN_SUCCESS = 0,
        VEN_FAILURE = 1
    }
```

4.6 Constant.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 namespace ven {
00010
00011     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00012     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00013
00014     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00015     static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/";
00016
00017     using return_type_t = enum ReturnType : uint8_t {
00018         VEN_SUCCESS = 0,
00019         VEN_FAILURE = 1
00020     };
00021
00022 } // namespace ven
```

4.7 include/VEngine/Descriptors.hpp File Reference

This file contains the Descriptors class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"
```

Classes

- class ven::DescriptorSetLayout

  Class for descriptor set layout.
- class ven::DescriptorSetLayout::Builder
- class ven::DescriptorPool

  Class for descriptor pool.
- class ven::DescriptorPool::Builder
- class ven::DescriptorWriter

  Class for descriptor writer.

4.7.1 Detailed Description

This file contains the Descriptors class.

4.8 Descriptors.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00021     class DescriptorSetLayout {
00022
00023         public:
00024
00025             class Builder {
00026
00027                 public:
00028
00029                     explicit Builder(Device &device) : m_device{device} {}
00030
00031                     Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType,
      VkShaderStageFlags stageFlags, uint32_t count = 1);
00032                     std::unique_ptr<DescriptorSetLayout> build() const { return
      std::make_unique<DescriptorSetLayout>(m_device, m_bindings); }
00033
00034                 private:
00035                     Device &m_device;
00036                     std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00037             };
00038
00039             DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
      VkDescriptorSetLayoutBinding>& bindings);
00040             ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(),
      m_descriptorSetLayout, nullptr); }
00041             DescriptorSetLayout(const DescriptorSetLayout &) = delete;
00042             DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;
00043
00044             VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }
00045
00046         private:
00047
00048             Device &m_device;
00049             VkDescriptorSetLayout m_descriptorSetLayout;
00050             std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00051
00052             friend class DescriptorWriter;
00053
00054     }; // class DescriptorSetLayout
00055
00061     class DescriptorPool {
00062
00063         public:
00064
00065             class Builder {
00066
00067                 public:
00068
00069                     explicit Builder(Device &device) : m_device{device} {}
00070
00071                     Builder &addPoolSize(VkDescriptorType descriptorType, uint32_t count);
00072                     Builder &setPoolFlags(VkDescriptorPoolCreateFlags flags);
00073                     Builder &setMaxSets(uint32_t count);
00074                     [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
      std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00075
00076                 private:
00077
00078                     Device &m_device;
00079                     std::vector<VkDescriptorPoolSize> m_poolSizes;
00080                     uint32_t m_maxSets = 1000;
00081                     VkDescriptorPoolCreateFlags m_poolFlags = 0;
00082             };
00083
00084             DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
      const std::vector<VkDescriptorPoolSize> &poolSizes);
00085             ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
      }
00086             DescriptorPool(const DescriptorPool &) = delete;
00087             DescriptorPool &operator=(const DescriptorPool &) = delete;
00088
00089             bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
      &descriptor) const;
```

```
00090
00091             void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
      vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
      descriptors.data()); }
00092
00093             void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00094
00095         private:
00096
00097             Device &m_device;
00098             VkDescriptorPool m_descriptorPool;
00099
00100             friend class DescriptorWriter;
00101
00102     }; // class DescriptorPool
00103
00109     class DescriptorWriter {
00110
00111         public:
00112
00113             DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :
      m_setLayout{setLayout}, m_pool{pool} {}
00114
00115             DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00116             DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00117
00118             bool build(VkDescriptorSet &set);
00119             void overwrite(const VkDescriptorSet &set);
00120
00121         private:
00122
00123             DescriptorSetLayout &m_setLayout;
00124             DescriptorPool &m_pool;
00125             std::vector<VkWriteDescriptorSet> m_writes;
00126
00127     }; // class DescriptorWriter
00128
00129 } // namespace ven
```

4.9 include/VEngine/Device.hpp File Reference

This file contains the Device class.

```
#include <vector>
#include "VEngine/Window.hpp"
```

Classes

- struct ven::SwapChainSupportDetails
- struct ven::QueueFamilyIndices
- class ven::Device

4.9.1 Detailed Description

This file contains the Device class.

4.10 Device.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <vector>
00010
00011 #include "VEngine/Window.hpp"
00012
00013 namespace ven {
00014
00015     struct SwapChainSupportDetails {
00016         VkSurfaceCapabilitiesKHR capabilities;
00017         std::vector<VkSurfaceFormatKHR> formats;
00018         std::vector<VkPresentModeKHR> presentModes;
00019     };
00020
00021     struct QueueFamilyIndices {
00022         uint32_t graphicsFamily{};
00023         uint32_t presentFamily{};
00024         bool graphicsFamilyHasValue = false;
00025         bool presentFamilyHasValue = false;
00026         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
      presentFamilyHasValue; }
00027     };
00028
00029     class Device {
00030
00031         public:
00032
00033             #ifdef NDEBUG
00034                 const bool enableValidationLayers = false;
00035             #else
00036                 const bool enableValidationLayers = true;
00037             #endif
00038
00039             explicit Device(Window &window);
00040             ~Device();
00041
00042             Device(const Device &) = delete;
00043             Device& operator=(const Device &) = delete;
00044             Device(Device &&) = delete;
00045             Device &operator=(Device &&) = delete;
00046
00047             [[nodiscard]] VkCommandPool getCommandPool() const { return commandPool; }
00048             [[nodiscard]] VkDevice device() const { return device_; }
00049             [[nodiscard]] VkSurfaceKHR surface() const { return surface_; }
00050             [[nodiscard]] VkQueue graphicsQueue() const { return graphicsQueue_; }
00051             [[nodiscard]] VkQueue presentQueue() const { return presentQueue_; }
00052
00053         [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return
      querySwapChainSupport(physicalDevice); }
00054         [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags propertiesp)
      const;
00055         [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
      findQueueFamilies(physicalDevice); }
00056         [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
      VkImageTiling tiling, VkFormatFeatureFlags features) const;
00057
00058             // Buffer Helper Functions
00059             void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
      propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00060             [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00061             void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00062             void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00063             void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
      uint32_t layerCount) const;
00064
00065             void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
      properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00066
00067             VkPhysicalDeviceProperties m_properties;
00068
00069             [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return physicalDevice; }
00070             [[nodiscard]] VkQueue getGraphicsQueue() const { return graphicsQueue_; }
00071
00072         private:
00073
00074             void createInstance();
00075             void setupDebugMessenger();
00076             void createSurface() { m_window.createWindowSurface(instance, &surface_); };
00077             void pickPhysicalDevice();
00078             void createLogicalDevice();
```

```
00079             void createCommandPool();
00080
00081             // helper functions
00082             bool isDeviceSuitable(VkPhysicalDevice device) const;
00083             [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00084             [[nodiscard]] bool checkValidationLayerSupport() const;
00085             QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00086             static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
      &createInfo);
00087             void hasGlfwRequiredInstanceExtensions() const;
00088             bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00089             SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00090
00091             VkInstance instance;
00092             VkDebugUtilsMessengerEXT debugMessenger;
00093             VkPhysicalDevice physicalDevice = VK_NULL_HANDLE;
00094             Window &m_window;
00095             VkCommandPool commandPool;
00096
00097             VkDevice device_;
00098             VkSurfaceKHR surface_;
00099             VkQueue graphicsQueue_;
00100             VkQueue presentQueue_;
00101
00102             const std::vector<const char *> validationLayers = {"VK_LAYER_KHRONOS_validation"};
00103             const std::vector<const char *> deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00104
00105     }; // class Device
00106
00107 }  // namespace ven
```

4.11 include/VEngine/Engine.hpp File Reference

This file contains the Engine class.

```
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Constant.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors.hpp"
```

Classes

- class ven::Engine

4.11.1 Detailed Description

This file contains the Engine class.

4.12 Engine.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Window.hpp"
```

```
00012 #include "VEngine/Constant.hpp"
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Object.hpp"
00015 #include "VEngine/Renderer.hpp"
00016 #include "VEngine/Descriptors.hpp"
00017
00018 namespace ven {
00019
00020     class Engine {
00021
00022     public:
00023
00024         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string &title
      = DEFAULT_TITLE.data());
00025         ~Engine() = default;
00026
00027         Engine(const Engine &) = delete;
00028         Engine operator=(const Engine &) = delete;
00029
00030         Window &getWindow() { return m_window; };
00031
00032         void mainLoop();
00033
00034     private:
00035
00036         void loadObjects();
00037
00038         Window m_window;
00039         Device m_device{m_window};
00040         Renderer m_renderer{m_window, m_device};
00041
00042         std::unique_ptr<DescriptorPool> m_globalPool;
00043         Object::Map m_objects;
00044
00045         VkInstance m_instance{nullptr};
00046         VkSurfaceKHR m_surface{nullptr};
00047
00048         void createInstance();
00049         void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWindow(),
      nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00050
00051     }; // class Engine
00052
00053 } // namespace ven
```

4.13 include/VEngine/FrameCounter.hpp File Reference

This file contains the FrameCounter class.

```
#include <iostream>
```

Classes

- class ven::FrameCounter

4.13.1 Detailed Description

This file contains the FrameCounter class.

4.14 FrameCounter.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <iostream>
00010
00011 namespace ven {
00012
00013     class FrameCounter {
00014
00015         public:
00016
00017             FrameCounter() = default;
00018             ~FrameCounter() = default;
00019
00020             void update(const float deltaTime) {
00021                 m_frameCounter += 1.F;
00022                 m_timeCounter += deltaTime;
00023
00024                 if (m_timeCounter >= 1.F) {
00025                     std::cout « "FPS: " « m_frameCounter « '\n';
00026                     m_fps = m_frameCounter;
00027                     m_frameTime = 1000.F / m_fps;
00028                     m_frameCounter = 0.F;
00029                     m_timeCounter = 0.F;
00030                 }
00031             }
00032
00033             [[nodiscard]] float getFps() const { return m_fps; }
00034             [[nodiscard]] float getFrameTime() const { return m_frameTime; }
00035
00036         private:
00037
00038             float m_fps{0.F};
00039             float m_frameTime{0.F};
00040             float m_frameCounter{0.F};
00041             float m_timeCounter{0.F};
00042
00043     }; // class FrameCounter
00044
00045 } // namespace ven
```

4.15 include/VEngine/FrameInfo.hpp File Reference

This file contains the FrameInfo class.

```
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"
```

Classes

- struct ven::PointLight
- struct ven::GlobalUbo
- struct ven::FrameInfo

4.15.1 Detailed Description

This file contains the FrameInfo class.

4.16 FrameInfo.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Camera.hpp"
00012 #include "VEngine/Object.hpp"
00013
00014 namespace ven {
00015
00016 static constexpr std::size_t MAX_LIGHTS = 10;
00017
00018     struct PointLight
00019     {
00020         glm::vec4 position{};
00021         glm::vec4 color{};
00022     };
00023
00024     struct GlobalUbo
00025     {
00026         glm::mat4 projection{1.F};
00027         glm::mat4 view{1.F};
00028         glm::mat4 inverseView{1.F};
00029         glm::vec4 ambientLightColor{1.F, 1.F, 1.F, .02F};
00030         std::array<PointLight, MAX_LIGHTS> pointLights;
00031         int numLights;
00032     };
00033
00034     struct FrameInfo
00035     {
00036         int frameIndex;
00037         float frameTime;
00038         VkCommandBuffer commandBuffer;
00039         Camera &camera;
00040         VkDescriptorSet globalDescriptorSet;
00041         Object::Map &objects;
00042     };
00043
00044 } // namespace ven
```

4.17 KeyboardController.hpp

```
00001
00006
00007 #pragma once
00008
00009 #include "VEngine/Window.hpp"
00010 #include "VEngine/Object.hpp"
00011
00012 namespace ven {
00013
00014     class KeyboardController {
00015
00016         public:
00017
00018             struct KeyMappings {
00019                 int moveLeft = GLFW_KEY_A;
00020                 int moveRight = GLFW_KEY_D;
00021                 int moveForward = GLFW_KEY_W;
00022                 int moveBackward = GLFW_KEY_S;
00023                 int moveUp = GLFW_KEY_SPACE;
00024                 int moveDown = GLFW_KEY_LEFT_SHIFT;
00025                 int lookLeft = GLFW_KEY_LEFT;
00026                 int lookRight = GLFW_KEY_RIGHT;
00027                 int lookUp = GLFW_KEY_UP;
00028                 int lookDown = GLFW_KEY_DOWN;
00029             };
00030
00031             void moveInPlaneXZ(GLFWwindow* window, float dt, Object& object) const;
00032
00033             KeyMappings m_keys{};
00034             float m_moveSpeed{3.F};
00035             float m_lookSpeed{1.5F};
00036
00037     }; // class KeyboardController
00038
00039 } // namespace ven
```

4.18 include/VEngine/Model.hpp File Reference

This file contains the Model class.

```
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"
```

Classes

- class ven::Model
- struct ven::Model::Vertex
- struct ven::Model::Builder

4.18.1 Detailed Description

This file contains the Model class.

4.19 Model.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Buffer.hpp"
00013
00014 namespace ven {
00015
00016     class Model {
00017
00018         public:
00019
00020             struct Vertex {
00021                 glm::vec3 position{};
00022                 glm::vec3 color{};
00023                 glm::vec3 normal{};
00024                 glm::vec2 uv{};
00025
00026                 static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00027                 static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00028
00029                 bool operator==(const Vertex& other) const {
00030                     return position == other.position && color == other.color && normal ==
      other.normal && uv == other.uv;
00031                 }
00032             };
00033
00034             struct Builder {
00035                 std::vector<Vertex> vertices;
00036                 std::vector<uint32_t> indices;
00037
00038                 void loadModel(const std::string &filename);
00039             };
00040
00041             Model(Device &device, const Builder &builder);
00042             ~Model();
00043
00044             Model(const Model&) = delete;
00045             void operator=(const Model&) = delete;
00046
00047             static std::unique_ptr<Model> createModelFromFile(Device &device, const std::string
      &filename);
```

```
00048
00049             void bind(VkCommandBuffer commandBuffer) const;
00050             void draw(VkCommandBuffer commandBuffer) const;
00051
00052         private:
00053
00054             void createVertexBuffer(const std::vector<Vertex>& vertices);
00055             void createIndexBuffer(const std::vector<uint32_t>& indices);
00056
00057             Device& m_device;
00058             std::unique_ptr<Buffer> m_vertexBuffer;
00059             uint32_t m_vertexCount;
00060
00061             bool m_hasIndexBuffer{false};
00062             std::unique_ptr<Buffer> m_indexBuffer;
00063             uint32_t m_indexCount;
00064
00065     }; // class Model
00066
00067 } // namespace ven
```

4.20 include/VEngine/Object.hpp File Reference

This file contains the Object class.

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"
```

Classes

- struct ven::Transform3DComponent
- struct ven::PointLightComponent
- class ven::Object

Typedefs

- using **ven::id_t** = unsigned int

4.20.1 Detailed Description

This file contains the Object class.

4.21 Object.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include <glm/gtc/matrix_transform.hpp>
00013
00014 #include "VEngine/Model.hpp"
00015
```

```
00016 namespace ven {
00017
00018     using id_t = unsigned int;
00019
00020     struct Transform3DComponent {
00021         glm::vec3 translation{};
00022         glm::vec3 scale{1.F, 1.F, 1.F};
00023         glm::vec3 rotation{};
00024
00025         [[nodiscard]] glm::mat4 mat4() const;
00026         [[nodiscard]] glm::mat3 normalMatrix() const;
00027     };
00028
00029     struct PointLightComponent {
00030         float lightIntensity = 1.0F;
00031     };
00032
00033     class Object {
00034
00035         public:
00036
00037          using Map = std::unordered_map<id_t, Object>;
00038
00039
00040             static Object createObject() { static id_t objId = 0; return Object(objId++); }
00041
00042             ~Object() = default;
00043
00044             static Object makePointLight(float intensity = 10.F, float radius = 0.1F, glm::vec3 color
      = glm::vec3(1.F));
00045
00046             Object(const Object&) = delete;
00047             Object& operator=(const Object&) = delete;
00048             Object(Object&&) = default;
00049             Object& operator=(Object&&) = default;
00050
00051             [[nodiscard]] id_t getId() const { return m_objId; }
00052
00053             std::shared_ptr<Model> model{};
00054             glm::vec3 color{};
00055             Transform3DComponent transform3D{};
00056
00057             std::unique_ptr<PointLightComponent> pointLight = nullptr;
00058
00059     private:
00060             explicit Object(const id_t objId) : m_objId(objId) {}
00061
00062             id_t m_objId;
00063
00064     }; // class Object
00065
00066 } // namespace ven
```

4.22 include/VEngine/Renderer.hpp File Reference

This file contains the Renderer class.

```
#include <memory>
#include <cassert>
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/SwapChain.hpp"
```

Classes

- class ven::Renderer

4.22.1 Detailed Description

This file contains the Renderer class.

4.23 Renderer.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <cassert>
00011
00012 #include <vulkan/vulkan.h>
00013
00014 #include "VEngine/Window.hpp"
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/SwapChain.hpp"
00017
00018 namespace ven {
00019
00020     class Renderer {
00021
00022     public:
00023
00024         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
      recreateSwapChain(); createCommandBuffers(); }
00025         ~Renderer() { freeCommandBuffers(); }
00026
00027         Renderer(const Renderer &) = delete;
00028         Renderer& operator=(const Renderer &) = delete;
00029
00030         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
      m_swapChain->getRenderPass(); }
00031         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00032         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00033         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress() &&
      "cannot get command m_buffer when frame not in progress"); return
      m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00034
00035         [[nodiscard]] int getFrameIndex() const { assert(isFrameInProgress() && "cannot get frame
      index when frame not in progress"); return m_currentFrameIndex; }
00036
00037         VkCommandBuffer beginFrame();
00038         void endFrame();
00039         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00040         static void endSwapChainRenderPass(VkCommandBuffer commandBuffer);
00041
00042     private:
00043
00044         void createCommandBuffers();
00045         void freeCommandBuffers();
00046         void recreateSwapChain();
00047
00048         Window &m_window;
00049         Device &m_device;
00050         std::unique_ptr<SwapChain> m_swapChain;
00051         std::vector<VkCommandBuffer> m_commandBuffers;
00052
00053         uint32_t m_currentImageIndex{0};
00054         int m_currentFrameIndex{0};
00055         bool m_isFrameStarted{false};
00056
00057     }; // class Renderer
00058
00059 } // namespace ven
```

4.24 include/VEngine/Shaders.hpp File Reference

This file contains the Shader class.

```
#include <string>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
#include "VEngine/Device.hpp"
#include "VEngine/Model.hpp"
```

Classes

- struct ven::PipelineConfigInfo
- class ven::Shaders

4.24.1 Detailed Description

This file contains the Shader class.

4.25 Shaders.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #include <vulkan/vulkan.h>
00012 #include <glm/glm.hpp>
00013
00014 #include "VEngine/Device.hpp"
00015 #include "VEngine/Model.hpp"
00016
00017 namespace ven {
00018
00019     struct PipelineConfigInfo {
00020         PipelineConfigInfo() = default;
00021         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00022         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00023
00024         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00025         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00026         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00027         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00028         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00029         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00030         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00031         VkPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00032         std::vector<VkDynamicState> dynamicStateEnables;
00033         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00034         VkPipelineLayout pipelineLayout = nullptr;
00035         VkRenderPass renderPass = nullptr;
00036         uint32_t subpass = 0;
00037     };
00038
00039     class Shaders {
00040
00041         public:
00042
00043             Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
      const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
      fragFilepath, configInfo); };
00044             ~Shaders();
00045
00046             Shaders(const Shaders&) = delete;
00047             Shaders& operator=(const Shaders&) = delete;
00048
00049             static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00050             void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
      VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00051
00052         private:
00053
00054             static std::vector<char> readFile(const std::string &filename);
00055             void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
      fragFilepath, const PipelineConfigInfo& configInfo);
00056             void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
      const;
00057
00058             Device& m_device;
00059             VkPipeline m_graphicsPipeline{nullptr};
00060             VkShaderModule m_vertShaderModule{nullptr};
00061             VkShaderModule m_fragShaderModule{nullptr};
00062
00063     }; // class Shaders
00064
00065 } // namespace ven
```

4.26 include/VEngine/SwapChain.hpp File Reference

This file contains the Shader class.

```
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"
```

Classes

- class ven::SwapChain

4.26.1 Detailed Description

This file contains the Shader class.

4.27 SwapChain.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010 #include <memory>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     class SwapChain {
00017
00018         public:
00019
00020             static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00021
00022             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : device{deviceRef},
      windowExtent{windowExtentRef} { init(); }
00023             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
      previous) : device{deviceRef}, windowExtent{windowExtentRef}, oldSwapChain{std::move(previous)} {
      init(); oldSwapChain = nullptr; }
00024             ~SwapChain();
00025
00026             SwapChain(const SwapChain &) = delete;
00027             SwapChain& operator=(const SwapChain &) = delete;
00028
00029             [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return
      swapChainFramebuffers[index]; }
00030             [[nodiscard]] VkRenderPass getRenderPass() const { return renderPass; }
00031             [[nodiscard]] VkImageView getImageView(const int index) const { return
      swapChainImageViews[static_cast<unsigned long>(index)]; }
00032             [[nodiscard]] size_t imageCount() const { return swapChainImages.size(); }
00033             [[nodiscard]] VkFormat getSwapChainImageFormat() const { return swapChainImageFormat; }
00034             [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00035             [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
00036             [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00037
00038             [[nodiscard]] float extentAspectRatio() const { return
      static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
00039             VkFormat findDepthFormat() const;
00040
00041             VkResult acquireNextImage(uint32_t *imageIndex) const;
00042             VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00043
00044             [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChainp) const {
00045                 return swapChainImageFormat == swapChainp.swapChainImageFormat && swapChainDepthFormat
      == swapChainp.swapChainDepthFormat;
```

```
00046             }
00047
00048         private:
00049
00050             void init();
00051             void createSwapChain();
00052             void createImageViews();
00053             void createDepthResources();
00054             void createRenderPass();
00055             void createFramebuffers();
00056             void createSyncObjects();
00057
00058             static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
      &availableFormats);
00059             static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
      &availablePresentModes);
00060             VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const;
00061
00062             VkFormat swapChainImageFormat{};
00063             VkFormat swapChainDepthFormat{};
00064             VkExtent2D m_swapChainExtent{};
00065
00066             std::vector<VkFramebuffer> swapChainFramebuffers;
00067             VkRenderPass renderPass{};
00068
00069             std::vector<VkImage> depthImages;
00070             std::vector<VkDeviceMemory> depthImageMemorys;
00071             std::vector<VkImageView> depthImageViews;
00072             std::vector<VkImage> swapChainImages;
00073             std::vector<VkImageView> swapChainImageViews;
00074
00075             Device &device;
00076             VkExtent2D windowExtent;
00077
00078             VkSwapchainKHR swapChain{};
00079             std::shared_ptr<SwapChain> oldSwapChain;
00080
00081             std::vector<VkSemaphore> imageAvailableSemaphores;
00082             std::vector<VkSemaphore> renderFinishedSemaphores;
00083             std::vector<VkFence> inFlightFences;
00084             std::vector<VkFence> imagesInFlight;
00085             size_t currentFrame = 0;
00086
00087     }; // class SwapChain
00088
00089 }  // namespace ven
```

4.28 include/VEngine/System/PointLightSystem.hpp File Reference

This file contains the PointLightSystem class.

```
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"
```

Classes

- class ven::PointLightSystem

  Class for point light system.

4.28.1 Detailed Description

This file contains the PointLightSystem class.

4.29 PointLightSystem.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Shaders.hpp"
00013 #include "VEngine/FrameInfo.hpp"
00014
00015 namespace ven {
00016
00022     class PointLightSystem {
00023
00024         public:
00025
00026             explicit PointLightSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
      globalSetLayout);
00027             ~PointLightSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout,
      nullptr); }
00028
00029             PointLightSystem(const PointLightSystem&) = delete;
00030             PointLightSystem& operator=(const PointLightSystem&) = delete;
00031
00032             static void update(const FrameInfo &frameInfo, GlobalUbo &ubo);
00033             void render(const FrameInfo &frameInfo) const;
00034
00035         private:
00036
00037             void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00038             void createPipeline(VkRenderPass renderPass);
00039
00040             Device &m_device;
00041
00042             std::unique_ptr<Shaders> m_shaders;
00043             VkPipelineLayout m_pipelineLayout{nullptr};
00044
00045     }; // class PointLightSystem
00046
00047 } // namespace ven
```

4.30 include/VEngine/System/RenderSystem.hpp File Reference

This file contains the RenderSystem class.

```
#include <memory>
#include <vulkan/vulkan.h>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"
```

Classes

- struct ven::SimplePushConstantData
- class ven::RenderSystem

  Class for render system.

4.30.1 Detailed Description

This file contains the RenderSystem class.

4.31 RenderSystem.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Shaders.hpp"
00015 #include "VEngine/FrameInfo.hpp"
00016
00017 namespace ven {
00018
00019     struct SimplePushConstantData {
00020         glm::mat4 modelMatrix{1.F};
00021         glm::mat4 normalMatrix{1.F};
00022     };
00023
00028     class RenderSystem {
00029
00030     public:
00031
00032         explicit RenderSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
      globalSetLayout);
00033         ~RenderSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout, nullptr); }
00034
00035         RenderSystem(const RenderSystem&) = delete;
00036         RenderSystem& operator=(const RenderSystem&) = delete;
00037
00038         void renderObjects(const FrameInfo &frameInfo) const;
00039
00040     private:
00041
00042         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00043         void createPipeline(VkRenderPass renderPass);
00044
00045         Device &m_device;
00046
00047         std::unique_ptr<Shaders> m_shaders;
00048         VkPipelineLayout m_pipelineLayout{nullptr};
00049
00050     }; // class RenderSystem
00051
00052 } // namespace ven
```

4.32 include/VEngine/Utils.hpp File Reference

```
#include <functional>
```

Functions

- template<typename T , typename... Rest>
  void **ven::hashCombine** (std::size_t &seed, const T &v, const Rest &... rest)

4.33 Utils.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <functional>
00010
```

```
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed « 6) + (seed » 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018
00019 } // namespace ven
```

4.34 include/VEngine/Window.hpp File Reference

This file contains the Window class.

```
#include <string>
#include <GLFW/glfw3.h>
#include <vulkan/vulkan.h>
```

Classes

- class ven::Window

4.34.1 Detailed Description

This file contains the Window class.

4.35 Window.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     class Window {
00018
00019         public:
00020
00021             Window(const uint32_t width, const uint32_t height, const std::string &title) :
      m_window(createWindow(width, height, title)), m_width(width), m_height(height) {};
00022             ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr;};
00023
00024             [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
      &title);
00025             void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00026
00027             [[nodiscard]] GLFWwindow* getGLFWindow() const { return m_window; };
00028
00029             [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; };
00030             [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00031             void resetWindowResizedFlag() { m_framebufferResized = false; }
00032
00033         private:
00034
00035             static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00036
```

```
00037             GLFWwindow* m_window{nullptr};
00038             uint32_t m_width;
00039             uint32_t m_height;
00040 
00041             bool m_framebufferResized = false;
00042 
00043     }; // class Window
00044 
00045 } // namespace ven
```

4.36 lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference

Clock class for time management.

```
#include <chrono>
#include "myLib/Clock/Time.hpp"
```

Classes

- class myLib::Clock

  Class for time management.

Typedefs

- using **TimePoint** = std::chrono::time_point<std::chrono::high_resolution_clock>

  TimePoint is a type alias for a time point which is a very long and complicated type in the standard library.

4.36.1 Detailed Description

Clock class for time management.

4.37 Clock.hpp

Go to the documentation of this file.

```
00001 
00006 
00007 #pragma once
00008 
00009 #include <chrono>
00010 
00011 #include "myLib/Clock/Time.hpp"
00012 
00016 using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;
00017 
00018 namespace myLib {
00019 
00023     class Clock {
00024 
00025         public:
00026 
00027             Clock() : m_start(std::chrono::high_resolution_clock::now()) {};
00028 
00029             ~Clock() = default;
00030 
00034             void restart() { m_start = std::chrono::high_resolution_clock::now(); };
00035 
00039             void pause();
00040 
```

```
00044             void resume();
00045
00050             [[nodiscard]] Time getElapsedTime() const;
00051
00052         private:
00053
00057             TimePoint m_start;
00058
00062             TimePoint m_pause;
00063
00067             bool m_paused{false};
00068
00069     }; // Clock
00070
00071 } // namespace myLib
```

4.38 lib/local/static/myLib/include/myLib/Clock/Time.hpp File Reference

Class for time management.

Classes

- class myLib::Time

  Class used for time management.

4.38.1 Detailed Description

Class for time management.

4.39 Time.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 namespace myLib {
00010
00015     class Time {
00016
00017         public:
00018
00022             explicit Time(const double seconds) : m_seconds(seconds) {};
00023
00028             [[nodiscard]] int asSeconds() const { return static_cast<int>(m_seconds); };
00029
00034             [[nodiscard]] int asMilliseconds() const { return static_cast<int>(m_seconds * 1000); }
00035
00040             [[nodiscard]] int asMicroseconds() const { return static_cast<int>(m_seconds * 1000000);
      };
00041
00042         private:
00043
00047             double m_seconds{0.0F};
00048
00049     }; // Time
00050
00051 } // namespace myLib
```

4.40 lib/local/static/myLib/include/myLib/Random.hpp File Reference

Class for random number generation.

`#include <random>`

Classes

- class myLib::Random

  Class for random number generation.

4.40.1 Detailed Description

Class for random number generation.

4.41 Random.hpp

Go to the documentation of this file.

```
00001
00006
00007 #pragma once
00008
00009 #include <random>
00010
00011 namespace myLib {
00012
00017     class Random {
00018
00019         public:
00020
00027             static int randomInt(int min, int max);
00028             static int randomInt() { return randomInt(-1000, 1000); };
00029
00035             static float randomFloat(float min, float max);
00036             static float randomFloat() { return randomFloat(-1.0f, 1.0f); };
00037
00038     }; // class Random
00039
00040 } // namespace myLib
```

Index