

**vengine**

0.0.1

Generated by Doxygen 1.11.0



---

<b>1 engine</b>	<b>1</b>
1.1 VEngine - Vulkan based Game Engine .....	1
1.1.1 Features .....	1
1.1.2 Planned Features: .....	1
1.1.3 Build .....	2
1.1.3.1 Prerequisites .....	2
1.1.3.2 Linux .....	2
1.1.3.3 Windows .....	2
1.1.3.4 Command Line Options .....	2
1.1.4 Key Bindings .....	3
1.1.5 Documentation .....	3
1.1.6 External Libraries .....	4
1.1.7 Commit Norms .....	4
1.1.8 License .....	4
1.1.9 Acknowledgements .....	4
<b>2 Namespace Index</b>	<b>5</b>
2.1 Namespace List .....	5
<b>3 Hierarchical Index</b>	<b>7</b>
3.1 Class Hierarchy .....	7
<b>4 Class Index</b>	<b>9</b>
4.1 Class List .....	9
<b>5 File Index</b>	<b>11</b>
5.1 File List .....	11
<b>6 Namespace Documentation</b>	<b>15</b>
6.1 ven Namespace Reference .....	15
6.1.1 Typedef Documentation .....	18
6.1.1.1 TimePoint .....	18
6.1.2 Enumeration Type Documentation .....	18
6.1.2.1 ENGINE_STATE .....	18
6.1.2.2 GUI_STATE .....	18
6.1.2.3 LogLevel .....	18
6.1.3 Function Documentation .....	19
6.1.3.1 hashCombine() .....	19
6.1.3.2 isNumeric() .....	19
6.1.4 Variable Documentation .....	20
6.1.4.1 COLOR_MAX .....	20
6.1.4.2 DEFAULT_AMBIENT_LIGHT_COLOR .....	20
6.1.4.3 DEFAULT_AMBIENT_LIGHT_INTENSITY .....	20
6.1.4.4 DEFAULT_CLEAR_COLOR .....	20

6.1.4.5 DEFAULT_CLEAR_DEPTH . . . . .	20
6.1.4.6 DEFAULT_FAR . . . . .	20
6.1.4.7 DEFAULT_FOV . . . . .	21
6.1.4.8 DEFAULT_HEIGHT . . . . .	21
6.1.4.9 DEFAULT_KEY_MAPPINGS . . . . .	21
6.1.4.10 DEFAULT_LIGHT_COLOR . . . . .	21
6.1.4.11 DEFAULT_LIGHT_INTENSITY . . . . .	21
6.1.4.12 DEFAULT_LIGHT_RADIUS . . . . .	21
6.1.4.13 DEFAULT_LOOK_SPEED . . . . .	21
6.1.4.14 DEFAULT_MAX_SETS . . . . .	22
6.1.4.15 DEFAULT_MOVE_SPEED . . . . .	22
6.1.4.16 DEFAULT_NEAR . . . . .	22
6.1.4.17 DEFAULT_POSITION . . . . .	22
6.1.4.18 DEFAULT_ROTATION . . . . .	22
6.1.4.19 DEFAULT_SHININESS . . . . .	22
6.1.4.20 DEFAULT_TITLE . . . . .	23
6.1.4.21 DEFAULT_TRANSFORM . . . . .	23
6.1.4.22 DEFAULT_WIDTH . . . . .	23
6.1.4.23 DESCRIPTOR_COUNT . . . . .	23
6.1.4.24 EPSILON . . . . .	23
6.1.4.25 FUNCTION_MAP_OPT_LONG . . . . .	23
6.1.4.26 FUNCTION_MAP_OPT_SHORT . . . . .	24
6.1.4.27 HELP_MESSAGE . . . . .	24
6.1.4.28 MAX_FRAMES_IN_FLIGHT . . . . .	24
6.1.4.29 MAX_LIGHTS . . . . .	24
6.1.4.30 MAX_OBJECTS . . . . .	25
6.1.4.31 SHADERS_BIN_PATH . . . . .	25
6.1.4.32 VERSION_MESSAGE . . . . .	25
<b>7 Class Documentation</b> . . . . .	<b>27</b>
7.1 ven::ARenderSystemBase Class Reference . . . . .	27
7.1.1 Detailed Description . . . . .	29
7.1.2 Constructor & Destructor Documentation . . . . .	29
7.1.2.1 ARenderSystemBase() . . . . .	29
7.1.2.2 ~ARenderSystemBase() . . . . .	29
7.1.3 Member Function Documentation . . . . .	30
7.1.3.1 createPipeline() . . . . .	30
7.1.3.2 createPipelineLayout() . . . . .	30
7.1.3.3 getDevice() . . . . .	31
7.1.3.4 getPipelineLayout() . . . . .	31
7.1.3.5 getShaders() . . . . .	32
7.1.3.6 render() . . . . .	32

---

7.1.4 Member Data Documentation . . . . .	32
7.1.4.1 m_device . . . . .	32
7.1.4.2 m_pipelineLayout . . . . .	33
7.1.4.3 m_shaders . . . . .	33
7.1.4.4 renderSystemLayout . . . . .	33
7.2 ven::Buffer Class Reference . . . . .	33
7.2.1 Detailed Description . . . . .	36
7.2.2 Constructor & Destructor Documentation . . . . .	36
7.2.2.1 Buffer() [1/2] . . . . .	36
7.2.2.2 ~Buffer() . . . . .	36
7.2.2.3 Buffer() [2/2] . . . . .	36
7.2.3 Member Function Documentation . . . . .	36
7.2.3.1 descriptorInfo() . . . . .	36
7.2.3.2 descriptorInfoForIndex() . . . . .	37
7.2.3.3 flush() . . . . .	38
7.2.3.4 flushIndex() . . . . .	38
7.2.3.5 getAlignment() . . . . .	39
7.2.3.6 getAlignmentSize() . . . . .	39
7.2.3.7 getBuffer() . . . . .	40
7.2.3.8 getBufferSize() . . . . .	40
7.2.3.9 getInstanceCount() . . . . .	40
7.2.3.10 getInstanceSize() . . . . .	40
7.2.3.11 getMappedMemory() . . . . .	40
7.2.3.12 getMemoryPropertyFlags() . . . . .	40
7.2.3.13 getUsageFlags() . . . . .	41
7.2.3.14 invalidate() . . . . .	41
7.2.3.15 invalidateIndex() . . . . .	41
7.2.3.16 map() . . . . .	42
7.2.3.17 operator=( ) . . . . .	43
7.2.3.18 unmap() . . . . .	43
7.2.3.19 writeToBuffer() . . . . .	43
7.2.3.20 writeToIndex() . . . . .	43
7.2.4 Member Data Documentation . . . . .	44
7.2.4.1 m_alignmentSize . . . . .	44
7.2.4.2 m_buffer . . . . .	44
7.2.4.3 m_bufferSize . . . . .	44
7.2.4.4 m_device . . . . .	45
7.2.4.5 m_instanceCount . . . . .	45
7.2.4.6 m_instanceSize . . . . .	45
7.2.4.7 m_mapped . . . . .	45
7.2.4.8 m_memory . . . . .	45
7.2.4.9 m_memoryPropertyFlags . . . . .	45

7.2.4.10 m_usageFlags . . . . .	46
7.3 ven::DescriptorPool::Builder Class Reference . . . . .	46
7.3.1 Detailed Description . . . . .	48
7.3.2 Constructor & Destructor Documentation . . . . .	48
7.3.2.1 Builder() . . . . .	48
7.3.3 Member Function Documentation . . . . .	48
7.3.3.1 addPoolSize() . . . . .	48
7.3.3.2 build() . . . . .	49
7.3.3.3 setMaxSets() . . . . .	49
7.3.3.4 setPoolFlags() . . . . .	50
7.3.4 Member Data Documentation . . . . .	50
7.3.4.1 m_device . . . . .	50
7.3.4.2 m_maxSets . . . . .	50
7.3.4.3 m_poolFlags . . . . .	50
7.3.4.4 m_poolSizes . . . . .	51
7.4 ven::DescriptorsetLayout::Builder Class Reference . . . . .	51
7.4.1 Detailed Description . . . . .	53
7.4.2 Constructor & Destructor Documentation . . . . .	53
7.4.2.1 Builder() . . . . .	53
7.4.3 Member Function Documentation . . . . .	53
7.4.3.1 addBinding() . . . . .	53
7.4.3.2 build() . . . . .	54
7.4.4 Member Data Documentation . . . . .	54
7.4.4.1 m_bindings . . . . .	54
7.4.4.2 m_device . . . . .	54
7.5 ven::Model::Builder Struct Reference . . . . .	55
7.5.1 Detailed Description . . . . .	56
7.5.2 Member Function Documentation . . . . .	56
7.5.2.1 loadModel() . . . . .	56
7.5.2.2 processMesh() . . . . .	56
7.5.2.3 processNode() . . . . .	56
7.5.3 Member Data Documentation . . . . .	57
7.5.3.1 indices . . . . .	57
7.5.3.2 vertices . . . . .	57
7.6 ven::Camera Class Reference . . . . .	57
7.6.1 Detailed Description . . . . .	59
7.6.2 Constructor & Destructor Documentation . . . . .	59
7.6.2.1 Camera() [1/2] . . . . .	59
7.6.2.2 ~Camera() . . . . .	60
7.6.2.3 Camera() [2/2] . . . . .	60
7.6.3 Member Function Documentation . . . . .	60
7.6.3.1 getFar() . . . . .	60

---

7.6.3.2 getFov()	60
7.6.3.3 getInverseView()	61
7.6.3.4 getLookSpeed()	61
7.6.3.5 getMoveSpeed()	61
7.6.3.6 getNear()	62
7.6.3.7 getProjection()	62
7.6.3.8 getView()	62
7.6.3.9 operator=()	62
7.6.3.10 setFar()	63
7.6.3.11 setFov()	63
7.6.3.12 setLookSpeed()	64
7.6.3.13 setMoveSpeed()	64
7.6.3.14 setNear()	65
7.6.3.15 setOrthographicProjection()	65
7.6.3.16 setPerspectiveProjection()	65
7.6.3.17 setViewDirection()	65
7.6.3.18 setViewTarget()	66
7.6.3.19 setViewXYZ()	66
7.6.4 Member Data Documentation	66
7.6.4.1 m_far	66
7.6.4.2 m_fov	66
7.6.4.3 m_inverseViewMatrix	66
7.6.4.4 m_lookSpeed	66
7.6.4.5 m_moveSpeed	67
7.6.4.6 m_near	67
7.6.4.7 m_projectionMatrix	67
7.6.4.8 m_viewMatrix	67
7.6.4.9 transform	67
7.7 ven::CameraConf Struct Reference	68
7.7.1 Detailed Description	68
7.7.2 Member Data Documentation	68
7.7.2.1 far	68
7.7.2.2 fov	68
7.7.2.3 look_speed	69
7.7.2.4 move_speed	69
7.7.2.5 near	69
7.8 ven::Clock Class Reference	69
7.8.1 Detailed Description	70
7.8.2 Constructor & Destructor Documentation	70
7.8.2.1 Clock() [1/2]	70
7.8.2.2 ~Clock()	71
7.8.2.3 Clock() [2/2]	71

7.8.3 Member Function Documentation . . . . .	71
7.8.3.1 getDeltaTime() . . . . .	71
7.8.3.2 getDeltaTimeMS() . . . . .	71
7.8.3.3 getFPS() . . . . .	72
7.8.3.4 now() . . . . .	72
7.8.3.5 operator=() . . . . .	72
7.8.3.6 resume() . . . . .	72
7.8.3.7 start() . . . . .	72
7.8.3.8 stop() . . . . .	73
7.8.3.9 update() . . . . .	73
7.8.4 Member Data Documentation . . . . .	73
7.8.4.1 m_deltaTime . . . . .	73
7.8.4.2 m_isStopped . . . . .	73
7.8.4.3 m_startTime . . . . .	73
7.8.4.4 m_stopTime . . . . .	74
7.9 ven::Gui::ClockData Struct Reference . . . . .	74
7.9.1 Detailed Description . . . . .	74
7.9.2 Member Data Documentation . . . . .	74
7.9.2.1 deltaTimeMS . . . . .	74
7.9.2.2 fps . . . . .	75
7.10 ven::Colors Class Reference . . . . .	75
7.10.1 Detailed Description . . . . .	77
7.10.2 Member Data Documentation . . . . .	77
7.10.2.1 AQUA_3 . . . . .	77
7.10.2.2 AQUA_4 . . . . .	77
7.10.2.3 AQUA_V . . . . .	77
7.10.2.4 BLACK_3 . . . . .	77
7.10.2.5 BLACK_4 . . . . .	78
7.10.2.6 BLACK_V . . . . .	78
7.10.2.7 BLUE_3 . . . . .	78
7.10.2.8 BLUE_4 . . . . .	78
7.10.2.9 BLUE_V . . . . .	78
7.10.2.10 COLOR_PRESETS_3 . . . . .	78
7.10.2.11 COLOR_PRESETS_4 . . . . .	79
7.10.2.12 COLOR_PRESETS_VK . . . . .	79
7.10.2.13 CYAN_3 . . . . .	79
7.10.2.14 CYAN_4 . . . . .	80
7.10.2.15 CYAN_V . . . . .	80
7.10.2.16 FUCHSIA_3 . . . . .	80
7.10.2.17 FUCHSIA_4 . . . . .	80
7.10.2.18 FUCHSIA_V . . . . .	80
7.10.2.19 GRAY_3 . . . . .	80

---

7.10.2.20 GRAY_4 . . . . .	80
7.10.2.21 GRAY_V . . . . .	81
7.10.2.22 GREEN_3 . . . . .	81
7.10.2.23 GREEN_4 . . . . .	81
7.10.2.24 GREEN_V . . . . .	81
7.10.2.25 LIME_3 . . . . .	81
7.10.2.26 LIME_4 . . . . .	81
7.10.2.27 LIME_V . . . . .	81
7.10.2.28 MAGENTA_3 . . . . .	82
7.10.2.29 MAGENTA_4 . . . . .	82
7.10.2.30 MAGENTA_V . . . . .	82
7.10.2.31 MAROON_3 . . . . .	82
7.10.2.32 MAROON_4 . . . . .	82
7.10.2.33 MAROON_V . . . . .	82
7.10.2.34 NAVY_3 . . . . .	82
7.10.2.35 NAVY_4 . . . . .	83
7.10.2.36 NAVY_V . . . . .	83
7.10.2.37 NIGHT_BLUE_3 . . . . .	83
7.10.2.38 NIGHT_BLUE_4 . . . . .	83
7.10.2.39 NIGHT_BLUE_V . . . . .	83
7.10.2.40 OLIVE_3 . . . . .	83
7.10.2.41 OLIVE_4 . . . . .	83
7.10.2.42 OLIVE_V . . . . .	84
7.10.2.43 RED_3 . . . . .	84
7.10.2.44 RED_4 . . . . .	84
7.10.2.45 RED_V . . . . .	84
7.10.2.46 SILVER_3 . . . . .	84
7.10.2.47 SILVER_4 . . . . .	84
7.10.2.48 SILVER_V . . . . .	84
7.10.2.49 SKY_BLUE_3 . . . . .	85
7.10.2.50 SKY_BLUE_4 . . . . .	85
7.10.2.51 SKY_BLUE_V . . . . .	85
7.10.2.52 SUNSET_3 . . . . .	85
7.10.2.53 SUNSET_4 . . . . .	85
7.10.2.54 SUNSET_V . . . . .	85
7.10.2.55 TEAL_3 . . . . .	85
7.10.2.56 TEAL_4 . . . . .	86
7.10.2.57 TEAL_V . . . . .	86
7.10.2.58 WHITE_3 . . . . .	86
7.10.2.59 WHITE_4 . . . . .	86
7.10.2.60 WHITE_V . . . . .	86
7.10.2.61 YELLOW_3 . . . . .	86

7.10.2.62 YELLOW_4 . . . . .	86
7.10.2.63 YELLOW_V . . . . .	87
7.11 ven::Config Struct Reference . . . . .	87
7.11.1 Detailed Description . . . . .	87
7.11.2 Member Data Documentation . . . . .	88
7.11.2.1 camera . . . . .	88
7.11.2.2 vsync . . . . .	88
7.11.2.3 window . . . . .	88
7.12 ven::DescriptorPool Class Reference . . . . .	88
7.12.1 Detailed Description . . . . .	90
7.12.2 Constructor & Destructor Documentation . . . . .	90
7.12.2.1 DescriptorPool() [1/2] . . . . .	90
7.12.2.2 ~DescriptorPool() . . . . .	91
7.12.2.3 DescriptorPool() [2/2] . . . . .	91
7.12.3 Member Function Documentation . . . . .	91
7.12.3.1 allocateDescriptor() . . . . .	91
7.12.3.2 freeDescriptors() . . . . .	91
7.12.3.3 getDescriptorPool() . . . . .	92
7.12.3.4 operator=() . . . . .	92
7.12.3.5 resetPool() . . . . .	92
7.12.4 Friends And Related Symbol Documentation . . . . .	92
7.12.4.1 DescriptorWriter . . . . .	92
7.12.5 Member Data Documentation . . . . .	92
7.12.5.1 m_descriptorPool . . . . .	92
7.12.5.2 m_device . . . . .	93
7.13 ven::DescriptorsetLayout Class Reference . . . . .	93
7.13.1 Detailed Description . . . . .	95
7.13.2 Constructor & Destructor Documentation . . . . .	95
7.13.2.1 DescriptorsetLayout() [1/2] . . . . .	95
7.13.2.2 ~DescriptorsetLayout() . . . . .	96
7.13.2.3 DescriptorsetLayout() [2/2] . . . . .	96
7.13.3 Member Function Documentation . . . . .	96
7.13.3.1 getDescriptorsetLayout() . . . . .	96
7.13.3.2 operator=() . . . . .	96
7.13.4 Friends And Related Symbol Documentation . . . . .	96
7.13.4.1 DescriptorWriter . . . . .	96
7.13.5 Member Data Documentation . . . . .	97
7.13.5.1 m_bindings . . . . .	97
7.13.5.2 m_descriptorsetLayout . . . . .	97
7.13.5.3 m_device . . . . .	97
7.14 ven::DescriptorWriter Class Reference . . . . .	97
7.14.1 Detailed Description . . . . .	99

---

7.14.2 Constructor & Destructor Documentation . . . . .	99
7.14.2.1 DescriptorWriter() [1/2] . . . . .	99
7.14.2.2 ~DescriptorWriter() . . . . .	99
7.14.2.3 DescriptorWriter() [2/2] . . . . .	99
7.14.3 Member Function Documentation . . . . .	99
7.14.3.1 build() . . . . .	99
7.14.3.2 operator=() . . . . .	100
7.14.3.3 overwrite() . . . . .	100
7.14.3.4 writeBuffer() . . . . .	100
7.14.3.5 writelImage() . . . . .	100
7.14.4 Member Data Documentation . . . . .	101
7.14.4.1 m_pool . . . . .	101
7.14.4.2 m_setLayout . . . . .	101
7.14.4.3 m_writes . . . . .	101
7.15 ven::Device Class Reference . . . . .	101
7.15.1 Detailed Description . . . . .	104
7.15.2 Constructor & Destructor Documentation . . . . .	104
7.15.2.1 Device() [1/2] . . . . .	104
7.15.2.2 ~Device() . . . . .	105
7.15.2.3 Device() [2/2] . . . . .	105
7.15.3 Member Function Documentation . . . . .	105
7.15.3.1 beginSingleTimeCommands() . . . . .	105
7.15.3.2 checkDeviceExtensionSupport() . . . . .	105
7.15.3.3 checkValidationLayerSupport() . . . . .	105
7.15.3.4 copyBuffer() . . . . .	106
7.15.3.5 copyBufferToImage() . . . . .	106
7.15.3.6 createBuffer() . . . . .	106
7.15.3.7 createCommandPool() . . . . .	107
7.15.3.8 createImageWithInfo() . . . . .	107
7.15.3.9 createInstance() . . . . .	108
7.15.3.10 createLogicalDevice() . . . . .	108
7.15.3.11 createSurface() . . . . .	108
7.15.3.12 device() . . . . .	109
7.15.3.13 endSingleTimeCommands() . . . . .	110
7.15.3.14 findMemoryType() . . . . .	110
7.15.3.15 findPhysicalQueueFamilies() . . . . .	111
7.15.3.16 findQueueFamilies() . . . . .	111
7.15.3.17 findSupportedFormat() . . . . .	112
7.15.3.18 getCommandPool() . . . . .	112
7.15.3.19 getGraphicsQueue() . . . . .	112
7.15.3.20 getInstance() . . . . .	112
7.15.3.21 getPhysicalDevice() . . . . .	113

---

7.15.3.22 getProperties()	113
7.15.3.23 getRequiredExtensions()	113
7.15.3.24 getSwapChainSupport()	114
7.15.3.25 graphicsQueue()	114
7.15.3.26 hasGlfwRequiredInstanceExtensions()	114
7.15.3.27 isDeviceSuitable()	115
7.15.3.28 operator=()	115
7.15.3.29 pickPhysicalDevice()	115
7.15.3.30 populateDebugMessengerCreateInfo()	116
7.15.3.31 presentQueue()	116
7.15.3.32 querySwapChainSupport()	116
7.15.3.33 setupDebugMessenger()	117
7.15.3.34 surface()	117
7.15.3.35 transitionImageLayout()	117
7.15.4 Member Data Documentation	118
7.15.4.1 enableValidationLayers	118
7.15.4.2 m_commandPool	118
7.15.4.3 m_debugMessenger	118
7.15.4.4 m_device	118
7.15.4.5 m_deviceExtensions	118
7.15.4.6 m_graphicsQueue	118
7.15.4.7 m_instance	119
7.15.4.8 m_physicalDevice	119
7.15.4.9 m_presentQueue	119
7.15.4.10 m_properties	119
7.15.4.11 m_surface	119
7.15.4.12 m_validationLayers	119
7.15.4.13 m_window	120
7.16 ven::Engine Class Reference	120
7.16.1 Detailed Description	122
7.16.2 Constructor & Destructor Documentation	122
7.16.2.1 Engine() [1/2]	122
7.16.2.2 ~Engine()	123
7.16.2.3 Engine() [2/2]	123
7.16.3 Member Function Documentation	123
7.16.3.1 loadObjects()	123
7.16.3.2 mainLoop()	124
7.16.3.3 operator=()	125
7.16.4 Member Data Documentation	125
7.16.4.1 m_camera	125
7.16.4.2 m_device	125
7.16.4.3 m_framePools	125

---

7.16.4.4 m_globalPool . . . . .	125
7.16.4.5 m_gui . . . . .	125
7.16.4.6 m_renderer . . . . .	126
7.16.4.7 m_sceneManager . . . . .	126
7.16.4.8 m_state . . . . .	126
7.16.4.9 m_window . . . . .	126
7.17 ven::EventManager Class Reference . . . . .	126
7.17.1 Detailed Description . . . . .	128
7.17.2 Constructor & Destructor Documentation . . . . .	128
7.17.2.1 EventManager() [1/2] . . . . .	128
7.17.2.2 ~EventManager() . . . . .	128
7.17.2.3 EventManager() [2/2] . . . . .	128
7.17.3 Member Function Documentation . . . . .	128
7.17.3.1 handleEvents() . . . . .	128
7.17.3.2 isKeyJustPressed() . . . . .	129
7.17.3.3 moveCamera() . . . . .	129
7.17.3.4 operator=() . . . . .	129
7.17.3.5 processKeyActions() . . . . .	129
7.17.3.6 updateEngineState() . . . . .	130
7.17.4 Member Data Documentation . . . . .	130
7.17.4.1 m_keyState . . . . .	130
7.18 ven::FrameInfo Struct Reference . . . . .	130
7.18.1 Detailed Description . . . . .	132
7.18.2 Member Data Documentation . . . . .	132
7.18.2.1 camera . . . . .	132
7.18.2.2 commandBuffer . . . . .	132
7.18.2.3 frameDescriptorPool . . . . .	132
7.18.2.4 frameIndex . . . . .	132
7.18.2.5 frameTime . . . . .	132
7.18.2.6 globalDescriptorSet . . . . .	133
7.18.2.7 lights . . . . .	133
7.18.2.8 objects . . . . .	133
7.19 ven::Gui::funcs Struct Reference . . . . .	133
7.19.1 Detailed Description . . . . .	134
7.19.2 Member Function Documentation . . . . .	134
7.19.2.1 IsLegacyNativeDupe() . . . . .	134
7.20 ven::GlobalUbo Struct Reference . . . . .	135
7.20.1 Detailed Description . . . . .	135
7.20.2 Member Data Documentation . . . . .	136
7.20.2.1 ambientLightColor . . . . .	136
7.20.2.2 inverseView . . . . .	136
7.20.2.3 numLights . . . . .	136

7.20.2.4 pointLights . . . . .	136
7.20.2.5 projection . . . . .	136
7.20.2.6 view . . . . .	136
7.21 ven::Gui Class Reference . . . . .	137
7.21.1 Detailed Description . . . . .	138
7.21.2 Constructor & Destructor Documentation . . . . .	139
7.21.2.1 Gui() [1/2] . . . . .	139
7.21.2.2 ~Gui() . . . . .	139
7.21.2.3 Gui() [2/2] . . . . .	139
7.21.3 Member Function Documentation . . . . .	139
7.21.3.1 cameraSection() . . . . .	139
7.21.3.2 cleanup() . . . . .	140
7.21.3.3 devicePropertiesSection() . . . . .	141
7.21.3.4 getLightsToRemove() . . . . .	141
7.21.3.5 getObjectsToRemove() . . . . .	141
7.21.3.6 getState() . . . . .	141
7.21.3.7 init() . . . . .	142
7.21.3.8 initStyle() . . . . .	143
7.21.3.9 inputsSection() . . . . .	143
7.21.3.10 lightsSection() . . . . .	143
7.21.3.11 objectsSection() . . . . .	144
7.21.3.12 operator=() . . . . .	144
7.21.3.13 render() . . . . .	144
7.21.3.14 rendererSection() . . . . .	145
7.21.3.15 renderFrameWindow() . . . . .	145
7.21.3.16 setState() . . . . .	145
7.21.4 Member Data Documentation . . . . .	146
7.21.4.1 m_intensity . . . . .	146
7.21.4.2 m_io . . . . .	146
7.21.4.3 m_lightsToRemove . . . . .	146
7.21.4.4 m_objectsToRemove . . . . .	146
7.21.4.5 m_shininess . . . . .	147
7.21.4.6 m_state . . . . .	147
7.22 std::hash< ven::Model::Vertex > Struct Reference . . . . .	147
7.22.1 Detailed Description . . . . .	147
7.22.2 Member Function Documentation . . . . .	148
7.22.2.1 operator()() . . . . .	148
7.23 ven::KeyAction Struct Reference . . . . .	148
7.23.1 Detailed Description . . . . .	149
7.23.2 Member Data Documentation . . . . .	149
7.23.2.1 dir . . . . .	149
7.23.2.2 key . . . . .	149

---

7.23.2.3 value . . . . .	149
7.24 ven::KeyMappings Struct Reference . . . . .	150
7.24.1 Detailed Description . . . . .	150
7.24.2 Member Data Documentation . . . . .	151
7.24.2.1 lookDown . . . . .	151
7.24.2.2 lookLeft . . . . .	151
7.24.2.3 lookRight . . . . .	151
7.24.2.4 lookUp . . . . .	151
7.24.2.5 moveBackward . . . . .	151
7.24.2.6 moveDown . . . . .	151
7.24.2.7 moveForward . . . . .	152
7.24.2.8 moveLeft . . . . .	152
7.24.2.9 moveRight . . . . .	152
7.24.2.10 moveUp . . . . .	152
7.24.2.11 toggleGui . . . . .	152
7.25 ven::Light Class Reference . . . . .	153
7.25.1 Detailed Description . . . . .	154
7.25.2 Member Typedef Documentation . . . . .	154
7.25.2.1 Map . . . . .	154
7.25.3 Constructor & Destructor Documentation . . . . .	154
7.25.3.1 Light() [1/3] . . . . .	154
7.25.3.2 ~Light() . . . . .	155
7.25.3.3 Light() [2/3] . . . . .	155
7.25.3.4 Light() [3/3] . . . . .	155
7.25.4 Member Function Documentation . . . . .	155
7.25.4.1 getId() . . . . .	155
7.25.4.2 getName() . . . . .	155
7.25.4.3 getShininess() . . . . .	155
7.25.4.4 operator=() [1/2] . . . . .	155
7.25.4.5 operator=() [2/2] . . . . .	156
7.25.4.6 setName() . . . . .	156
7.25.4.7 setShininess() . . . . .	156
7.25.5 Member Data Documentation . . . . .	156
7.25.5.1 color . . . . .	156
7.25.5.2 m_lightId . . . . .	156
7.25.5.3 m_name . . . . .	156
7.25.5.4 m_shininess . . . . .	157
7.25.5.5 transform . . . . .	157
7.26 ven::LightFactory Class Reference . . . . .	157
7.26.1 Detailed Description . . . . .	158
7.26.2 Constructor & Destructor Documentation . . . . .	158
7.26.2.1 LightFactory() [1/3] . . . . .	158

---

7.26.2.2 ~LightFactory() . . . . .	158
7.26.2.3 LightFactory() [2/3] . . . . .	158
7.26.2.4 LightFactory() [3/3] . . . . .	158
7.26.3 Member Function Documentation . . . . .	159
7.26.3.1 create() . . . . .	159
7.26.3.2 duplicate() . . . . .	159
7.26.3.3 operator=() [1/2] . . . . .	160
7.26.3.4 operator=() [2/2] . . . . .	160
7.26.4 Member Data Documentation . . . . .	160
7.26.4.1 m_currentLightId . . . . .	160
7.27 ven::LightPushConstantData Struct Reference . . . . .	160
7.27.1 Detailed Description . . . . .	161
7.27.2 Member Data Documentation . . . . .	161
7.27.2.1 color . . . . .	161
7.27.2.2 position . . . . .	161
7.27.2.3 radius . . . . .	161
7.28 ven::Logger Class Reference . . . . .	161
7.28.1 Detailed Description . . . . .	163
7.28.2 Constructor & Destructor Documentation . . . . .	163
7.28.2.1 Logger() . . . . .	163
7.28.3 Member Function Documentation . . . . .	163
7.28.3.1 formatLogMessage() . . . . .	163
7.28.3.2 getColorForDuration() . . . . .	163
7.28.3.3 getInstance() . . . . .	164
7.28.3.4 logExecutionTime() . . . . .	164
7.28.3.5 logWarning() . . . . .	165
7.28.4 Member Data Documentation . . . . .	165
7.28.4.1 LOG_LEVEL_COLOR . . . . .	165
7.28.4.2 LOG_LEVEL_STRING . . . . .	166
7.29 ven::Model Class Reference . . . . .	166
7.29.1 Detailed Description . . . . .	168
7.29.2 Constructor & Destructor Documentation . . . . .	168
7.29.2.1 Model() [1/2] . . . . .	168
7.29.2.2 ~Model() . . . . .	169
7.29.2.3 Model() [2/2] . . . . .	169
7.29.3 Member Function Documentation . . . . .	169
7.29.3.1 bind() . . . . .	169
7.29.3.2 createIndexBuffer() . . . . .	169
7.29.3.3 createVertexBuffer() . . . . .	170
7.29.3.4 draw() . . . . .	170
7.29.3.5 operator=() . . . . .	170
7.29.4 Member Data Documentation . . . . .	171

---

7.29.4.1 <code>m_device</code>	171
7.29.4.2 <code>m_hasIndexBuffer</code>	171
7.29.4.3 <code>m_indexBuffer</code>	171
7.29.4.4 <code>m_indexCount</code>	171
7.29.4.5 <code>m_vertexBuffer</code>	171
7.29.4.6 <code>m_vertexCount</code>	171
7.30 <code>ven::ModelFactory</code> Class Reference	172
7.30.1 Detailed Description	172
7.30.2 Constructor & Destructor Documentation	173
7.30.2.1 <code>ModelFactory()</code> [1/3]	173
7.30.2.2 <code>~ModelFactory()</code>	173
7.30.2.3 <code>ModelFactory()</code> [2/3]	173
7.30.2.4 <code>ModelFactory()</code> [3/3]	173
7.30.3 Member Function Documentation	173
7.30.3.1 <code>create()</code>	173
7.30.3.2 <code>loadAll()</code>	174
7.30.3.3 <code>operator=()</code> [1/2]	174
7.30.3.4 <code>operator=()</code> [2/2]	174
7.31 <code>ven::Object</code> Class Reference	175
7.31.1 Detailed Description	176
7.31.2 Member Typedef Documentation	176
7.31.2.1 <code>Map</code>	176
7.31.3 Constructor & Destructor Documentation	176
7.31.3.1 <code>Object()</code> [1/3]	176
7.31.3.2 <code>~Object()</code>	177
7.31.3.3 <code>Object()</code> [2/3]	177
7.31.3.4 <code>Object()</code> [3/3]	177
7.31.4 Member Function Documentation	177
7.31.4.1 <code>getBufferInfo()</code>	177
7.31.4.2 <code>getDiffuseMap()</code>	177
7.31.4.3 <code>getId()</code>	178
7.31.4.4 <code>getModel()</code>	178
7.31.4.5 <code>getName()</code>	178
7.31.4.6 <code>operator=()</code> [1/2]	178
7.31.4.7 <code>operator=()</code> [2/2]	179
7.31.4.8 <code>setBufferInfo()</code>	179
7.31.4.9 <code>setDiffuseMap()</code>	179
7.31.4.10 <code>setModel()</code>	179
7.31.4.11 <code>setName()</code>	179
7.31.5 Member Data Documentation	179
7.31.5.1 <code>m_bufferInfo</code>	179
7.31.5.2 <code>m_diffuseMap</code>	180

---

7.31.5.3 m_model . . . . .	180
7.31.5.4 m_name . . . . .	180
7.31.5.5 m_objId . . . . .	180
7.31.5.6 transform . . . . .	180
7.32 ven::ObjectBufferData Struct Reference . . . . .	181
7.32.1 Detailed Description . . . . .	181
7.32.2 Member Data Documentation . . . . .	181
7.32.2.1 modelMatrix . . . . .	181
7.32.2.2 normalMatrix . . . . .	181
7.33 ven::ObjectFactory Class Reference . . . . .	182
7.33.1 Detailed Description . . . . .	182
7.33.2 Constructor & Destructor Documentation . . . . .	183
7.33.2.1 ObjectFactory() [1/3] . . . . .	183
7.33.2.2 ~ObjectFactory() . . . . .	183
7.33.2.3 ObjectFactory() [2/3] . . . . .	183
7.33.2.4 ObjectFactory() [3/3] . . . . .	183
7.33.3 Member Function Documentation . . . . .	183
7.33.3.1 create() . . . . .	183
7.33.3.2 duplicate() . . . . .	184
7.33.3.3 operator=() [1/2] . . . . .	184
7.33.3.4 operator=() [2/2] . . . . .	185
7.33.4 Member Data Documentation . . . . .	185
7.33.4.1 m_currentObjId . . . . .	185
7.34 ven::ObjectPushConstantData Struct Reference . . . . .	185
7.34.1 Detailed Description . . . . .	185
7.34.2 Member Data Documentation . . . . .	186
7.34.2.1 modelMatrix . . . . .	186
7.34.2.2 normalMatrix . . . . .	186
7.35 ven::ObjectRenderSystem Class Reference . . . . .	186
7.35.1 Detailed Description . . . . .	189
7.35.2 Constructor & Destructor Documentation . . . . .	189
7.35.2.1 ObjectRenderSystem() [1/2] . . . . .	189
7.35.2.2 ObjectRenderSystem() [2/2] . . . . .	190
7.35.3 Member Function Documentation . . . . .	190
7.35.3.1 operator=() . . . . .	190
7.35.3.2 render() . . . . .	190
7.36 ven::Parser Class Reference . . . . .	191
7.36.1 Detailed Description . . . . .	193
7.36.2 Constructor & Destructor Documentation . . . . .	193
7.36.2.1 Parser() [1/3] . . . . .	193
7.36.2.2 ~Parser() . . . . .	193
7.36.2.3 Parser() [2/3] . . . . .	193

---

7.36.2.4 Parser() [3/3] . . . . .	194
7.36.3 Member Function Documentation . . . . .	194
7.36.3.1 getConfig() . . . . .	194
7.36.3.2 handleLongOption() . . . . .	194
7.36.3.3 handleShortOptions() . . . . .	194
7.36.3.4 isValidOption() . . . . .	195
7.36.3.5 operator=() [1/2] . . . . .	195
7.36.3.6 operator=() [2/2] . . . . .	195
7.36.3.7 parseArgs() . . . . .	195
7.36.3.8 parseEnv() . . . . .	195
7.36.3.9 printArguments() . . . . .	195
7.36.4 Member Data Documentation . . . . .	196
7.36.4.1 m_config . . . . .	196
7.36.4.2 m_state . . . . .	196
7.37 ven::ParserException Class Reference . . . . .	196
7.37.1 Detailed Description . . . . .	197
7.37.2 Constructor & Destructor Documentation . . . . .	198
7.37.2.1 ParserException() [1/3] . . . . .	198
7.37.2.2 ~ParserException() . . . . .	198
7.37.2.3 ParserException() [2/3] . . . . .	198
7.37.2.4 ParserException() [3/3] . . . . .	198
7.37.3 Member Function Documentation . . . . .	198
7.37.3.1 operator=() [1/2] . . . . .	198
7.37.3.2 operator=() [2/2] . . . . .	198
7.37.3.3 what() . . . . .	198
7.37.4 Member Data Documentation . . . . .	199
7.37.4.1 m_msg . . . . .	199
7.38 ven::PipelineConfigInfo Struct Reference . . . . .	199
7.38.1 Detailed Description . . . . .	200
7.38.2 Constructor & Destructor Documentation . . . . .	200
7.38.2.1 PipelineConfigInfo() [1/2] . . . . .	200
7.38.2.2 PipelineConfigInfo() [2/2] . . . . .	200
7.38.3 Member Function Documentation . . . . .	200
7.38.3.1 operator=() . . . . .	200
7.38.4 Member Data Documentation . . . . .	200
7.38.4.1 attributeDescriptions . . . . .	200
7.38.4.2 bindingDescriptions . . . . .	201
7.38.4.3 colorBlendAttachment . . . . .	201
7.38.4.4 colorBlendInfo . . . . .	201
7.38.4.5 depthStencilInfo . . . . .	201
7.38.4.6 dynamicStateEnables . . . . .	201
7.38.4.7 dynamicStateInfo . . . . .	201

---

7.38.4.8 inputAssemblyInfo . . . . .	202
7.38.4.9 multisampleInfo . . . . .	202
7.38.4.10 pipelineLayout . . . . .	202
7.38.4.11 rasterizationInfo . . . . .	202
7.38.4.12 renderPass . . . . .	202
7.38.4.13 subpass . . . . .	202
7.39 ven::PointLightData Struct Reference . . . . .	203
7.39.1 Detailed Description . . . . .	203
7.39.2 Member Data Documentation . . . . .	203
7.39.2.1 color . . . . .	203
7.39.2.2 padding . . . . .	203
7.39.2.3 position . . . . .	204
7.39.2.4 shininess . . . . .	204
7.40 ven::PointLightRenderSystem Class Reference . . . . .	204
7.40.1 Detailed Description . . . . .	206
7.40.2 Constructor & Destructor Documentation . . . . .	206
7.40.2.1 PointLightRenderSystem() [1/2] . . . . .	206
7.40.2.2 PointLightRenderSystem() [2/2] . . . . .	207
7.40.3 Member Function Documentation . . . . .	207
7.40.3.1 operator=() . . . . .	207
7.40.3.2 render() . . . . .	207
7.41 ven::QueueFamilyIndices Struct Reference . . . . .	208
7.41.1 Detailed Description . . . . .	208
7.41.2 Member Function Documentation . . . . .	208
7.41.2.1 isComplete() . . . . .	208
7.41.3 Member Data Documentation . . . . .	209
7.41.3.1 graphicsFamily . . . . .	209
7.41.3.2 graphicsFamilyHasValue . . . . .	209
7.41.3.3 presentFamily . . . . .	209
7.41.3.4 presentFamilyHasValue . . . . .	209
7.42 ven::Renderer Class Reference . . . . .	210
7.42.1 Detailed Description . . . . .	211
7.42.2 Constructor & Destructor Documentation . . . . .	212
7.42.2.1 Renderer() [1/2] . . . . .	212
7.42.2.2 ~Renderer() . . . . .	212
7.42.2.3 Renderer() [2/2] . . . . .	212
7.42.3 Member Function Documentation . . . . .	213
7.42.3.1 beginFrame() . . . . .	213
7.42.3.2 beginSwapChainRenderPass() . . . . .	213
7.42.3.3 createCommandBuffers() . . . . .	213
7.42.3.4 endFrame() . . . . .	214
7.42.3.5 endSwapChainRenderPass() . . . . .	214

---

7.42.3.6 freeCommandBuffers() . . . . .	214
7.42.3.7 getAspectRatio() . . . . .	214
7.42.3.8 getClearColor() . . . . .	215
7.42.3.9 getCurrentCommandBuffer() . . . . .	215
7.42.3.10 getFrameIndex() . . . . .	216
7.42.3.11 getSwapChainRenderPass() . . . . .	216
7.42.3.12 getWindow() . . . . .	216
7.42.3.13 isFrameInProgress() . . . . .	217
7.42.3.14 operator=(()) . . . . .	217
7.42.3.15 recreateSwapChain() . . . . .	217
7.42.3.16 setClearValue() . . . . .	218
7.42.4 Member Data Documentation . . . . .	218
7.42.4.1 m_clearValues . . . . .	218
7.42.4.2 m_commandBuffers . . . . .	218
7.42.4.3 m_currentFrameIndex . . . . .	218
7.42.4.4 m_currentImageIndex . . . . .	219
7.42.4.5 m_device . . . . .	219
7.42.4.6 m_isFrameStarted . . . . .	219
7.42.4.7 m_swapChain . . . . .	219
7.42.4.8 m_window . . . . .	219
7.43 ven::SceneManager Class Reference . . . . .	220
7.43.1 Detailed Description . . . . .	221
7.43.2 Constructor & Destructor Documentation . . . . .	221
7.43.2.1 SceneManager() [1/3] . . . . .	221
7.43.2.2 SceneManager() [2/3] . . . . .	222
7.43.2.3 SceneManager() [3/3] . . . . .	222
7.43.3 Member Function Documentation . . . . .	222
7.43.3.1 addLight() . . . . .	222
7.43.3.2 addObject() . . . . .	222
7.43.3.3 destroyEntity() . . . . .	223
7.43.3.4 destroyLight() . . . . .	223
7.43.3.5 destroyObject() . . . . .	223
7.43.3.6 getBufferInfoForObject() . . . . .	223
7.43.3.7 getDestroyState() . . . . .	223
7.43.3.8 getLights() . . . . .	224
7.43.3.9 getObjects() . . . . .	224
7.43.3.10 getTextureDefault() . . . . .	224
7.43.3.11 getUboBuffers() . . . . .	225
7.43.3.12 operator=(()) [1/2] . . . . .	225
7.43.3.13 operator=(()) [2/2] . . . . .	225
7.43.3.14 setDestroyState() . . . . .	225
7.43.3.15 updateBuffer() . . . . .	225

---

7.43.4 Member Data Documentation . . . . .	226
7.43.4.1 m_currentLightId . . . . .	226
7.43.4.2 m_currentObjId . . . . .	226
7.43.4.3 m_destroyState . . . . .	226
7.43.4.4 m_lights . . . . .	226
7.43.4.5 m_objects . . . . .	226
7.43.4.6 m_textureDefault . . . . .	226
7.43.4.7 m_uboBuffers . . . . .	227
7.44 ven::Shaders Class Reference . . . . .	227
7.44.1 Detailed Description . . . . .	229
7.44.2 Constructor & Destructor Documentation . . . . .	229
7.44.2.1 Shaders() [1/2] . . . . .	229
7.44.2.2 ~Shaders() . . . . .	230
7.44.2.3 Shaders() [2/2] . . . . .	230
7.44.3 Member Function Documentation . . . . .	230
7.44.3.1 bind() . . . . .	230
7.44.3.2 createGraphicsPipeline() . . . . .	230
7.44.3.3 createShaderModule() . . . . .	231
7.44.3.4 defaultPipelineConfigInfo() . . . . .	231
7.44.3.5 operator=( ) . . . . .	232
7.44.3.6 readFile() . . . . .	232
7.44.4 Member Data Documentation . . . . .	232
7.44.4.1 m_device . . . . .	232
7.44.4.2 m_fragShaderModule . . . . .	232
7.44.4.3 m_graphicsPipeline . . . . .	232
7.44.4.4 m_vertShaderModule . . . . .	233
7.45 ven::SwapChain Class Reference . . . . .	233
7.45.1 Detailed Description . . . . .	236
7.45.2 Constructor & Destructor Documentation . . . . .	236
7.45.2.1 SwapChain() [1/3] . . . . .	236
7.45.2.2 SwapChain() [2/3] . . . . .	236
7.45.2.3 ~SwapChain() . . . . .	237
7.45.2.4 SwapChain() [3/3] . . . . .	237
7.45.3 Member Function Documentation . . . . .	237
7.45.3.1 acquireNextImage() . . . . .	237
7.45.3.2 chooseSwapExtent() . . . . .	237
7.45.3.3 chooseSwapPresentMode() . . . . .	237
7.45.3.4 chooseSwapSurfaceFormat() . . . . .	238
7.45.3.5 compareSwapFormats() . . . . .	238
7.45.3.6 createDepthResources() . . . . .	238
7.45.3.7 createFrameBuffers() . . . . .	238
7.45.3.8 createImageViews() . . . . .	238

---

7.45.3.9 <code>createRenderPass()</code>	238
7.45.3.10 <code>createSwapChain()</code>	238
7.45.3.11 <code>createSyncObjects()</code>	239
7.45.3.12 <code>extentAspectRatio()</code>	239
7.45.3.13 <code>findDepthFormat()</code>	239
7.45.3.14 <code>getFrameBuffer()</code>	239
7.45.3.15 <code>getImageView()</code>	239
7.45.3.16 <code>getRenderPass()</code>	239
7.45.3.17 <code>getSwapChainExtent()</code>	240
7.45.3.18 <code>getSwapChainImageFormat()</code>	240
7.45.3.19 <code>height()</code>	240
7.45.3.20 <code>imageCount()</code>	240
7.45.3.21 <code>init()</code>	240
7.45.3.22 <code>operator=()</code>	241
7.45.3.23 <code>submitCommandBuffers()</code>	241
7.45.3.24 <code>width()</code>	241
7.45.4 Member Data Documentation	241
7.45.4.1 <code>m_currentFrame</code>	241
7.45.4.2 <code>m_depthImageMemory</code>	241
7.45.4.3 <code>m_depthImages</code>	241
7.45.4.4 <code>m_depthImageViews</code>	242
7.45.4.5 <code>m_device</code>	242
7.45.4.6 <code>m_imageAvailableSemaphores</code>	242
7.45.4.7 <code>m_imagesInFlight</code>	242
7.45.4.8 <code>m_inFlightFences</code>	242
7.45.4.9 <code>m_oldSwapChain</code>	242
7.45.4.10 <code>m_renderFinishedSemaphores</code>	243
7.45.4.11 <code>m_renderPass</code>	243
7.45.4.12 <code>m_swapChain</code>	243
7.45.4.13 <code>m_swapChainDepthFormat</code>	243
7.45.4.14 <code>m_swapChainExtent</code>	243
7.45.4.15 <code>m_swapChainFrameBuffers</code>	243
7.45.4.16 <code>m_swapChainImageFormat</code>	244
7.45.4.17 <code>m_swapChainImages</code>	244
7.45.4.18 <code>m_swapChainImageViews</code>	244
7.45.4.19 <code>m_windowExtent</code>	244
7.46 <code>ven::SwapChainSupportDetails</code> Struct Reference	244
7.46.1 Detailed Description	245
7.46.2 Member Data Documentation	245
7.46.2.1 <code>capabilities</code>	245
7.46.2.2 <code>formats</code>	245
7.46.2.3 <code>presentModes</code>	245

---

7.47 ven::Texture Class Reference . . . . .	246
7.47.1 Detailed Description . . . . .	247
7.47.2 Constructor & Destructor Documentation . . . . .	248
7.47.2.1 Texture() [1/3] . . . . .	248
7.47.2.2 Texture() [2/3] . . . . .	248
7.47.2.3 ~Texture() . . . . .	249
7.47.2.4 Texture() [3/3] . . . . .	249
7.47.3 Member Function Documentation . . . . .	249
7.47.3.1 createTextureImage() . . . . .	249
7.47.3.2 createTextureImageView() . . . . .	250
7.47.3.3 createTextureSampler() . . . . .	250
7.47.3.4 getExtent() . . . . .	250
7.47.3.5 getFormat() . . . . .	251
7.47.3.6 getImage() . . . . .	251
7.47.3.7 getImageInfo() . . . . .	251
7.47.3.8 getImageLayout() . . . . .	251
7.47.3.9 getView() . . . . .	251
7.47.3.10 imageView() . . . . .	251
7.47.3.11 operator=( ) . . . . .	252
7.47.3.12 sampler() . . . . .	252
7.47.3.13 transitionLayout() . . . . .	252
7.47.3.14 updateDescriptor() . . . . .	252
7.47.4 Member Data Documentation . . . . .	252
7.47.4.1 m_descriptor . . . . .	252
7.47.4.2 m_device . . . . .	253
7.47.4.3 m_extent . . . . .	253
7.47.4.4 m_format . . . . .	253
7.47.4.5 m_layerCount . . . . .	253
7.47.4.6 m_mipLevels . . . . .	253
7.47.4.7 m_textureImage . . . . .	253
7.47.4.8 m_textureImageMemory . . . . .	253
7.47.4.9 m_textureImageView . . . . .	254
7.47.4.10 m_textureLayout . . . . .	254
7.47.4.11 m_textureSampler . . . . .	254
7.48 ven::TextureFactory Class Reference . . . . .	254
7.48.1 Detailed Description . . . . .	255
7.48.2 Constructor & Destructor Documentation . . . . .	255
7.48.2.1 TextureFactory() [1/3] . . . . .	255
7.48.2.2 ~TextureFactory() . . . . .	255
7.48.2.3 TextureFactory() [2/3] . . . . .	255
7.48.2.4 TextureFactory() [3/3] . . . . .	255
7.48.3 Member Function Documentation . . . . .	256

---

7.48.3.1 <code>create()</code>	256
7.48.3.2 <code>loadAll()</code>	256
7.48.3.3 <code>operator=()</code> [1/2]	257
7.48.3.4 <code>operator=()</code> [2/2]	257
7.49 <code>ven::Transform3D</code> Class Reference	257
7.49.1 Detailed Description	258
7.49.2 Member Function Documentation	258
7.49.2.1 <code>normalMatrix()</code>	258
7.49.2.2 <code>transformMatrix()</code>	258
7.49.3 Member Data Documentation	259
7.49.3.1 <code>rotation</code>	259
7.49.3.2 <code>scale</code>	259
7.49.3.3 <code>translation</code>	259
7.50 <code>ven::Model::Vertex</code> Struct Reference	259
7.50.1 Detailed Description	260
7.50.2 Member Function Documentation	260
7.50.2.1 <code>getAttributeDescriptions()</code>	260
7.50.2.2 <code>getBindingDescriptions()</code>	261
7.50.2.3 <code>operator==()</code>	261
7.50.3 Member Data Documentation	261
7.50.3.1 <code>color</code>	261
7.50.3.2 <code>normal</code>	261
7.50.3.3 <code>position</code>	262
7.50.3.4 <code>uv</code>	262
7.51 <code>ven::Window</code> Class Reference	262
7.51.1 Detailed Description	263
7.51.2 Constructor & Destructor Documentation	263
7.51.2.1 <code>Window()</code> [1/2]	263
7.51.2.2 <code>~Window()</code>	264
7.51.2.3 <code>Window()</code> [2/2]	264
7.51.3 Member Function Documentation	264
7.51.3.1 <code>createWindow()</code>	264
7.51.3.2 <code>createWindowSurface()</code>	264
7.51.3.3 <code>framebufferResizeCallback()</code>	265
7.51.3.4 <code>getExtent()</code>	265
7.51.3.5 <code>getGLFWWindow()</code>	266
7.51.3.6 <code>operator=()</code>	266
7.51.3.7 <code>resetWindowResizedFlag()</code>	266
7.51.3.8 <code>setFullscreen()</code>	266
7.51.3.9 <code>setWindowIcon()</code>	267
7.51.3.10 <code>wasWindowResized()</code>	267
7.51.4 Member Data Documentation	267

---

7.51.4.1 m_framebufferResized . . . . .	267
7.51.4.2 m_height . . . . .	267
7.51.4.3 m_width . . . . .	268
7.51.4.4 m_window . . . . .	268
7.52 ven::WindowConf Struct Reference . . . . .	268
7.52.1 Detailed Description . . . . .	269
7.52.2 Member Data Documentation . . . . .	269
7.52.2.1 fullscreen . . . . .	269
7.52.2.2 height . . . . .	269
7.52.2.3 width . . . . .	269
<b>8 File Documentation</b> . . . . .	<b>271</b>
8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag File Reference . . . . .	271
8.2 fragment_point_light.frag . . . . .	271
8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag File Reference . . . . .	271
8.4 fragment_shader.frag . . . . .	271
8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert File Reference . . . . .	272
8.6 vertex_point_light.vert . . . . .	272
8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference . . . . .	273
8.8 vertex_shader.vert . . . . .	273
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp File Reference . . . . .	274
8.9.1 Detailed Description . . . . .	275
8.10 Device.hpp . . . . .	275
8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Engine.hpp File Reference . . . . .	276
8.11.1 Detailed Description . . . . .	277
8.12 Engine.hpp . . . . .	278
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp File Reference . . . . .	278
8.13.1 Detailed Description . . . . .	279
8.14 EventManager.hpp . . . . .	280
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp File Reference . . . . .	280
8.15.1 Detailed Description . . . . .	281
8.16 FrameInfo.hpp . . . . .	282
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp File Reference . . . . .	282
8.17.1 Detailed Description . . . . .	284
8.18 Gui.hpp . . . . .	284
8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ABase.hpp File Reference	285
8.19.1 Detailed Description . . . . .	286
8.20 ABase.hpp . . . . .	286
8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/PointLight.hpp File Reference . . . . .	287
8.21.1 Detailed Description . . . . .	287
8.22 PointLight.hpp . . . . .	288
8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Window.hpp File Reference . . . . .	288

---

8.23.1 Detailed Description . . . . .	290
8.23.2 Macro Definition Documentation . . . . .	290
8.23.2.1 GLFW_INCLUDE_VULKAN . . . . .	290
8.24 Window.hpp . . . . .	290
8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Light.hpp File Reference . . . . .	291
8.25.1 Detailed Description . . . . .	292
8.26 Light.hpp . . . . .	292
8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp File Reference . . . . .	293
8.27.1 Detailed Description . . . . .	294
8.28 Light.hpp . . . . .	294
8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp File Reference . . . . .	295
8.29.1 Detailed Description . . . . .	295
8.30 Model.hpp . . . . .	296
8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Model.hpp File Reference . . . . .	296
8.31.1 Detailed Description . . . . .	297
8.32 Model.hpp . . . . .	297
8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/Object.hpp File Reference	298
8.33.1 Detailed Description . . . . .	299
8.34 Object.hpp . . . . .	300
8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Object.hpp File Reference . . . . .	300
8.35.1 Detailed Description . . . . .	301
8.36 Object.hpp . . . . .	301
8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp File Reference . . . . .	302
8.37.1 Detailed Description . . . . .	303
8.38 Object.hpp . . . . .	303
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Texture.hpp File Reference . . . . .	304
8.39.1 Detailed Description . . . . .	305
8.40 Texture.hpp . . . . .	305
8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp File Reference . . . . .	306
8.41.1 Detailed Description . . . . .	307
8.42 Texture.hpp . . . . .	307
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Generated/Version.hpp File Reference . . . . .	308
8.43.1 Macro Definition Documentation . . . . .	308
8.43.1.1 PROJECT_VERSION . . . . .	308
8.43.1.2 PROJECT_VERSION_MAJOR . . . . .	308
8.43.1.3 PROJECT_VERSION_MINOR . . . . .	309
8.43.1.4 PROJECT_VERSION_PATCH . . . . .	309
8.44 Version.hpp . . . . .	309
8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Buffer.hpp File Reference . . . . .	309
8.45.1 Detailed Description . . . . .	310
8.46 Buffer.hpp . . . . .	310
8.47 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp File Reference . . . . .	312

---

8.47.1 Detailed Description . . . . .	313
8.48 Pool.hpp . . . . .	314
8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp File Reference . . . . .	315
8.49.1 Detailed Description . . . . .	316
8.50 SetLayout.hpp . . . . .	316
8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp File Reference . . . . .	317
8.51.1 Detailed Description . . . . .	318
8.52 Writer.hpp . . . . .	318
8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Renderer.hpp File Reference . . . . .	318
8.53.1 Detailed Description . . . . .	320
8.54 Renderer.hpp . . . . .	320
8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp File Reference . . . . .	321
8.55.1 Detailed Description . . . . .	322
8.56 Shaders.hpp . . . . .	322
8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp File Reference . . . . .	323
8.57.1 Detailed Description . . . . .	324
8.58 SwapChain.hpp . . . . .	324
8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Camera.hpp File Reference . . . . .	326
8.59.1 Detailed Description . . . . .	327
8.60 Camera.hpp . . . . .	327
8.61 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp File Reference . . . . .	328
8.61.1 Detailed Description . . . . .	329
8.62 Manager.hpp . . . . .	329
8.63 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp File Reference . . . . .	330
8.63.1 Detailed Description . . . . .	331
8.64 Transform3D.hpp . . . . .	331
8.65 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp File Reference . . . . .	331
8.65.1 Detailed Description . . . . .	332
8.66 Clock.hpp . . . . .	333
8.67 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp File Reference . . . . .	333
8.68 Colors.hpp . . . . .	335
8.69 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Config.hpp File Reference . . . . .	337
8.70 Config.hpp . . . . .	338
8.71 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp File Reference . . . . .	339
8.72 HashCombine.hpp . . . . .	340
8.73 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp File Reference . . . . .	340
8.73.1 Detailed Description . . . . .	341
8.74 Logger.hpp . . . . .	341
8.75 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Message.hpp File Reference . . . . .	342
8.76 Message.hpp . . . . .	343
8.77 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Parser.hpp File Reference . . . . .	344
8.77.1 Detailed Description . . . . .	345

---

8.78 Parser.hpp . . . . .	345
8.79 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp File Reference . . . . .	347
8.79.1 Detailed Description . . . . .	348
8.80 Utils.hpp . . . . .	349
8.81 /home/runner/work/VEngine/VEngine/README.md File Reference . . . . .	349
8.82 /home/runner/work/VEngine/VEngine/src/Core/device.cpp File Reference . . . . .	349
8.82.1 Function Documentation . . . . .	350
8.82.1.1 CreateDebugUtilsMessengerEXT() . . . . .	350
8.82.1.2 debugCallback() . . . . .	350
8.82.1.3 DestroyDebugUtilsMessengerEXT() . . . . .	351
8.83 device.cpp . . . . .	351
8.84 /home/runner/work/VEngine/VEngine/src/Core/engine.cpp File Reference . . . . .	358
8.85 engine.cpp . . . . .	359
8.86 /home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp File Reference . . . . .	361
8.86.1 Macro Definition Documentation . . . . .	362
8.86.1.1 GLM_ENABLE_EXPERIMENTAL . . . . .	362
8.87 eventManager.cpp . . . . .	362
8.88 /home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp File Reference . . . . .	363
8.89 init.cpp . . . . .	363
8.90 /home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp File Reference . . . . .	364
8.91 render.cpp . . . . .	365
8.92 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp File Reference . . . . .	370
8.93 base.cpp . . . . .	370
8.94 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp File Reference . . . . .	371
8.95 object.cpp . . . . .	371
8.96 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp File Reference . . . . .	372
8.97 pointLight.cpp . . . . .	372
8.98 /home/runner/work/VEngine/VEngine/src/Core/window.cpp File Reference . . . . .	373
8.98.1 Macro Definition Documentation . . . . .	373
8.98.1.1 STB_IMAGE_IMPLEMENTATION . . . . .	373
8.99 window.cpp . . . . .	373
8.100 /home/runner/work/VEngine/VEngine/src/Factories/Light.cpp File Reference . . . . .	375
8.101 Light.cpp . . . . .	375
8.102 /home/runner/work/VEngine/VEngine/src/Factories/Model.cpp File Reference . . . . .	376
8.103 Model.cpp . . . . .	376
8.104 /home/runner/work/VEngine/VEngine/src/Factories/Object.cpp File Reference . . . . .	377
8.105 Object.cpp . . . . .	377
8.106 /home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp File Reference . . . . .	377
8.107 Texture.cpp . . . . .	378
8.108 /home/runner/work/VEngine/VEngine/src/Gfx/buffer.cpp File Reference . . . . .	378
8.109 buffer.cpp . . . . .	379
8.110 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp File Reference . . . . .	380

---

8.111 pool.cpp . . . . .	380
8.112 /home/runner/work/VEngine/VEngine/src/Gfx/DescriptorssetLayout.cpp File Reference . . . . .	381
8.113 setLayout.cpp . . . . .	381
8.114 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptorswriter.cpp File Reference . . . . .	382
8.115 writer.cpp . . . . .	383
8.116 /home/runner/work/VEngine/VEngine/src/Gfx/model.cpp File Reference . . . . .	384
8.116.1 Macro Definition Documentation . . . . .	385
8.116.1.1 GLM_ENABLE_EXPERIMENTAL . . . . .	385
8.117 model.cpp . . . . .	385
8.118 /home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp File Reference . . . . .	387
8.119 renderer.cpp . . . . .	387
8.120 /home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp File Reference . . . . .	389
8.121 shaders.cpp . . . . .	389
8.122 /home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp File Reference . . . . .	392
8.123 swapChain.cpp . . . . .	392
8.124 /home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp File Reference . . . . .	397
8.125 texture.cpp . . . . .	397
8.126 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference . . . . .	401
8.126.1 Function Documentation . . . . .	402
8.126.1.1 main() . . . . .	402
8.127 main.cpp . . . . .	402
8.128 /home/runner/work/VEngine/VEngine/src/Scene/camera.cpp File Reference . . . . .	403
8.129 camera.cpp . . . . .	403
8.130 /home/runner/work/VEngine/VEngine/src/Scene/manager.cpp File Reference . . . . .	405
8.131 manager.cpp . . . . .	405
8.132 /home/runner/work/VEngine/VEngine/src/Utils/clock.cpp File Reference . . . . .	406
8.133 clock.cpp . . . . .	406
8.134 /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp File Reference . . . . .	407
8.135 logger.cpp . . . . .	407
8.136 /home/runner/work/VEngine/VEngine/src/Utils/parser.cpp File Reference . . . . .	408
8.137 parser.cpp . . . . .	408

# Chapter 1

## vengine

### 1.1 VEngine - Vulkan based Game Engine

#### WORK IN PROGRESS!

Welcome to **VEngine**, a Vulkan-based game engine.

I Build this project to learn more about Vulkan and graphics programming in general. The goal is to create an efficient engine that can be used for various projects, such as games, simulations, and visualizations.

#### 1.1.1 Features

- **Vulkan Rendering Pipeline:** Leveraging Vulkan for high-performance graphics rendering
- **Basic Camera System:** Control camera movement in the 3D space
- **Model Loading:** Import 3D models using `assimp`
- **Real-time debugging:** Use ImGui for real-time debugging and tool development
- **Cross-platform support** (Linux, Windows)
- **Doxygen Documentation:** Automatically generated documentation hosted on [GitHub Pages](#)

#### 1.1.2 Planned Features:

- **Ray Tracing**
- **Physics Integration**
- **Audio Integration**
- Support for more **input devices** (e.g., mouse, game controller)
- Support for more **platforms** (e.g., macOS, Android, iOS, PS5 ...)

### 1.1.3 Build

Before building the project, make sure you update the submodules:

```
git submodule update --init --recursive
```

#### 1.1.3.1 Prerequisites

Make sure you have the following dependencies installed on your system:

- [CMake 3.27](#)
- [C++20](#)
- [Vulkan SDK](#)
- [X11 \(Linux only\)](#)
- [LLVM](#)

If you are using a Debian-based distribution, you can install the dependencies using the following command:

```
./tools/install-dependencies.sh build
```

#### 1.1.3.2 Linux

##### 1.1.3.2.1 Build and Run

```
./tools/build.sh build
```

This script also handle several other commands: tests, format and doc.

Then you can run the engine:

```
./vengine [options]
```

#### 1.1.3.3 Windows

##### 1.1.3.3.1 Build and Run

I build the project with [CLion](#), also works with [Visual Studio](#). I'm using the Visual studio toolchain with [ninja](#).

You should create your own CMake profile depending on your configuration. Basic configuration should run the following commands:

```
cmake -G "Visual Studio 17 2022" .
cmake --build . --config Release
```

Then you can run the engine:

```
.\vengine.exe [options]
```

#### 1.1.3.4 Command Line Options

The following command-line options are available:

Option	Description
--help, -h	Show help message and exit
--version, -v	Show version information and exit
--fullscreen, -f	Enable fullscreen mode
--vsync, -V	Enable vertical sync
--width <value>	Set the width of the window (e.g., 800)
--height <value>	Set the height of the window (e.g., 600)
--fov <value>	Set the field of view (1.0 to 300.0)
--mspeed <value>	Set the move speed (0.1 to 100.0)
--lspeed <value>	Set the look speed (0.1 to 100.0)
--near <value>	Set the near plane (0.1 to 100.0)
--far <value>	Set the far plane (0.1 to 100.0)

#### 1.1.4 Key Bindings

The following keyboard controls are currently available for interacting with the engine:

Key	Description
Z	Move forward
S	Move backward
Q	Move left
D	Move right
SHIFT	Move down
SPACE	Move up
↑	Look up
↓	Look down
←	Look left
→	Look right
à	Show debug windows

#### 1.1.5 Documentation

The documentation is generated using [Doxygen](#). You can access the latest version on the [GitHub Pages](#).

To generate the documentation locally, run the following command:

```
./tools/build.sh doc
```

The generated documentation will be available in the `docs` directory.

You can also run the following command to host the documentation locally:

```
./tools/run-doc.sh [ node | php | python ]
```

Then you can access the documentation at <http://localhost:8080>.

## 1.1.6 External Libraries

- **Assimp**: Open Asset Import Library to load various 3D model formats into the engine.
- **Doxygen Awesome CSS**: A custom CSS theme for Doxygen documentation.
- **GLFW**: For creating windows, receiving input, and managing OpenGL and Vulkan contexts.
- **GLM**: A header-only C++ mathematics library for 3D transformations, vectors, and matrices, compatible with OpenGL and Vulkan.
- **Google Test**: A testing framework for C++.
- **ImGui**: Immediate Mode Graphical User Interface for real-time debugging and tool development.
- **stb**: A set of single-file public domain libraries for graphics, image loading, and more.

These libraries are included directly into the project to simplify dependency management. Be sure to initialize and update the submodules when cloning the repository:

## 1.1.7 Commit Norms

Commit Type	Description
build	Changes that affect the build system or external dependencies (npm, make, etc.)
ci	Changes related to integration files and scripts or configuration (Travis, Ansible, BrowserStack, etc.)
feat	Addition of a new feature
fix	Bug fix
perf	Performance improvements
refactor	Modification that neither adds a new feature nor improves performance
style	Change that does not affect functionality or semantics (indentation, formatting, adding space, renaming a variable, etc.)
docs	Writing or updating documentation
test	Addition or modification of tests

## 1.1.8 License

This project is licensed under the MIT License - see the [LICENSE](#) file for details.

## 1.1.9 Acknowledgements

Special thanks to [Brendan Galea](#) for inspiration and resources related to Vulkan development.

## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">ven</a>	.....	15
---------------------	-------	----



# Chapter 3

## Hierarchical Index

### 3.1 Class Hierarchy

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

ven::ARenderSystemBase . . . . .	27
ven::ObjectRenderSystem . . . . .	186
ven::PointLightRenderSystem . . . . .	204
ven::Buffer . . . . .	33
ven::DescriptorPool::Builder . . . . .	46
ven::DescriptorSetLayout::Builder . . . . .	51
ven::Model::Builder . . . . .	55
ven::Camera . . . . .	57
ven::CameraConf . . . . .	68
ven::Clock . . . . .	69
ven::Gui::ClockData . . . . .	74
ven::Colors . . . . .	75
ven::Config . . . . .	87
ven::DescriptorPool . . . . .	88
ven::DescriptorSetLayout . . . . .	93
ven::DescriptorWriter . . . . .	97
ven::Device . . . . .	101
ven::Engine . . . . .	120
ven::EventManager . . . . .	126
std::exception . . . . .	
ven::ParserException . . . . .	196
ven::FrameInfo . . . . .	130
ven::Gui::funcs . . . . .	133
ven::GlobalUbo . . . . .	135
ven::Gui . . . . .	137
std::hash< ven::Model::Vertex > . . . . .	147
ven::KeyAction . . . . .	148
ven::KeyMappings . . . . .	150
ven::Light . . . . .	153
ven::LightFactory . . . . .	157
ven::LightPushConstantData . . . . .	160
ven::Logger . . . . .	161
ven::Model . . . . .	166
ven::ModelFactory . . . . .	172
ven::Object . . . . .	175

ven::ObjectBufferData . . . . .	181
ven::ObjectFactory . . . . .	182
ven::ObjectPushConstantData . . . . .	185
ven::Parser . . . . .	191
ven::PipelineConfigInfo . . . . .	199
ven::PointLightData . . . . .	203
ven::QueueFamilyIndices . . . . .	208
ven::Renderer . . . . .	210
ven::SceneManager . . . . .	220
ven::Shaders . . . . .	227
ven::SwapChain . . . . .	233
ven::SwapChainSupportDetails . . . . .	244
ven::Texture . . . . .	246
ven::TextureFactory . . . . .	254
ven::Transform3D . . . . .	257
ven::Model::Vertex . . . . .	259
ven::Window . . . . .	262
ven::WindowConf . . . . .	268

# Chapter 4

## Class Index

### 4.1 Class List

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

<a href="#">ven::ARenderSystemBase</a>	Abstract class for render system base . . . . .	27
<a href="#">ven::Buffer</a>	Class for buffer . . . . .	33
<a href="#">ven::DescriptorPool::Builder</a>	. . . . .	46
<a href="#">ven::DescriptorsetLayout::Builder</a>	. . . . .	51
<a href="#">ven::Model::Builder</a>	. . . . .	55
<a href="#">ven::Camera</a>	Class for camera . . . . .	57
<a href="#">ven::CameraConf</a>	. . . . .	68
<a href="#">ven::Clock</a>	Class for clock . . . . .	69
<a href="#">ven::Gui::ClockData</a>	. . . . .	74
<a href="#">ven::Colors</a>	Class for colors . . . . .	75
<a href="#">ven::Config</a>	. . . . .	87
<a href="#">ven::DescriptorPool</a>	Class for descriptor pool . . . . .	88
<a href="#">ven::DescriptorsetLayout</a>	Class for descriptor set layout . . . . .	93
<a href="#">ven::DescriptorWriter</a>	Class for descriptor writer . . . . .	97
<a href="#">ven::Device</a>	Class for device . . . . .	101
<a href="#">ven::Engine</a>	Class for engine . . . . .	120
<a href="#">ven::EventManager</a>	Class for event manager . . . . .	126
<a href="#">ven::FrameInfo</a>	. . . . .	130
<a href="#">ven::Gui::funcs</a>	. . . . .	133
<a href="#">ven::GlobalUbo</a>	. . . . .	135
<a href="#">ven::Gui</a>	Class for Gui . . . . .	137
<a href="#">std::hash&lt; ven::Model::Vertex &gt;</a>	. . . . .	147
<a href="#">ven::KeyAction</a>	. . . . .	148

ven::KeyMappings . . . . .	150
ven::Light	
Class for light . . . . .	153
ven::LightFactory	
Class for light factory . . . . .	157
ven::LightPushConstantData . . . . .	160
ven::Logger . . . . .	161
ven::Model	
Class for model . . . . .	166
ven::ModelFactory	
Class for Model factory . . . . .	172
ven::Object	
Class for object . . . . .	175
ven::ObjectBufferData . . . . .	181
ven::ObjectFactory	
Class for object factory . . . . .	182
ven::ObjectPushConstantData . . . . .	185
ven::ObjectRenderSystem	
Class for object render system . . . . .	186
ven::Parser	
Class for Parser . . . . .	191
ven::ParserException	
Custom exception class for parsing errors . . . . .	196
ven::PipelineConfigInfo . . . . .	199
ven::PointLightData . . . . .	203
ven::PointLightRenderSystem	
Class for point light system . . . . .	204
ven::QueueFamilyIndices . . . . .	208
ven::Renderer	
Class for renderer . . . . .	210
ven::SceneManager	
Class for object manager . . . . .	220
ven::Shaders	
Class for shaders . . . . .	227
ven::SwapChain	
Class for swap chain . . . . .	233
ven::SwapChainSupportDetails . . . . .	244
ven::Texture	
Class for texture . . . . .	246
ven::TextureFactory	
Class for Texture factory . . . . .	254
ven::Transform3D	
Class for 3D transformation . . . . .	257
ven::Model::Vertex . . . . .	259
ven::Window	
Class for window . . . . .	262
ven::WindowConf . . . . .	268

# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/VEngine/VEngine/assets/shaders/ <a href="#">fragment_point_light.frag</a>	271
/home/runner/work/VEngine/VEngine/assets/shaders/ <a href="#">fragment_shader.frag</a>	271
/home/runner/work/VEngine/VEngine/assets/shaders/ <a href="#">vertex_point_light.vert</a>	272
/home/runner/work/VEngine/VEngine/assets/shaders/ <a href="#">vertex_shader.vert</a>	273
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">Device.hpp</a>	
This file contains the Device class	274
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">Engine.hpp</a>	
This file contains the Engine class	276
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">EventManager.hpp</a>	
This file contains the EventManager class	278
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">FrameInfo.hpp</a>	
This file contains the FrameInfo class	280
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">Gui.hpp</a>	
This file contains the ImGuiWindowManager class	282
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ <a href="#">Window.hpp</a>	
This file contains the Window class	288
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ <a href="#">ABase.hpp</a>	
This file contains the ARenderSystemBase class	285
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ <a href="#">Object.hpp</a>	
This file contains the ObjectRenderSystem class	298
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ <a href="#">PointLight.hpp</a>	
This file contains the PointLightRenderSystem class	287
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ <a href="#">Light.hpp</a>	
This file contains the Light Factory class	291
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ <a href="#">Model.hpp</a>	
This file contains the Model Factory class	295
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ <a href="#">Object.hpp</a>	
This file contains the Object Factory class	300
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ <a href="#">Texture.hpp</a>	
This file contains the Texture Factory class	304
/home/runner/work/VEngine/VEngine/include/VEngine/Generated/ <a href="#">Version.hpp</a>	308
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/ <a href="#">Buffer.hpp</a>	
This file contains the Buffer class	309
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/ <a href="#">Model.hpp</a>	
This file contains the Model class	296

/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Renderer.hpp This file contains the Renderer class . . . . .	318
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp This file contains the Shader class . . . . .	321
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp This file contains the SwapChain class . . . . .	323
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp This file contains the Texture class . . . . .	306
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp This file contains the DescriptorPool class . . . . .	312
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp This file contains the SetLayout class . . . . .	315
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp This file contains the DescriptorsWriter class . . . . .	317
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Camera.hpp This file contains the Camera class . . . . .	326
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp This file contains the SceneManager class . . . . .	328
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp This file contains the Transform3D class . . . . .	330
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp This file contains the Light class . . . . .	293
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp This file contains the Object class . . . . .	302
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp This file contains the Clock class . . . . .	331
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp . . . . .	333
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Config.hpp . . . . .	337
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp . . . . .	339
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp This file contains the Logger class . . . . .	340
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Message.hpp . . . . .	342
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Parser.hpp This file contains the Parser class . . . . .	344
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp This file contains utils for VEngine . . . . .	347
/home/runner/work/VEngine/VEngine/src/main.cpp . . . . .	401
/home/runner/work/VEngine/VEngine/src/Core/device.cpp . . . . .	349
/home/runner/work/VEngine/VEngine/src/Core/engine.cpp . . . . .	358
/home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp . . . . .	361
/home/runner/work/VEngine/VEngine/src/Core/window.cpp . . . . .	373
/home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp . . . . .	363
/home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp . . . . .	364
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp . . . . .	370
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp . . . . .	371
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp . . . . .	372
/home/runner/work/VEngine/VEngine/src/Factories/Light.cpp . . . . .	375
/home/runner/work/VEngine/VEngine/src/Factories/Model.cpp . . . . .	376
/home/runner/work/VEngine/VEngine/src/Factories/Object.cpp . . . . .	377
/home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp . . . . .	377
/home/runner/work/VEngine/VEngine/src/Gfx/buffer.cpp . . . . .	378
/home/runner/work/VEngine/VEngine/src/Gfx/model.cpp . . . . .	384
/home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp . . . . .	387
/home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp . . . . .	389
/home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp . . . . .	392
/home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp . . . . .	397
/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp . . . . .	380
/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/setLayout.cpp . . . . .	381

/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/ <a href="#">writer.cpp</a>	382
/home/runner/work/VEngine/VEngine/src/Scene/ <a href="#">camera.cpp</a>	403
/home/runner/work/VEngine/VEngine/src/Scene/ <a href="#">manager.cpp</a>	405
/home/runner/work/VEngine/VEngine/src/Utils/ <a href="#">clock.cpp</a>	406
/home/runner/work/VEngine/VEngine/src/Utils/ <a href="#">logger.cpp</a>	407
/home/runner/work/VEngine/VEngine/src/Utils/ <a href="#">parser.cpp</a>	408



# Chapter 6

## Namespace Documentation

### 6.1 ven Namespace Reference

#### Classes

- class [ARenderSystemBase](#)  
*Abstract class for render system base.*
- class [Buffer](#)  
*Class for buffer.*
- class [Camera](#)  
*Class for camera.*
- struct [CameraConf](#)
- class [Clock](#)  
*Class for clock.*
- class [Colors](#)  
*Class for colors.*
- struct [Config](#)
- class [DescriptorPool](#)  
*Class for descriptor pool.*
- class [DescriptorsetLayout](#)  
*Class for descriptor set layout.*
- class [DescriptorWriter](#)  
*Class for descriptor writer.*
- class [Device](#)  
*Class for device.*
- class [Engine](#)  
*Class for engine.*
- class [EventManager](#)  
*Class for event manager.*
- struct [FrameInfo](#)
- struct [GlobalUbo](#)
- class [Gui](#)  
*Class for Gui.*
- struct [KeyAction](#)
- struct [KeyMappings](#)
- class [Light](#)

- class [LightFactory](#)  
*Class for light factory.*
- struct [LightPushConstantData](#)
- class [Logger](#)
- class [Model](#)  
*Class for model.*
- class [ModelFactory](#)  
*Class for Model factory.*
- class [Object](#)  
*Class for object.*
- struct [ObjectBufferData](#)
- class [ObjectFactory](#)  
*Class for object factory.*
- struct [ObjectPushConstantData](#)
- class [ObjectRenderSystem](#)  
*Class for object render system.*
- class [Parser](#)  
*Class for Parser.*
- class [ParserException](#)  
*Custom exception class for parsing errors.*
- struct [PipelineConfigInfo](#)
- struct [PointLightData](#)
- class [PointLightRenderSystem](#)  
*Class for point light system.*
- struct [QueueFamilyIndices](#)
- class [Renderer](#)  
*Class for renderer.*
- class [SceneManager](#)  
*Class for object manager.*
- class [Shaders](#)  
*Class for shaders.*
- class [SwapChain](#)  
*Class for swap chain.*
- struct [SwapChainSupportDetails](#)
- class [Texture](#)  
*Class for texture.*
- class [TextureFactory](#)  
*Class for Texture factory.*
- class [Transform3D](#)  
*Class for 3D transformation.*
- class [Window](#)  
*Class for window.*
- struct [WindowConf](#)

## Typedefs

- using [TimePoint](#) = std::chrono::time\_point<std::chrono::high\_resolution\_clock>

## Enumerations

- enum `GUI_STATE` : `uint8_t` { `SHOW_EDITOR` = 0 , `SHOW_PLAYER` = 1 , `HIDDEN` = 2 }
- enum class `LogLevel` : `uint8_t` { `INFO` , `WARNING` }
- enum `ENGINE_STATE` : `uint8_t` { `EDITOR` = 0 , `PLAYER` = 1 , `PAUSED` = 2 , `EXIT` = 3 }

## Functions

- template<typename T , typename... Rest>  
void `hashCombine` (`std::size_t &seed`, `const T &v`, `const Rest &... rest`)
- bool `isNumeric` (`const std::string_view str`)

## Variables

- static constexpr float `EPSILON` = `std::numeric_limits<float>::epsilon()`
- static constexpr `KeyMappings` `DEFAULT_KEY_MAPPINGS` {}
- static constexpr float `DEFAULT_AMBIENT_LIGHT_INTENSITY` = .2F
- static constexpr `glm::vec4` `DEFAULT_AMBIENT_LIGHT_COLOR` = {`glm::vec3(1.F)`, `DEFAULT_AMBIENT_LIGHT_INTENSITY`}
- static constexpr `uint16_t` `DESCRIPTOR_COUNT` = 1000
- static constexpr `std::string_view` `DEFAULT_TITLE` = "VEngine"
- static constexpr `uint32_t` `DEFAULT_WIDTH` = 1920
- static constexpr `uint32_t` `DEFAULT_HEIGHT` = 1080
- static constexpr `Transform3D` `DEFAULT_TRANSFORM` = {`.translation = {0.F, 0.F, 0.F}`, `.scale = {0.1F, 0.F, 0.F}`, `.rotation = {0.F, 0.F, 0.F}`}
- static constexpr `uint32_t` `DEFAULT_MAX_SETS` = 1000
- static constexpr `VkClearColorValue` `DEFAULT_CLEAR_COLOR` = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr `VkClearDepthStencilValue` `DEFAULT_CLEAR_DEPTH` = {1.0F, 0}
- static constexpr `std::string_view` `SHADERS_BIN_PATH` = "build/shaders/"
- static constexpr int `MAX_FRAMES_IN_FLIGHT` = 2
- static constexpr `glm::vec3` `DEFAULT_POSITION` {0.F, 0.F, -2.5F}
- static constexpr `glm::vec3` `DEFAULT_ROTATION` {0.F, 0.F, 0.F}
- static constexpr float `DEFAULT_FOV` = `glm::radians(50.0F)`
- static constexpr float `DEFAULT_NEAR` = 0.1F
- static constexpr float `DEFAULT_FAR` = 100.F
- static constexpr float `DEFAULT_MOVE_SPEED` = 3.F
- static constexpr float `DEFAULT_LOOK_SPEED` = 1.5F
- static constexpr float `DEFAULT_LIGHT_INTENSITY` = .2F
- static constexpr float `DEFAULT_LIGHT_RADIUS` = 0.1F
- static constexpr float `DEFAULT_SHININESS` = 32.F
- static constexpr `glm::vec4` `DEFAULT_LIGHT_COLOR` = {`glm::vec3(1.F)`, `DEFAULT_LIGHT_INTENSITY`}
- static constexpr `uint8_t` `MAX_LIGHTS` = 10
- static constexpr `uint16_t` `MAX_OBJECTS` = 1000
- static constexpr float `COLOR_MAX` = 255.0F
- constexpr auto `VERSION_MESSAGE`
- constexpr auto `HELP_MESSAGE`
- static const `std::unordered_map< std::string_view, std::function< void(Config &conf, std::string_view arg)>` > `FUNCTION_MAP_OPT_LONG`
- static const `std::unordered_map< std::string_view, std::function< void(Config &conf)>` > `FUNCTION_MAP_OPT_SHORT`

## 6.1.1 Typedef Documentation

### 6.1.1.1 TimePoint

```
using ven::TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>
```

Definition at line 13 of file [Clock.hpp](#).

## 6.1.2 Enumeration Type Documentation

### 6.1.2.1 ENGINE\_STATE

```
enum ven::ENGINE_STATE : uint8_t
```

Enumerator

EDITOR	
PLAYER	
PAUSED	
EXIT	

Definition at line 13 of file [Utils.hpp](#).

### 6.1.2.2 GUI\_STATE

```
enum ven::GUI_STATE : uint8_t
```

Enumerator

SHOW_EDITOR	
SHOW_PLAYER	
HIDDEN	

Definition at line 19 of file [Gui.hpp](#).

### 6.1.2.3 LogLevel

```
enum class ven::LogLevel : uint8_t [strong]
```

Enumerator

INFO	
WARNING	

Definition at line 23 of file [Logger.hpp](#).

### 6.1.3 Function Documentation

#### 6.1.3.1 hashCombine()

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

Definition at line 14 of file [HashCombine.hpp](#).

References [hashCombine\(\)](#).

Referenced by [hashCombine\(\)](#), and [std::hash< ven::Model::Vertex >::operator\(\)\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.1.3.2 isNumeric()

```
bool ven::isNumeric (
    const std::string_view str) [inline]
```

Definition at line 44 of file [Parser.hpp](#).

## 6.1.4 Variable Documentation

### 6.1.4.1 COLOR\_MAX

```
float ven::COLOR_MAX = 255.0F [static], [constexpr]
```

Definition at line 15 of file [Colors.hpp](#).

### 6.1.4.2 DEFAULT\_AMBIENT\_LIGHT\_COLOR

```
glm::vec4 ven::DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENSITY}  
[static], [constexpr]
```

Definition at line 19 of file [FrameInfo.hpp](#).

### 6.1.4.3 DEFAULT\_AMBIENT\_LIGHT\_INTENSITY

```
float ven::DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F [static], [constexpr]
```

Definition at line 18 of file [FrameInfo.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

### 6.1.4.4 DEFAULT\_CLEAR\_COLOR

```
VkClearColorValue ven::DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 16 of file [Renderer.hpp](#).

### 6.1.4.5 DEFAULT\_CLEAR\_DEPTH

```
VkClearDepthStencilValue ven::DEFAULT_CLEAR_DEPTH = {1.0F, 0} [static], [constexpr]
```

Definition at line 17 of file [Renderer.hpp](#).

### 6.1.4.6 DEFAULT\_FAR

```
float ven::DEFAULT_FAR = 100.F [static], [constexpr]
```

Definition at line 18 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.7 DEFAULT\_FOV

```
float ven::DEFAULT_FOV = glm::radians(50.0F) [static], [constexpr]
```

Definition at line 16 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.8 DEFAULT\_HEIGHT

```
uint32_t ven::DEFAULT_HEIGHT = 1080 [static], [constexpr]
```

Definition at line 19 of file [Window.hpp](#).

#### 6.1.4.9 DEFAULT\_KEY\_MAPPINGS

```
KeyMappings ven::DEFAULT_KEY_MAPPINGS {} [static], [constexpr]
```

Definition at line 35 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::handleEvents\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

#### 6.1.4.10 DEFAULT\_LIGHT\_COLOR

```
glm::vec4 ven::DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY} [static], [constexpr]
```

Definition at line 19 of file [Light.hpp](#).

#### 6.1.4.11 DEFAULT\_LIGHT\_INTENSITY

```
float ven::DEFAULT_LIGHT_INTENSITY = .2F [static], [constexpr]
```

Definition at line 16 of file [Light.hpp](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

#### 6.1.4.12 DEFAULT\_LIGHT\_RADIUS

```
float ven::DEFAULT_LIGHT_RADIUS = 0.1F [static], [constexpr]
```

Definition at line 17 of file [Light.hpp](#).

#### 6.1.4.13 DEFAULT\_LOOK\_SPEED

```
float ven::DEFAULT_LOOK_SPEED = 1.5F [static], [constexpr]
```

Definition at line 21 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.14 DEFAULT\_MAX\_SETS

```
uint32_t ven::DEFAULT_MAX_SETS = 1000 [static], [constexpr]
```

Definition at line 15 of file [Pool.hpp](#).

#### 6.1.4.15 DEFAULT\_MOVE\_SPEED

```
float ven::DEFAULT_MOVE_SPEED = 3.F [static], [constexpr]
```

Definition at line 20 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.16 DEFAULT\_NEAR

```
float ven::DEFAULT_NEAR = 0.1F [static], [constexpr]
```

Definition at line 17 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.17 DEFAULT\_POSITION

```
glm::vec3 ven::DEFAULT_POSITION {0.F, 0.F, -2.5F} [static], [constexpr]
```

Definition at line 13 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.18 DEFAULT\_ROTATION

```
glm::vec3 ven::DEFAULT_ROTATION {0.F, 0.F, 0.F} [static], [constexpr]
```

Definition at line 14 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

#### 6.1.4.19 DEFAULT\_SHININESS

```
float ven::DEFAULT_SHININESS = 32.F [static], [constexpr]
```

Definition at line 18 of file [Light.hpp](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

#### 6.1.4.20 DEFAULT\_TITLE

```
std::string_view ven::DEFAULT_TITLE = "VEngine" [static], [constexpr]
```

Definition at line 17 of file [Window.hpp](#).

#### 6.1.4.21 DEFAULT\_TRANSFORM

```
Transform3D ven::DEFAULT_TRANSFORM = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F, 0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}} [static], [constexpr]
```

Definition at line 15 of file [Light.hpp](#).

#### 6.1.4.22 DEFAULT\_WIDTH

```
uint32_t ven::DEFAULT_WIDTH = 1920 [static], [constexpr]
```

Definition at line 18 of file [Window.hpp](#).

#### 6.1.4.23 DESCRIPTOR\_COUNT

```
uint16_t ven::DESCRIPTOR_COUNT = 1000 [static], [constexpr]
```

Definition at line 17 of file [Gui.hpp](#).

Referenced by [ven::Gui::init\(\)](#).

#### 6.1.4.24 EPSILON

```
float ven::EPSILON = std::numeric_limits<float>::epsilon() [static], [constexpr]
```

Definition at line 34 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

#### 6.1.4.25 FUNCTION\_MAP\_OPT\_LONG

```
const std::unordered_map<std::string_view, std::function<void(Config& conf, std::string_view arg)>> ven::FUNCTION_MAP_OPT_LONG [static]
```

Definition at line 46 of file [Parser.hpp](#).

Referenced by [ven::Parser::handleLongOption\(\)](#), and [ven::Parser::isValidOption\(\)](#).

### 6.1.4.26 FUNCTION\_MAP\_OPT\_SHORT

```
const std::unordered_map<std::string_view, std::function<void(Config& conf)>> ven::FUNCTION_MAP_OPT_SHORT [static]
```

**Initial value:**

```
= {
    { "h", [](Config& conf) { std::cout << HELP_MESSAGE; throw ParserException(""); } },
    { "v", [](Config& conf) { std::cout << VERSION_MESSAGE; throw ParserException(""); } },
    { "f", [](Config& conf) { conf.window.fullscreen = true; } },
    { "V", [](Config& conf) { conf.vsync = true; } }
}
```

Definition at line 122 of file [Parser.hpp](#).

Referenced by [ven::Parser::handleShortOptions\(\)](#), and [ven::Parser::isValidOption\(\)](#).

### 6.1.4.27 HELP\_MESSAGE

```
auto ven::HELP_MESSAGE [constexpr]
```

**Initial value:**

```
= "Usage: VEngine [options]\n"
"Options:\n"
"  --help, -h           Show this help message and exit\n"
"  --version, -v         Show version information and exit\n"
"  --fullscreen, -f       Enable fullscreen mode\n"
"  --vsync, -V            Enable vertical sync\n"
"  --width <value>        Set the width of the window (e.g., 800)\n"
"  --height <value>       Set the height of the window (e.g., 600)\n"
"  --fov <value>          Set the field of view (1.0 to 300.0)\n"
"  --mspeed <value>       Set the move speed (0.1 to 100.0)\n"
"  --lspeed <value>       Set the look speed (0.1 to 100.0)\n"
"  --near <value>         Set the near plane (0.1 to 100.0)\n"
"  --far <value>          Set the far plane (0.1 to 100.0)\n"
```

Definition at line 21 of file [Message.hpp](#).

### 6.1.4.28 MAX\_FRAMES\_IN\_FLIGHT

```
int ven::MAX_FRAMES_IN_FLIGHT = 2 [static], [constexpr]
```

Definition at line 15 of file [SwapChain.hpp](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::SwapChain::createSyncObjects\(\)](#), [ven::Renderer::endFrame\(\)](#), [ven::Engine::Engine\(\)](#), [ven::Engine::mainLoop\(\)](#), [ven::SwapChain::submitCommandBuffers\(\)](#), and [ven::SwapChain::~SwapChain\(\)](#).

### 6.1.4.29 MAX\_LIGHTS

```
uint8_t ven::MAX_LIGHTS = 10 [static], [constexpr]
```

Definition at line 21 of file [Light.hpp](#).

Referenced by [ven::LightFactory::create\(\)](#).

#### 6.1.4.30 MAX\_OBJECTS

```
uint16_t ven::MAX_OBJECTS = 1000 [static], [constexpr]
```

Definition at line 17 of file [Object.hpp](#).

Referenced by [ven::ObjectFactory::create\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

#### 6.1.4.31 SHADERS\_BIN\_PATH

```
std::string_view ven::SHADERS_BIN_PATH = "build/shaders/" [static], [constexpr]
```

Definition at line 13 of file [Shaders.hpp](#).

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

#### 6.1.4.32 VERSION\_MESSAGE

```
auto ven::VERSION_MESSAGE [constexpr]
```

##### Initial value:

```
= "VEngine Version " PROJECT_VERSION "\n"
    "Built on " __DATE__
    " at " __TIME__
    "\nAuthor: Elliot Masina (bobis33)\n"
    "License: MIT\n"
    "Repository: https://github.com/bobis33/VEngine\n"
    "Documentation: https://bobis33.github.io/VEngine/\n"
```

Definition at line 13 of file [Message.hpp](#).



# Chapter 7

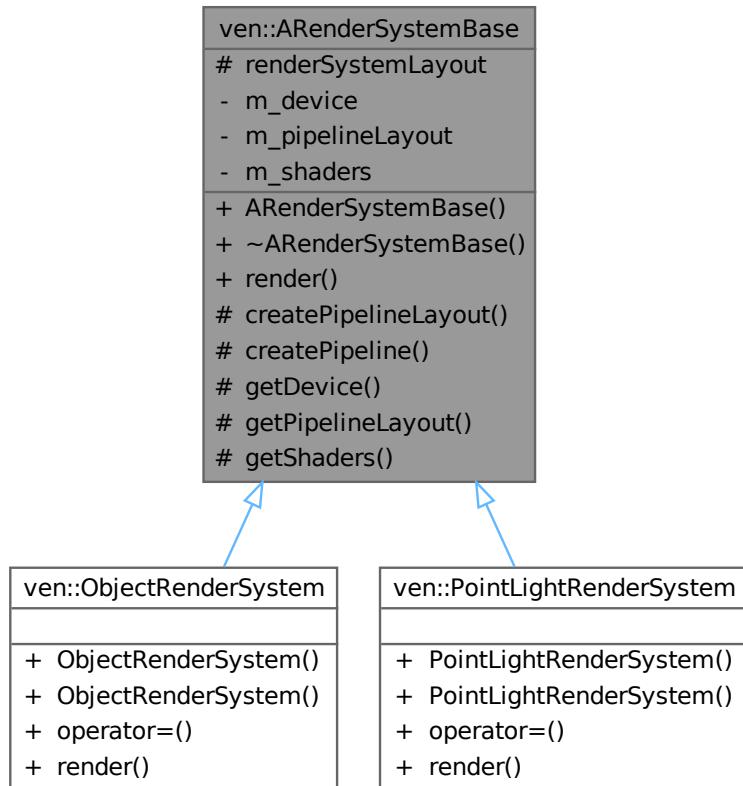
## Class Documentation

### 7.1 ven::ARenderSystemBase Class Reference

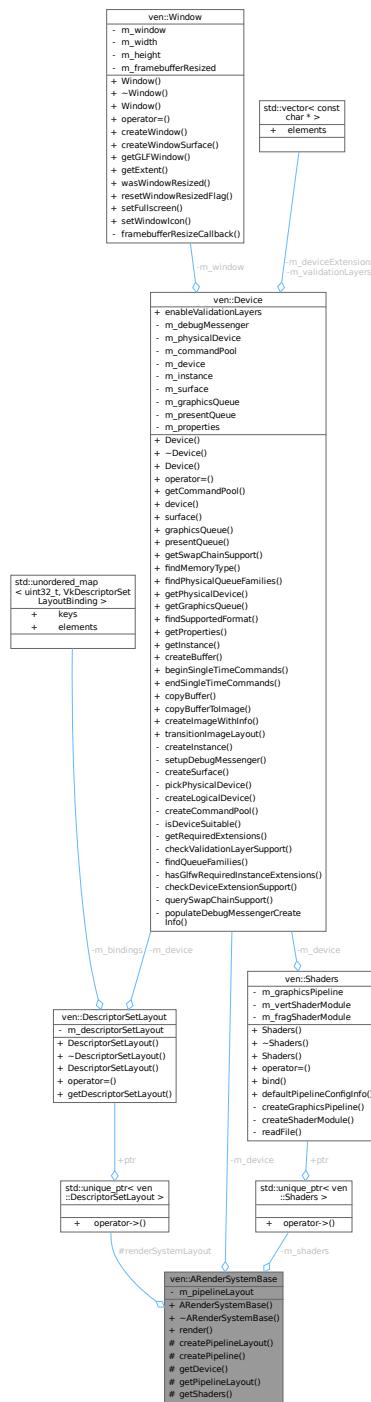
Abstract class for render system base.

```
#include <ABase.hpp>
```

Inheritance diagram for ven::ARenderSystemBase:



## Collaboration diagram for ven::ARenderSystemBase



## Public Member Functions

- ARenderSystemBase (Device &device)
  - virtual ~ARenderSystemBase ()
  - virtual void render (const FrameInfo &frameInfo) const =0

### Protected Member Functions

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalsetLayout, uint32\_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device & getDevice](#) () const
- [VkPipelineLayout getPipelineLayout](#) () const
- const std::unique\_ptr<[Shaders](#)> & [getShaders](#) () const

### Protected Attributes

- std::unique\_ptr<[DescriptorSetLayout](#)> [renderSystemLayout](#)

### Private Attributes

- [Device & m\\_device](#)
- [VkPipelineLayout m\\_pipelineLayout](#) {nullptr}
- std::unique\_ptr<[Shaders](#)> [m\\_shaders](#)

## 7.1.1 Detailed Description

Abstract class for render system base.

Definition at line [20](#) of file [ABase.hpp](#).

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 ARenderSystemBase()

```
ven::ARenderSystemBase::ARenderSystemBase (
    Device & device) [inline], [explicit]
```

Definition at line [24](#) of file [ABase.hpp](#).

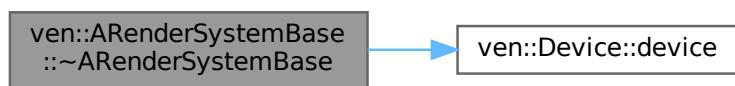
### 7.1.2.2 ~ARenderSystemBase()

```
virtual ven::ARenderSystemBase::~ARenderSystemBase () [inline], [virtual]
```

Definition at line [25](#) of file [ABase.hpp](#).

References [ven::Device::device\(\)](#), [m\\_device](#), and [m\\_pipelineLayout](#).

Here is the call graph for this function:



### 7.1.3 Member Function Documentation

#### 7.1.3.1 createPipeline()

```
void ven::ARenderSystemBase::createPipeline (
    VkRenderPass renderPass,
    const std::string & shadersVertPath,
    const std::string & shadersFragPath,
    bool isLight) [protected]
```

Definition at line 35 of file [base.cpp](#).

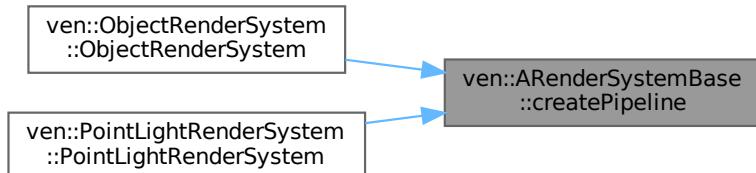
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.1.3.2 createPipelineLayout()

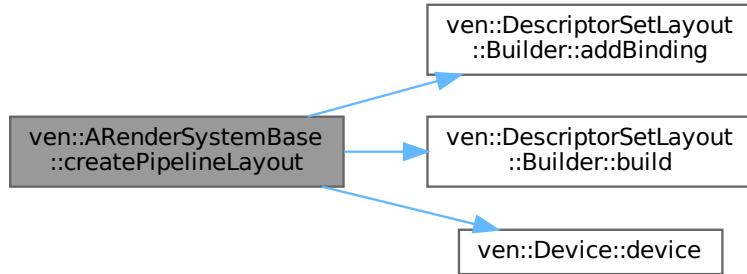
```
void ven::ARenderSystemBase::createPipelineLayout (
    VkDescriptorSetLayout globalsetLayout,
    uint32_t pushConstantSize) [protected]
```

Definition at line 3 of file [base.cpp](#).

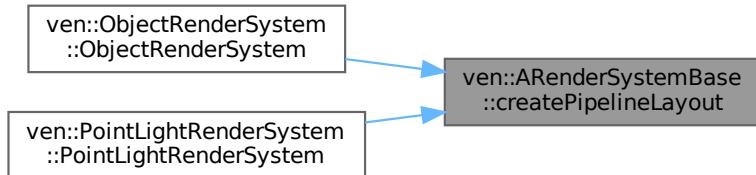
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::Device::device\(\)](#), [m\\_device](#), [m\\_pipelineLayout](#), and [renderSystemLayout](#).

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.1.3.3 getDevice()

`Device & ven::ARenderSystemBase::getDevice () const [inline], [nodiscard], [protected]`

Definition at line 34 of file [ABase.hpp](#).

References [m\\_device](#).

### 7.1.3.4 getPipelineLayout()

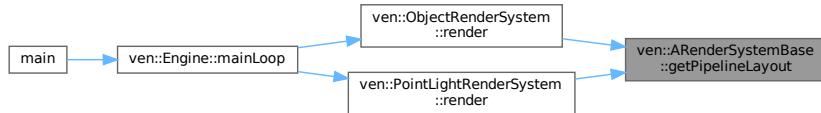
`VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]`

Definition at line 35 of file [ABase.hpp](#).

References [m\\_pipelineLayout](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

Here is the caller graph for this function:



### 7.1.3.5 getShaders()

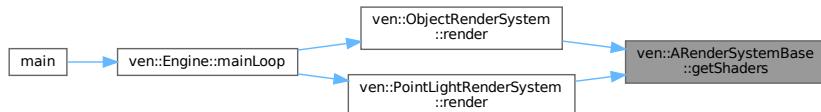
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],  
[nodiscard], [protected]
```

Definition at line 36 of file [ABase.hpp](#).

References [m\\_shaders](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

Here is the caller graph for this function:



### 7.1.3.6 render()

```
virtual void ven::ARenderSystemBase::render (  
    const FrameInfo & frameInfo) const [pure virtual]
```

Implemented in [ven::ObjectRenderSystem](#), and [ven::PointLightRenderSystem](#).

## 7.1.4 Member Data Documentation

### 7.1.4.1 m\_device

```
Device& ven::ARenderSystemBase::m_device [private]
```

Definition at line 42 of file [ABase.hpp](#).

Referenced by [createPipelineLayout\(\)](#), [getDevice\(\)](#), and [~ARenderSystemBase\(\)](#).

### 7.1.4.2 m\_pipelineLayout

```
VkPipelineLayout ven::ARenderSystemBase::m_pipelineLayout {nullptr} [private]
```

Definition at line 43 of file [ABase.hpp](#).

Referenced by [createPipelineLayout\(\)](#), [getPipelineLayout\(\)](#), and [~ARenderSystemBase\(\)](#).

### 7.1.4.3 m\_shaders

```
std::unique_ptr<Shaders> ven::ARenderSystemBase::m_shaders [private]
```

Definition at line 44 of file [ABase.hpp](#).

Referenced by [getShaders\(\)](#).

### 7.1.4.4 renderSystemLayout

```
std::unique_ptr<DescriptorSetLayout> ven::ARenderSystemBase::renderSystemLayout [protected]
```

Definition at line 38 of file [ABase.hpp](#).

Referenced by [createPipelineLayout\(\)](#), and [ven::ObjectRenderSystem::render\(\)](#).

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

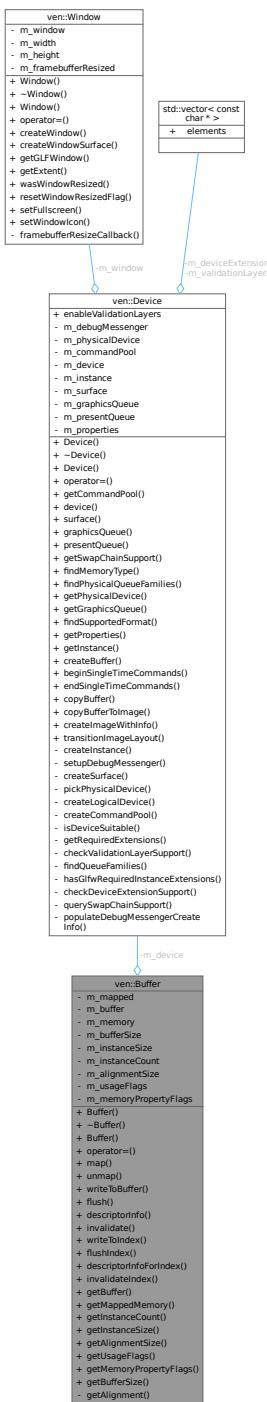
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[ABase.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/[base.cpp](#)

## 7.2 ven::Buffer Class Reference

Class for buffer.

```
#include <Buffer.hpp>
```

Collaboration diagram for ven::Buffer:



## Public Member Functions

- `Buffer (Device &device, VkDeviceSize instanceSize, uint32_t instanceCount, VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize minOffsetAlignment=1)`
- `~Buffer ()`
- `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.*
- `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.*
- `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
 

*Copies "instanceSize" bytes of data to the mapped buffer at an offset of index \* alignmentSize.*
- `VkResult flushIndex` (`const VkDeviceSize index`) const
 

*Flush the memory range at index \* alignmentSize of the buffer to make it visible to the device.*
- `VkDescriptorBufferInfo descriptorInfoForIndex` (`const VkDeviceSize index`) const
 

*Create a buffer info descriptor.*
- `VkResult invalidateIndex` (`const VkDeviceSize index`) const
 

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

### Static Private Member Functions

- `static VkDeviceSize getAlignment` (`const VkDeviceSize instanceSize, const VkDeviceSize minOffsetAlignment`)
 

*Returns the minimum instance size required to be compatible with devices minOffsetAlignment.*

### Private Attributes

- `Device & m_device`
- `void * m_mapped = nullptr`
- `VkBuffer m_buffer = VK_NULL_HANDLE`
- `VkDeviceMemory m_memory = VK_NULL_HANDLE`
- `VkDeviceSize m_bufferSize`
- `VkDeviceSize m_instanceSize`
- `uint32_t m_instanceCount`
- `VkDeviceSize m_alignmentSize`
- `VkBufferUsageFlags m_usageFlags`
- `VkMemoryPropertyFlags m_memoryPropertyFlags`

## 7.2.1 Detailed Description

Class for buffer.

Definition at line 20 of file [Buffer.hpp](#).

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 Buffer() [1/2]

```
ven::Buffer::Buffer (
    Device & device,
    VkDeviceSize instanceSize,
    uint32_t instanceCount,
    VkBufferUsageFlags usageFlags,
    VkMemoryPropertyFlags memoryPropertyFlags,
    VkDeviceSize minOffsetAlignment = 1)
```

Definition at line 5 of file [buffer.cpp](#).

References [ven::Device::createBuffer\(\)](#), [m\\_alignmentSize](#), [m\\_buffer](#), [m\\_bufferSize](#), [m\\_instanceCount](#), [m\\_memory](#), [m\\_memoryPropertyFlags](#), and [m\\_usageFlags](#).

Here is the call graph for this function:



### 7.2.2.2 ~Buffer()

```
ven::Buffer::~Buffer ()
```

Definition at line 11 of file [buffer.cpp](#).

### 7.2.2.3 Buffer() [2/2]

```
ven::Buffer::Buffer (
    const Buffer & ) [delete]
```

## 7.2.3 Member Function Documentation

### 7.2.3.1 descriptorInfo()

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

Create a buffer info descriptor.

**Parameters**

<i>size</i>	(Optional) Size of the memory range of the descriptor
<i>offset</i>	(Optional) Byte offset from beginning

**Returns**

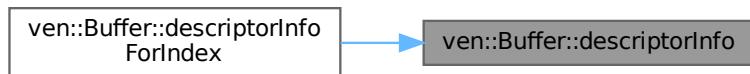
VkDescriptorBufferInfo of specified offset and range

Definition at line 76 of file [Buffer.hpp](#).

References [m\\_buffer](#).

Referenced by [descriptorInfoForIndex\(\)](#).

Here is the caller graph for this function:

**7.2.3.2 descriptorInfoForIndex()**

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

Create a buffer info descriptor.

**Parameters**

<i>index</i>	Specifies the region given by index * alignmentSize
--------------	---

**Returns**

VkDescriptorBufferInfo for instance at index

Definition at line 115 of file [Buffer.hpp](#).

References [descriptorInfo\(\)](#), and [m\\_alignmentSize](#).

Here is the call graph for this function:



### 7.2.3.3 flush()

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

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

#### Note

Only required for non-coherent memory

#### Parameters

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

#### Returns

VkResult of the flush call

Definition at line 45 of file [buffer.cpp](#).

Referenced by [flushIndex\(\)](#).

Here is the caller graph for this function:



### 7.2.3.4 flushIndex()

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

Flush the memory range at  $\text{index} * \text{alignmentSize}$  of the buffer to make it visible to the device.

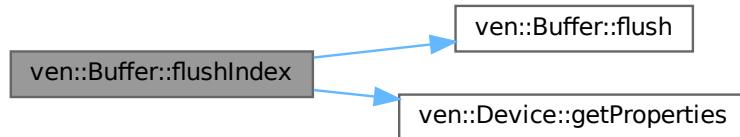
#### Parameters

<i>index</i>	Used in offset calculation
--------------	----------------------------

Definition at line 105 of file [Buffer.hpp](#).

References [flush\(\)](#), [ven::Device::getProperties\(\)](#), [m\\_alignmentSize](#), and [m\\_device](#).

Here is the call graph for this function:



### 7.2.3.5 `getAlignment()`

```
static VkDeviceSize ven::Buffer::getAlignment (
    const VkDeviceSize instanceSize,
    const VkDeviceSize minOffsetAlignment) [inline], [static], [private]
```

Returns the minimum instance size required to be compatible with devices minOffsetAlignment.

#### Parameters

<i>instanceSize</i>	The size of an instance
<i>minOffsetAlignment</i>	The minimum required alignment, in bytes, for the offset member (eg minUniformBufferOffsetAlignment)

#### Returns

VkResult of the buffer mapping call

Definition at line 147 of file [Buffer.hpp](#).

### 7.2.3.6 `getAlignmentSize()`

```
VkDeviceSize ven::Buffer::getAlignmentSize () const [inline], [nodiscard]
```

Definition at line 132 of file [Buffer.hpp](#).

References [m\\_alignmentSize](#).

### 7.2.3.7 `getBuffer()`

```
VkBuffer ven::Buffer::getBuffer () const [inline], [nodiscard]
```

Definition at line 128 of file [Buffer.hpp](#).

References [m\\_buffer](#).

### 7.2.3.8 `getBufferSize()`

```
VkDeviceSize ven::Buffer::getBufferSize () const [inline], [nodiscard]
```

Definition at line 135 of file [Buffer.hpp](#).

References [m\\_bufferSize](#).

### 7.2.3.9 `getInstanceCount()`

```
uint32_t ven::Buffer::getInstanceCount () const [inline], [nodiscard]
```

Definition at line 130 of file [Buffer.hpp](#).

References [m\\_instanceCount](#).

### 7.2.3.10 `getInstanceSize()`

```
VkDeviceSize ven::Buffer::getInstanceSize () const [inline], [nodiscard]
```

Definition at line 131 of file [Buffer.hpp](#).

References [m\\_instanceSize](#).

### 7.2.3.11 `getMappedMemory()`

```
void * ven::Buffer::getMappedMemory () const [inline], [nodiscard]
```

Definition at line 129 of file [Buffer.hpp](#).

References [m\\_mapped](#).

### 7.2.3.12 `getMemoryPropertyFlags()`

```
VkMemoryPropertyFlags ven::Buffer::getMemoryPropertyFlags () const [inline], [nodiscard]
```

Definition at line 134 of file [Buffer.hpp](#).

References [m\\_memoryPropertyFlags](#).

### 7.2.3.13 getUsageFlags()

```
VkBufferUsageFlags ven::Buffer::getUsageFlags () const [inline], [nodiscard]
```

Definition at line 133 of file [Buffer.hpp](#).

References [m\\_usageFlags](#).

### 7.2.3.14 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

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

#### Returns

`VkResult` of the invalidate call

Definition at line 55 of file [buffer.cpp](#).

Referenced by [invalidateIndex\(\)](#).

Here is the caller graph for this function:



### 7.2.3.15 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

<i>index</i>	Specifies the region to invalidate: index * alignmentSize
--------------	---

### Returns

VkResult of the invalidate call

Definition at line 126 of file [Buffer.hpp](#).

References [invalidate\(\)](#), and [m\\_alignmentSize](#).

Here is the call graph for this function:



### 7.2.3.16 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

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

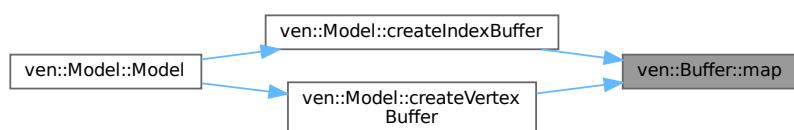
### Returns

VkResult of the buffer mapping call

Definition at line 18 of file [buffer.cpp](#).

Referenced by [ven::Model::createIndexBuffer\(\)](#), and [ven::Model::createVertexBuffer\(\)](#).

Here is the caller graph for this function:



### 7.2.3.17 operator=(**)**

```
Buffer & ven::Buffer::operator= (
    const Buffer & ) [delete]
```

### 7.2.3.18 unmap(**)**

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

Unmap a mapped memory range.

#### Note

Does not return a result as vkUnmapMemory can't fail

Definition at line 24 of file [buffer.cpp](#).

### 7.2.3.19 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

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

Definition at line 32 of file [buffer.cpp](#).

Referenced by [writeToIndex\(\)](#).

Here is the caller graph for this function:



### 7.2.3.20 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

<i>data</i>	Pointer to the data to copy
<i>index</i>	Used in offset calculation

Definition at line 98 of file [Buffer.hpp](#).

References [m\\_alignmentSize](#), [m\\_instanceSize](#), and [writeToBuffer\(\)](#).

Here is the call graph for this function:



## 7.2.4 Member Data Documentation

### 7.2.4.1 m\_alignmentSize

```
VkDeviceSize ven::Buffer::m_alignmentSize [private]
```

Definition at line 157 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), [descriptorInfoForIndex\(\)](#), [flushIndex\(\)](#), [getAlignmentSize\(\)](#), [invalidateIndex\(\)](#), and [writeToIndex\(\)](#).

### 7.2.4.2 m\_buffer

```
VkBuffer ven::Buffer::m_buffer = VK_NULL_HANDLE [private]
```

Definition at line 151 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), [descriptorInfo\(\)](#), and [getBuffer\(\)](#).

### 7.2.4.3 m\_bufferSize

```
VkDeviceSize ven::Buffer::m_bufferSize [private]
```

Definition at line 154 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getBufferSize\(\)](#).

#### 7.2.4.4 m\_device

```
Device& ven::Buffer::m_device [private]
```

Definition at line 149 of file [Buffer.hpp](#).

Referenced by [flushIndex\(\)](#).

#### 7.2.4.5 m\_instanceCount

```
uint32_t ven::Buffer::m_instanceCount [private]
```

Definition at line 156 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getInstanceCount\(\)](#).

#### 7.2.4.6 m\_instanceSize

```
VkDeviceSize ven::Buffer::m_instanceSize [private]
```

Definition at line 155 of file [Buffer.hpp](#).

Referenced by [getInstanceSize\(\)](#), and [writeToIndex\(\)](#).

#### 7.2.4.7 m\_mapped

```
void* ven::Buffer::m_mapped = nullptr [private]
```

Definition at line 150 of file [Buffer.hpp](#).

Referenced by [getMappedMemory\(\)](#).

#### 7.2.4.8 m\_memory

```
VkDeviceMemory ven::Buffer::m_memory = VK_NULL_HANDLE [private]
```

Definition at line 152 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#).

#### 7.2.4.9 m\_memoryPropertyFlags

```
VkMemoryPropertyFlags ven::Buffer::m_memoryPropertyFlags [private]
```

Definition at line 159 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getMemoryPropertyFlags\(\)](#).

#### 7.2.4.10 m\_usageFlags

```
VkBufferUsageFlags ven::Buffer::m_usageFlags [private]
```

Definition at line 158 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getUsageFlags\(\)](#).

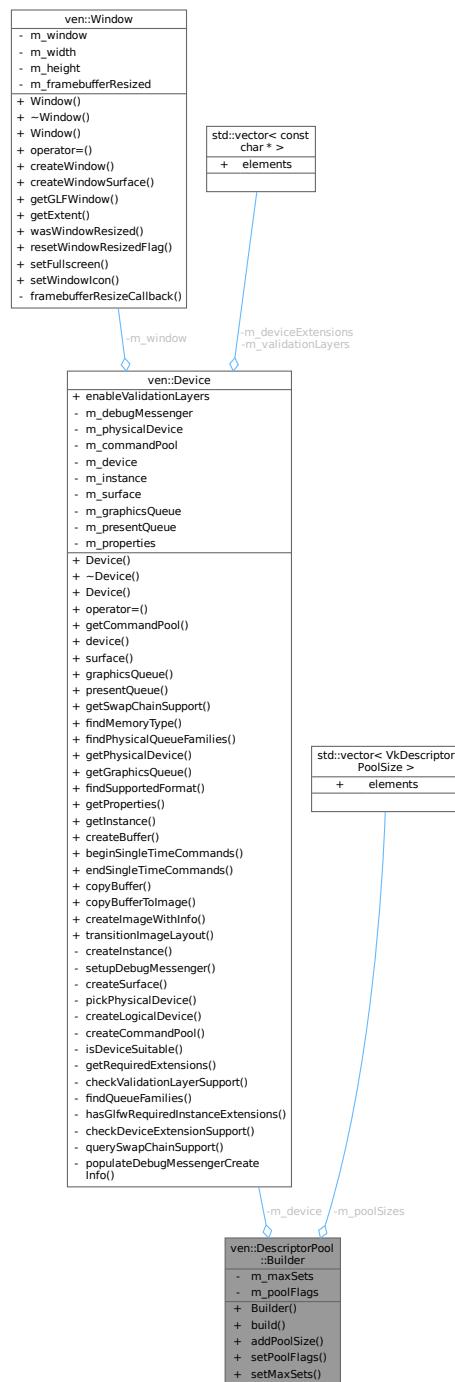
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Buffer.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[buffer.cpp](#)

### 7.3 ven::DescriptorPool::Builder Class Reference

```
#include <Pool.hpp>
```

Collaboration diagram for ven::DescriptorPool::Builder:



## Public Member Functions

- [Builder \(Device &device\)](#)
- [std::unique\\_ptr< DescriptorPool > build \(\) const](#)
- [Builder & addPoolSize \(const VkDescriptorType descriptorType, const uint32\\_t count\)](#)
- [Builder & setPoolFlags \(const VkDescriptorPoolCreateFlags flags\)](#)
- [Builder & setMaxSets \(const uint32\\_t count\)](#)

## Private Attributes

- `Device & m_device`
- `std::vector< VkDescriptorPoolSize > m_poolSizes`
- `uint32_t m_maxSets {DEFAULT_MAX_SETS}`
- `VkDescriptorPoolCreateFlags m_poolFlags {0}`

### 7.3.1 Detailed Description

Definition at line 26 of file [Pool.hpp](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 Builder()

```
ven::DescriptorPool::Builder::Builder (
    Device & device) [inline], [explicit]
```

Definition at line 30 of file [Pool.hpp](#).

### 7.3.3 Member Function Documentation

#### 7.3.3.1 addPoolSize()

```
Builder & ven::DescriptorPool::Builder::addPoolSize (
    const VkDescriptorType descriptorType,
    const uint32_t count) [inline]
```

Definition at line 34 of file [Pool.hpp](#).

References [m\\_poolSizes](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



### 7.3.3.2 build()

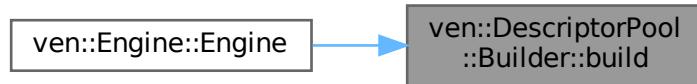
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],  
[nodiscard]
```

Definition at line 32 of file [Pool.hpp](#).

References [m\\_device](#), [m\\_maxSets](#), [m\\_poolFlags](#), and [m\\_poolSizes](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



### 7.3.3.3 setMaxSets()

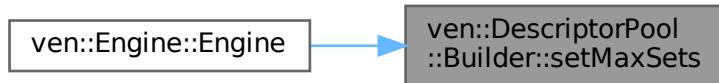
```
Builder & ven::DescriptorPool::Builder::setMaxSets (const uint32_t count) [inline]
```

Definition at line 36 of file [Pool.hpp](#).

References [m\\_maxSets](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



### 7.3.3.4 setPoolFlags()

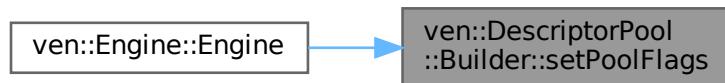
```
Builder & ven::DescriptorPool::Builder::setPoolFlags (
    const VkDescriptorPoolCreateFlags flags) [inline]
```

Definition at line 35 of file [Pool.hpp](#).

References [m\\_poolFlags](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



## 7.3.4 Member Data Documentation

### 7.3.4.1 m\_device

```
Device& ven::DescriptorPool::Builder::m_device [private]
```

Definition at line 40 of file [Pool.hpp](#).

Referenced by [build\(\)](#).

### 7.3.4.2 m\_maxSets

```
uint32_t ven::DescriptorPool::Builder::m_maxSets {DEFAULT_MAX_SETS} [private]
```

Definition at line 42 of file [Pool.hpp](#).

Referenced by [build\(\)](#), and [setMaxSets\(\)](#).

### 7.3.4.3 m\_poolFlags

```
VkDescriptorPoolCreateFlags ven::DescriptorPool::Builder::m_poolFlags {0} [private]
```

Definition at line 43 of file [Pool.hpp](#).

Referenced by [build\(\)](#), and [setPoolFlags\(\)](#).

### 7.3.4.4 m\_poolSizes

```
std::vector<VkDescriptorPoolSize> ven::DescriptorPool::Builder::m_poolSizes [private]
```

Definition at line 41 of file [Pool.hpp](#).

Referenced by [addPoolSize\(\)](#), and [build\(\)](#).

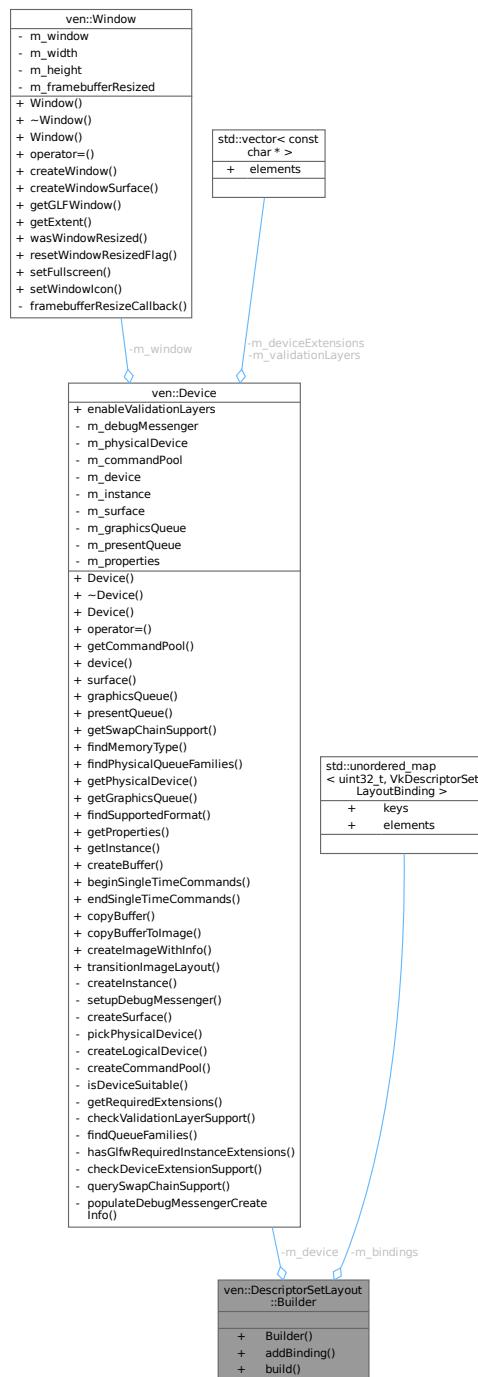
The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[Pool.hpp](#)

## 7.4 ven::DescriptorSetLayout::Builder Class Reference

```
#include <SetLayout.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout::Builder:



## 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`

### Private Attributes

- `Device & m_device`
- `std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > m_bindings`

### 7.4.1 Detailed Description

Definition at line 25 of file [SetLayout.hpp](#).

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 Builder()

```
ven::DescriptorsetLayout::Builder::Builder (
    Device & device) [inline], [explicit]
```

Definition at line 29 of file [SetLayout.hpp](#).

### 7.4.3 Member Function Documentation

#### 7.4.3.1 addBinding()

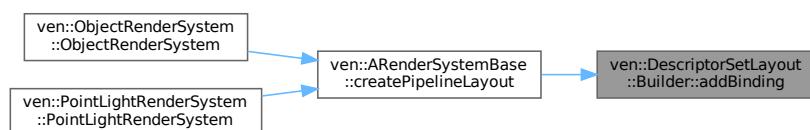
```
ven::DescriptorsetLayout::Builder & ven::DescriptorsetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

Definition at line 5 of file [setLayout.cpp](#).

References [m\\_bindings](#).

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#).

Here is the caller graph for this function:



### 7.4.3.2 build()

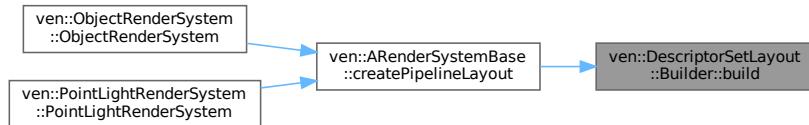
```
std::unique_ptr< DescriptorsetLayout > ven::DescriptorsetLayout::Builder::build () const
[inline]
```

Definition at line 32 of file [SetLayout.hpp](#).

References [m\\_bindings](#), and [m\\_device](#).

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#).

Here is the caller graph for this function:



## 7.4.4 Member Data Documentation

### 7.4.4.1 m\_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorsetLayout::Builder::m_bindings [private]
```

Definition at line 37 of file [SetLayout.hpp](#).

Referenced by [addBinding\(\)](#), and [build\(\)](#).

### 7.4.4.2 m\_device

```
Device& ven::DescriptorsetLayout::Builder::m_device [private]
```

Definition at line 36 of file [SetLayout.hpp](#).

Referenced by [build\(\)](#).

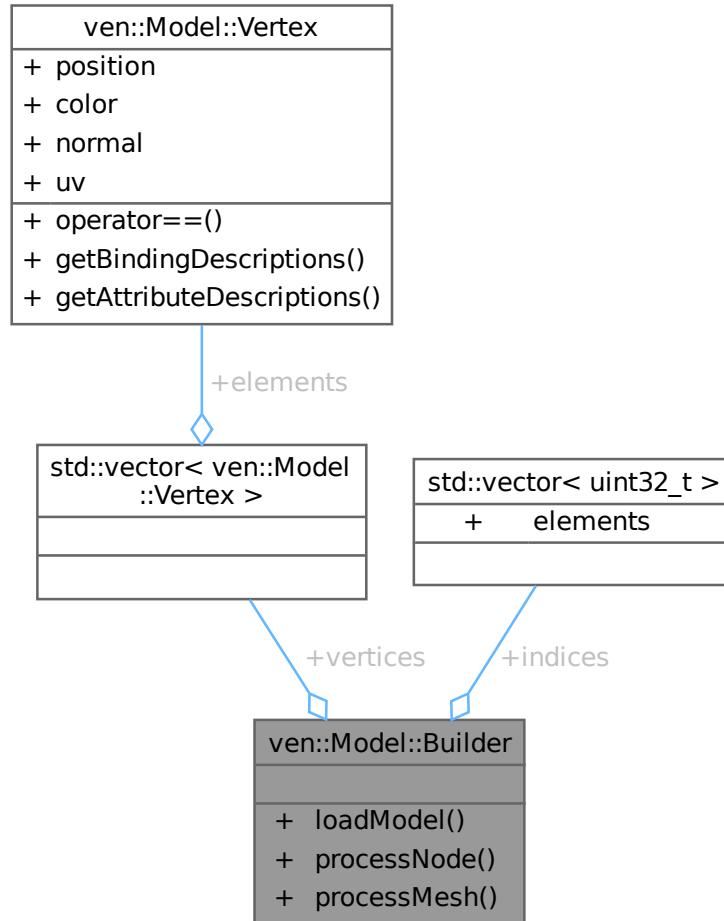
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[SetLayout.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[setLayout.cpp](#)

## 7.5 ven::Model::Builder Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model::Builder:



### Public Member Functions

- void [loadModel](#) (const std::string &filename)
- void [processNode](#) (const aiNode \*node, const aiScene \*scene)
- void [processMesh](#) (const aiMesh \*mesh, const aiScene \*scene)

### Public Attributes

- std::vector<[Vertex](#)> [vertices](#)
- std::vector<uint32\_t> [indices](#)

### 7.5.1 Detailed Description

Definition at line 45 of file [Model.hpp](#).

### 7.5.2 Member Function Documentation

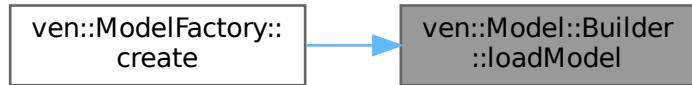
#### 7.5.2.1 `loadModel()`

```
void ven::Model::Builder::loadModel (
    const std::string & filename)
```

Definition at line 105 of file [model.cpp](#).

Referenced by [ven::ModelFactory::create\(\)](#).

Here is the caller graph for this function:



#### 7.5.2.2 `processMesh()`

```
void ven::Model::Builder::processMesh (
    const aiMesh * mesh,
    const aiScene * scene)
```

Definition at line 132 of file [model.cpp](#).

References [ven::Model::Vertex::position](#).

#### 7.5.2.3 `processNode()`

```
void ven::Model::Builder::processNode (
    const aiNode * node,
    const aiScene * scene)
```

Definition at line 121 of file [model.cpp](#).

### 7.5.3 Member Data Documentation

#### 7.5.3.1 indices

```
std::vector<uint32_t> ven::Model::Builder::indices
```

Definition at line 47 of file [Model.hpp](#).

Referenced by [ven::Model::Model\(\)](#).

#### 7.5.3.2 vertices

```
std::vector<Vertex> ven::Model::Builder::vertices
```

Definition at line 46 of file [Model.hpp](#).

Referenced by [ven::Model::Model\(\)](#).

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

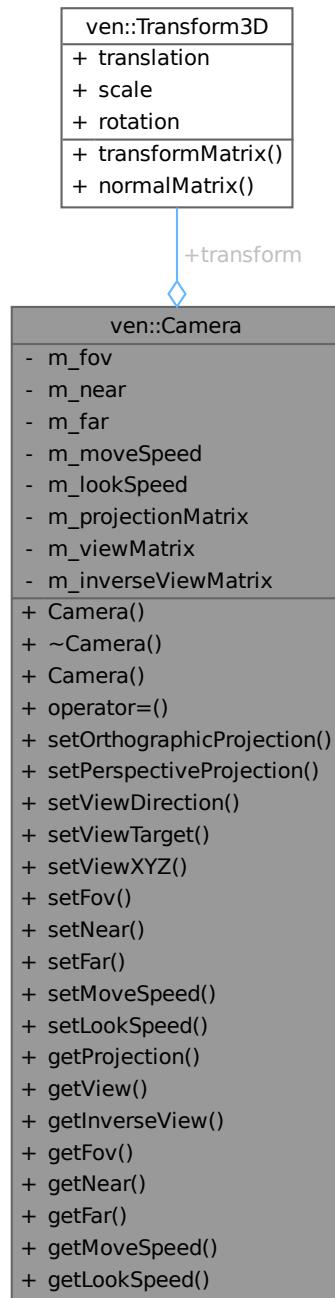
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

## 7.6 ven::Camera Class Reference

Class for camera.

```
#include <Camera.hpp>
```

Collaboration diagram for ven::Camera:



## Public Member Functions

- `Camera` (const float `fov`, const float `near`, const float `far`, const float `moveSpeed`, const float `lookSpeed`)
- `~Camera` ()=default
- `Camera` (const `Camera` &)=delete
- `Camera & operator=` (const `Camera` &)=delete
- void `setOrthographicProjection` (float `left`, float `right`, float `top`, float `bottom`, float `near`, float `far`)

- void [setPerspectiveProjection](#) (float aspect)
- void [setViewDirection](#) (glm::vec3 position, glm::vec3 direction, glm::vec3 up={0.F, -1.F, 0.F})
- void [setViewTarget](#) (const glm::vec3 position, const glm::vec3 target, const glm::vec3 up={0.F, -1.F, 0.F})
- void [setViewXYZ](#) (glm::vec3 position, glm::vec3 rotation)
- void [setFov](#) (const float fov)
- void [setNear](#) (const float near)
- void [setFar](#) (const float far)
- void [setMoveSpeed](#) (const float moveSpeed)
- void [setLookSpeed](#) (const float lookSpeed)
- const glm::mat4 & [getProjection](#) () const
- const glm::mat4 & [getView](#) () const
- const glm::mat4 & [getInverseView](#) () const
- float [getFov](#) () const
- float [getNear](#) () const
- float [getFar](#) () const
- float [getMoveSpeed](#) () const
- float [getLookSpeed](#) () const

### Public Attributes

- Transform3D transform {DEFAULT\_POSITION, {1.F, 1.F, 1.F}, DEFAULT\_ROTATION}

### Private Attributes

- float [m\\_fov](#)
- float [m\\_near](#)
- float [m\\_far](#)
- float [m\\_moveSpeed](#)
- float [m\\_lookSpeed](#)
- glm::mat4 [m\\_projectionMatrix](#) {1.F}
- glm::mat4 [m\\_viewMatrix](#) {1.F}
- glm::mat4 [m\\_inverseViewMatrix](#) {1.F}

## 7.6.1 Detailed Description

Class for camera.

Definition at line 28 of file [Camera.hpp](#).

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 Camera() [1/2]

```
ven::Camera::Camera (
    const float fov,
    const float near,
    const float far,
    const float moveSpeed,
    const float lookSpeed) [inline]
```

Definition at line 32 of file [Camera.hpp](#).

### 7.6.2.2 ~Camera()

```
ven::Camera::~Camera () [default]
```

### 7.6.2.3 Camera() [2/2]

```
ven::Camera::Camera (
    const Camera & ) [delete]
```

## 7.6.3 Member Function Documentation

### 7.6.3.1 getFar()

```
float ven::Camera::getFar () const [inline], [nodiscard]
```

Definition at line 54 of file [Camera.hpp](#).

References [m\\_far](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.2 getFov()

```
float ven::Camera::getFov () const [inline], [nodiscard]
```

Definition at line 52 of file [Camera.hpp](#).

References [m\\_fov](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.3 getInverseView()

```
const glm::mat4 & ven::Camera::getInverseView () const [inline], [nodiscard]
```

Definition at line 51 of file [Camera.hpp](#).

References [m\\_inverseViewMatrix](#).

### 7.6.3.4 getLookSpeed()

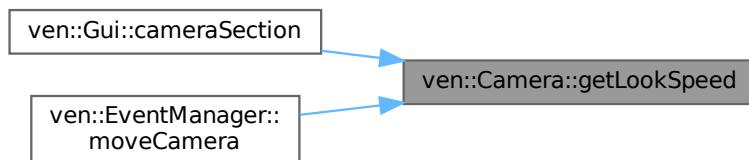
```
float ven::Camera::getLookSpeed () const [inline], [nodiscard]
```

Definition at line 56 of file [Camera.hpp](#).

References [m\\_lookSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



### 7.6.3.5 getMoveSpeed()

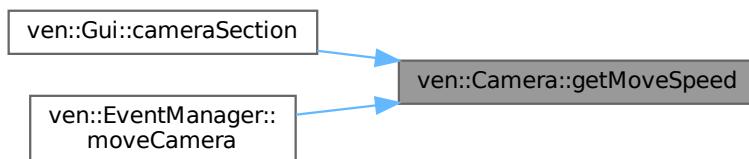
```
float ven::Camera::getMoveSpeed () const [inline], [nodiscard]
```

Definition at line 55 of file [Camera.hpp](#).

References [m\\_moveSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



### 7.6.3.6 `getNear()`

```
float ven::Camera::getNear () const [inline], [nodiscard]
```

Definition at line 53 of file [Camera.hpp](#).

References [m\\_near](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.7 `getProjection()`

```
const glm::mat4 & ven::Camera::getProjection () const [inline], [nodiscard]
```

Definition at line 49 of file [Camera.hpp](#).

References [m\\_projectionMatrix](#).

### 7.6.3.8 `getView()`

```
const glm::mat4 & ven::Camera::getView () const [inline], [nodiscard]
```

Definition at line 50 of file [Camera.hpp](#).

References [m\\_viewMatrix](#).

### 7.6.3.9 `operator=()`

```
Camera & ven::Camera::operator= (
    const Camera &) [delete]
```

### 7.6.3.10 setFar()

```
void ven::Camera::setFar (
    const float far) [inline]
```

Definition at line 45 of file [Camera.hpp](#).

References [m\\_far](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.11 setFov()

```
void ven::Camera::setFov (
    const float fov) [inline]
```

Definition at line 43 of file [Camera.hpp](#).

References [m\\_fov](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.12 setLookSpeed()

```
void ven::Camera::setLookSpeed (
    const float lookSpeed) [inline]
```

Definition at line 47 of file [Camera.hpp](#).

References [m\\_lookSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.13 setMoveSpeed()

```
void ven::Camera::setMoveSpeed (
    const float moveSpeed) [inline]
```

Definition at line 46 of file [Camera.hpp](#).

References [m\\_moveSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.14 setNear()

```
void ven::Camera::setNear (
    const float near) [inline]
```

Definition at line 44 of file [Camera.hpp](#).

References [m\\_near](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



### 7.6.3.15 setOrthographicProjection()

```
void ven::Camera::setOrthographicProjection (
    float left,
    float right,
    float top,
    float bottom,
    float near,
    float far)
```

Definition at line 6 of file [camera.cpp](#).

References [m\\_projectionMatrix](#).

### 7.6.3.16 setPerspectiveProjection()

```
void ven::Camera::setPerspectiveProjection (
    float aspect)
```

Definition at line 17 of file [camera.cpp](#).

### 7.6.3.17 setViewDirection()

```
void ven::Camera::setViewDirection (
    glm::vec3 position,
    glm::vec3 direction,
    glm::vec3 up = {0.F, -1.F, 0.F})
```

Definition at line 29 of file [camera.cpp](#).

### 7.6.3.18 `setViewTarget()`

```
void ven::Camera::setViewTarget (
    const glm::vec3 position,
    const glm::vec3 target,
    const glm::vec3 up = {0.F, -1.F, 0.F}) [inline]
```

Definition at line 41 of file [Camera.hpp](#).

### 7.6.3.19 `setViewXYZ()`

```
void ven::Camera::setViewXYZ (
    glm::vec3 position,
    glm::vec3 rotation)
```

Definition at line 64 of file [camera.cpp](#).

## 7.6.4 Member Data Documentation

### 7.6.4.1 `m_far`

```
float ven::Camera::m_far [private]
```

Definition at line 64 of file [Camera.hpp](#).

Referenced by [getFar\(\)](#), and [setFar\(\)](#).

### 7.6.4.2 `m_fov`

```
float ven::Camera::m_fov [private]
```

Definition at line 62 of file [Camera.hpp](#).

Referenced by [getFov\(\)](#), and [setFov\(\)](#).

### 7.6.4.3 `m_inverseViewMatrix`

```
glm::mat4 ven::Camera::m_inverseViewMatrix {1.F} [private]
```

Definition at line 69 of file [Camera.hpp](#).

Referenced by [getInverseView\(\)](#).

### 7.6.4.4 `m_lookSpeed`

```
float ven::Camera::m_lookSpeed [private]
```

Definition at line 66 of file [Camera.hpp](#).

Referenced by [getLookSpeed\(\)](#), and [setLookSpeed\(\)](#).

#### 7.6.4.5 m\_moveSpeed

```
float ven::Camera::m_moveSpeed [private]
```

Definition at line 65 of file [Camera.hpp](#).

Referenced by [getMoveSpeed\(\)](#), and [setMoveSpeed\(\)](#).

#### 7.6.4.6 m\_near

```
float ven::Camera::m_near [private]
```

Definition at line 63 of file [Camera.hpp](#).

Referenced by [getNear\(\)](#), and [setNear\(\)](#).

#### 7.6.4.7 m\_projectionMatrix

```
glm::mat4 ven::Camera::m_projectionMatrix {1.F} [private]
```

Definition at line 67 of file [Camera.hpp](#).

Referenced by [getProjection\(\)](#), and [setOrthographicProjection\(\)](#).

#### 7.6.4.8 m\_viewMatrix

```
glm::mat4 ven::Camera::m_viewMatrix {1.F} [private]
```

Definition at line 68 of file [Camera.hpp](#).

Referenced by [getView\(\)](#).

#### 7.6.4.9 transform

```
Transform3D ven::Camera::transform {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}
```

Definition at line 58 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

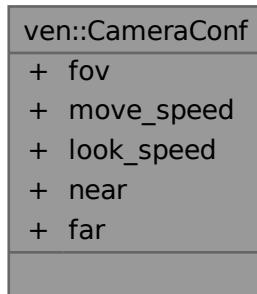
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/[Camera.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Scene/[camera.cpp](#)

## 7.7 ven::CameraConf Struct Reference

```
#include <Config.hpp>
```

Collaboration diagram for ven::CameraConf:



### Public Attributes

- float `fov` = `DEFAULT_FOV`
- float `move_speed` = `DEFAULT_MOVE_SPEED`
- float `look_speed` = `DEFAULT_LOOK_SPEED`
- float `near` = `DEFAULT_NEAR`
- float `far` = `DEFAULT_FAR`

### 7.7.1 Detailed Description

Definition at line 26 of file [Config.hpp](#).

### 7.7.2 Member Data Documentation

#### 7.7.2.1 `far`

```
float ven::CameraConf::far = DEFAULT_FAR
```

Definition at line 32 of file [Config.hpp](#).

#### 7.7.2.2 `fov`

```
float ven::CameraConf::fov = DEFAULT_FOV
```

Definition at line 28 of file [Config.hpp](#).

### 7.7.2.3 look\_speed

```
float ven::CameraConf::look_speed = DEFAULT_LOOK_SPEED
```

Definition at line 30 of file [Config.hpp](#).

### 7.7.2.4 move\_speed

```
float ven::CameraConf::move_speed = DEFAULT_MOVE_SPEED
```

Definition at line 29 of file [Config.hpp](#).

### 7.7.2.5 near

```
float ven::CameraConf::near = DEFAULT_NEAR
```

Definition at line 31 of file [Config.hpp](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Config.hpp

## 7.8 ven::Clock Class Reference

Class for clock.

```
#include <Clock.hpp>
```

Collaboration diagram for ven::Clock:

ven::Clock
- m_startTime
- m_stopTime
- m_deltaTime
- m_isStopped
+ Clock()
+ ~Clock()
+ Clock()
+ operator=(=)
+ start()
+ stop()
+ resume()
+ update()
+ getDeltaTime()
+ getDeltaTimeMS()
+ getFPS()
+ now()

## Public Member Functions

- `Clock ()`
- `~Clock ()=default`
- `Clock (const Clock &)=delete`
- `Clock & operator= (const Clock &)=delete`
- `void start ()`
- `void stop ()`
- `void resume ()`
- `void update ()`
- `float getDeltaTime () const`
- `float getDeltaTimeMS () const`
- `float getFPS () const`

## Static Public Member Functions

- static `std::chrono::high_resolution_clock::time_point now ()`

## Private Attributes

- `TimePoint m_startTime`
- `TimePoint m_stopTime`
- `std::chrono::duration< float > m_deltaTime {0.F}`
- `bool m_isStopped {false}`

### 7.8.1 Detailed Description

Class for clock.

Definition at line 20 of file [Clock.hpp](#).

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 `Clock()` [1/2]

```
ven::Clock::Clock () [inline]
```

Definition at line 24 of file [Clock.hpp](#).

References [start\(\)](#).

Here is the call graph for this function:



### 7.8.2.2 ~Clock()

```
ven::Clock::~Clock () [default]
```

### 7.8.2.3 Clock() [2/2]

```
ven::Clock::Clock (
    const Clock & ) [delete]
```

## 7.8.3 Member Function Documentation

### 7.8.3.1 getDeltaTime()

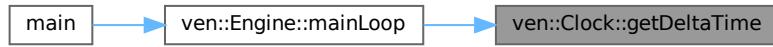
```
float ven::Clock::getDeltaTime () const [inline], [nodiscard]
```

Definition at line 36 of file [Clock.hpp](#).

References [m\\_deltaTime](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



### 7.8.3.2 getDeltaTimeMS()

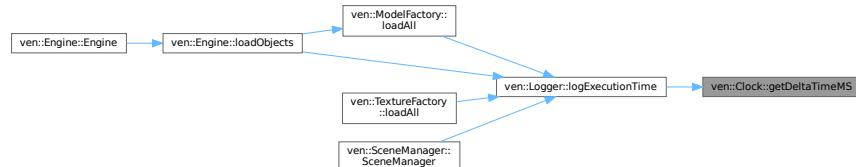
```
float ven::Clock::getDeltaTimeMS () const [inline], [nodiscard]
```

Definition at line 37 of file [Clock.hpp](#).

References [m\\_deltaTime](#).

Referenced by [ven::Logger::logExecutionTime\(\)](#).

Here is the caller graph for this function:



### 7.8.3.3 getFPS()

```
float ven::Clock::getFPS () const [inline], [nodiscard]
```

Definition at line 38 of file [Clock.hpp](#).

References [m\\_deltaTime](#).

### 7.8.3.4 now()

```
static std::chrono::high_resolution_clock::time_point ven::Clock::now () [inline], [static]
```

Definition at line 34 of file [Clock.hpp](#).

### 7.8.3.5 operator=( )

```
Clock & ven::Clock::operator= (
    const Clock & ) [delete]
```

### 7.8.3.6 resume()

```
void ven::Clock::resume ()
```

Definition at line 20 of file [clock.cpp](#).

### 7.8.3.7 start()

```
void ven::Clock::start () [inline]
```

Definition at line 30 of file [Clock.hpp](#).

References [m\\_startTime](#).

Referenced by [Clock\(\)](#).

Here is the caller graph for this function:



### 7.8.3.8 stop()

```
void ven::Clock::stop ()
```

Definition at line 10 of file [clock.cpp](#).

### 7.8.3.9 update()

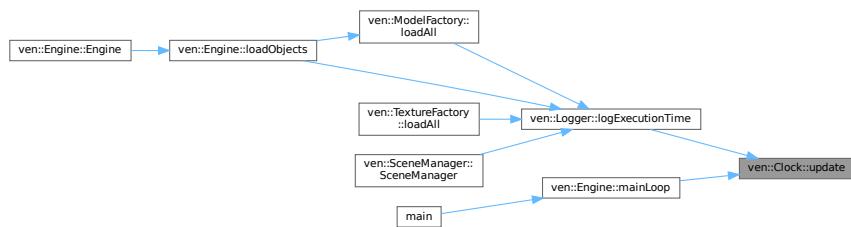
```
void ven::Clock::update ()
```

Definition at line 3 of file [clock.cpp](#).

References [m\\_deltaTime](#), and [m\\_startTime](#).

Referenced by [ven::Logger::logExecutionTime\(\)](#), and [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



## 7.8.4 Member Data Documentation

### 7.8.4.1 m\_deltaTime

```
std::chrono::duration<float> ven::Clock::m_deltaTime {0.F} [private]
```

Definition at line 44 of file [Clock.hpp](#).

Referenced by [getDeltaTime\(\)](#), [getDeltaTimeMS\(\)](#), [getFPS\(\)](#), and [update\(\)](#).

### 7.8.4.2 m\_isStopped

```
bool ven::Clock::m_isStopped {false} [private]
```

Definition at line 46 of file [Clock.hpp](#).

### 7.8.4.3 m\_startTime

```
TimePoint ven::Clock::m_startTime [private]
```

Definition at line 42 of file [Clock.hpp](#).

Referenced by [start\(\)](#), and [update\(\)](#).

#### 7.8.4.4 m\_stopTime

```
TimePoint ven::Clock::m_stopTime [private]
```

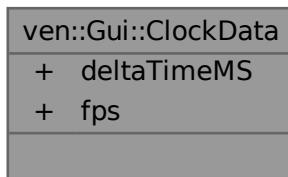
Definition at line 43 of file [Clock.hpp](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp
- /home/runner/work/VEngine/VEngine/src/Utils/[clock.cpp](#)

## 7.9 ven::Gui::ClockData Struct Reference

Collaboration diagram for ven::Gui::ClockData:



### Public Attributes

- float [deltaTimeMS](#) {0.0F}
- float [fps](#) {0.0F}

#### 7.9.1 Detailed Description

Definition at line 32 of file [Gui.hpp](#).

#### 7.9.2 Member Data Documentation

##### 7.9.2.1 deltaTimeMS

```
float ven::Gui::ClockData::deltaTimeMS {0.0F}
```

Definition at line 33 of file [Gui.hpp](#).

Referenced by [ven::Gui::renderFrameWindow\(\)](#).

### 7.9.2.2 fps

```
float ven::Gui::ClockData::fps {0.0F}
```

Definition at line 34 of file [Gui.hpp](#).

Referenced by [ven::Gui::renderFrameWindow\(\)](#).

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

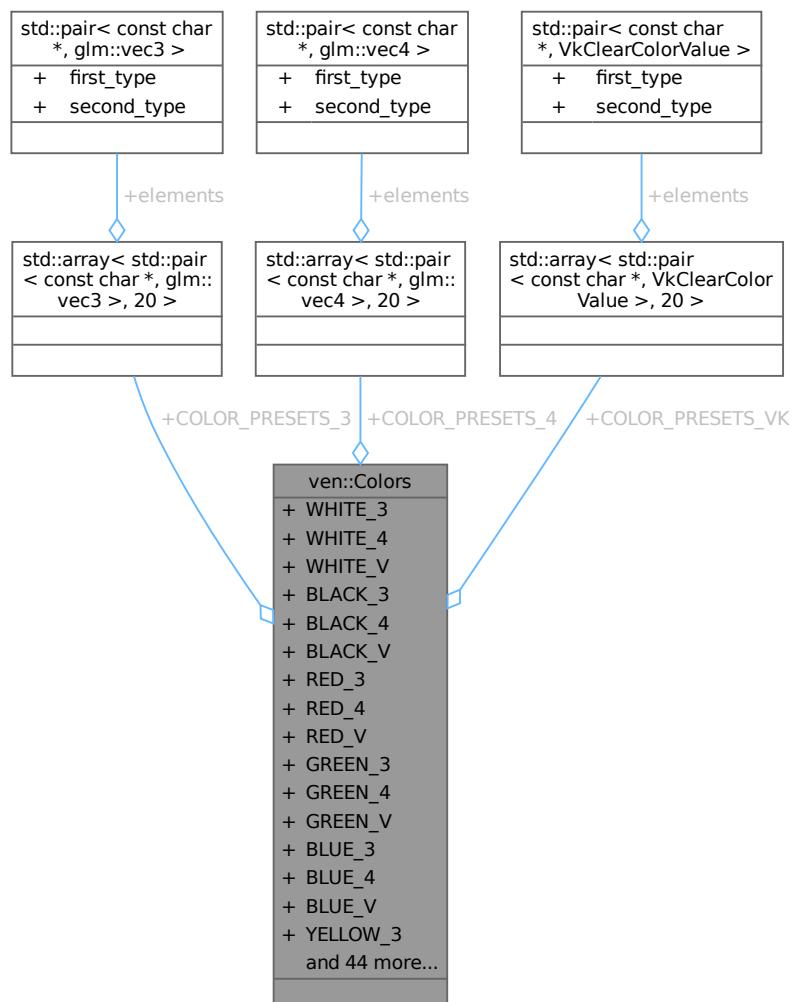
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Gui.hpp](#)

## 7.10 ven::Colors Class Reference

Class for colors.

```
#include <Colors.hpp>
```

Collaboration diagram for ven::Colors:



## Static Public Attributes

- static constexpr glm::vec3 **WHITE\_3** = glm::vec3(COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **WHITE\_4** = { 1.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **WHITE\_V** = { { 1.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **BLACK\_3** = glm::vec3(0.0F)
- static constexpr glm::vec4 **BLACK\_4** = { 0.0F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **BLACK\_V** = { { 0.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **RED\_3** = glm::vec3(COLOR\_MAX, 0.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **RED\_4** = { 1.0F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **RED\_V** = { { 1.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **GREEN\_3** = glm::vec3(0.0F, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **GREEN\_4** = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **GREEN\_V** = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **BLUE\_3** = glm::vec3(0.0F, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **BLUE\_4** = { 0.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **BLUE\_V** = { { 0.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **YELLOW\_3** = glm::vec3(COLOR\_MAX, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **YELLOW\_4** = { 1.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **YELLOW\_V** = { { 1.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **CYAN\_3** = glm::vec3(0.0F, COLOR\_MAX, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **CYAN\_4** = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **CYAN\_V** = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **MAGENTA\_3** = glm::vec3(COLOR\_MAX, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **MAGENTA\_4** = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **MAGENTA\_V** = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **SILVER\_3** = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR\_MAX
- static constexpr glm::vec4 **SILVER\_4** = { 0.75F, 0.75F, 0.75F, 1.0F }
- static constexpr VkClearColorValue **SILVER\_V** = { { 0.75F, 0.75F, 0.75F, 1.0F } }
- static constexpr glm::vec3 **GRAY\_3** = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec4 **GRAY\_4** = { 0.5F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue **GRAY\_V** = { { 0.5F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 **MAROON\_3** = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **MAROON\_4** = { 0.5F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **MAROON\_V** = { { 0.5F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **OLIVE\_3** = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **OLIVE\_4** = { 0.5F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **OLIVE\_V** = { { 0.5F, 0.5F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **LIME\_3** = glm::vec3(0.0F, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec4 **LIME\_4** = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **LIME\_V** = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 **AQUA\_3** = glm::vec3(0.0F, COLOR\_MAX, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **AQUA\_4** = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **AQUA\_V** = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **TEAL\_3** = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec4 **TEAL\_4** = { 0.0F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue **TEAL\_V** = { { 0.0F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 **NAVY\_3** = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec4 **NAVY\_4** = { 0.0F, 0.0F, 0.5F, 1.0F }
- static constexpr VkClearColorValue **NAVY\_V** = { { 0.0F, 0.0F, 0.5F, 1.0F } }
- static constexpr glm::vec3 **FUCHSIA\_3** = glm::vec3(COLOR\_MAX, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec4 **FUCHSIA\_4** = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **FUCHSIA\_V** = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **NIGHT\_BLUE\_3** = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR\_MAX
- static constexpr glm::vec4 **NIGHT\_BLUE\_4** = { 0.098F, 0.098F, 0.439F, 1.0F }

- static constexpr VkClearColorValue **NIGHT\_BLUE\_V** = { { 0.098F, 0.098F, 0.439F, 1.0F } }
- static constexpr glm::vec3 **SKY\_BLUE\_3** = glm::vec3(102.0F, 178.0F, 255.0F) / **COLOR\_MAX**
- static constexpr glm::vec4 **SKY\_BLUE\_4** = { 0.4F, 0.698F, 1.0F, 1.0F }
- static constexpr VkClearColorValue **SKY\_BLUE\_V** = { { 0.4F, 0.698F, 1.0F, 1.0F } }
- static constexpr glm::vec3 **SUNSET\_3** = glm::vec3(255.0F, 128.0F, 0.0F) / **COLOR\_MAX**
- static constexpr glm::vec4 **SUNSET\_4** = { 1.0F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue **SUNSET\_V** = { { 1.0F, 0.5F, 0.0F, 1.0F } }
- static constexpr std::array< std::pair< const char \*, glm::vec3 >, 20 > **COLOR\_PRESETS\_3**
- static constexpr std::array< std::pair< const char \*, glm::vec4 >, 20 > **COLOR\_PRESETS\_4**
- static constexpr std::array< std::pair< const char \*, VkClearColorValue >, 20 > **COLOR\_PRESETS\_VK**

## 7.10.1 Detailed Description

Class for colors.

Definition at line [22](#) of file [Colors.hpp](#).

## 7.10.2 Member Data Documentation

### 7.10.2.1 AQUA\_3

```
glm::vec3 ven::Colors::AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],  
[constexpr]
```

Definition at line [78](#) of file [Colors.hpp](#).

### 7.10.2.2 AQUA\_4

```
glm::vec4 ven::Colors::AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line [79](#) of file [Colors.hpp](#).

### 7.10.2.3 AQUA\_V

```
VkClearColorValue ven::Colors::AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line [80](#) of file [Colors.hpp](#).

### 7.10.2.4 BLACK\_3

```
glm::vec3 ven::Colors::BLACK_3 = glm::vec3(0.0F) [static], [constexpr]
```

Definition at line [30](#) of file [Colors.hpp](#).

### 7.10.2.5 BLACK\_4

```
glm::vec4 ven::Colors::BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 31 of file [Colors.hpp](#).

### 7.10.2.6 BLACK\_V

```
VkClearColorValue ven::Colors::BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 32 of file [Colors.hpp](#).

### 7.10.2.7 BLUE\_3

```
glm::vec3 ven::Colors::BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 42 of file [Colors.hpp](#).

### 7.10.2.8 BLUE\_4

```
glm::vec4 ven::Colors::BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 43 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.9 BLUE\_V

```
VkClearColorValue ven::Colors::BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 44 of file [Colors.hpp](#).

### 7.10.2.10 COLOR\_PRESETS\_3

```
std::array<std::pair<const char *, glm::vec3>, 20> ven::Colors::COLOR_PRESETS_3 [static], [constexpr]
```

#### Initial value:

```
= {{ "White", WHITE_3 },
    {"Black", BLACK_3 },
    {"Red", RED_3 },
    {"Green", GREEN_3 },
    {"Blue", BLUE_3 },
    {"Yellow", YELLOW_3 },
    {"Cyan", CYAN_3 },
    {"Magenta", MAGENTA_3 },
    {"Silver", SILVER_3 },
    {"Gray", GRAY_3 },
    {"Maroon", MAROON_3 },
    {"Olive", OLIVE_3 },
    {"Lime", LIME_3 },
    {"Aqua", AQUA_3 },
    {"Teal", TEAL_3 },
    {"Navy", NAVY_3 },
    {"Fuchsia", FUCHSIA_3 },
    {"Night Blue", NIGHT_BLUE_3 },
    {"Sky Blue", SKY_BLUE_3 },
    {"Sunset", SUNSET_3 }
}
```

Definition at line 107 of file [Colors.hpp](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

### 7.10.2.11 COLOR\_PRESETS\_4

```
std::array<std::pair<const char *, glm::vec4>, 20> ven::Colors::COLOR_PRESETS_4 [static],  
[constexpr]
```

**Initial value:**

```
= {{  
    {"White", WHITE_4},  
    {"Black", BLACK_4},  
    {"Red", RED_4},  
    {"Green", GREEN_4},  
    {"Blue", BLUE_4},  
    {"Yellow", YELLOW_4},  
    {"Cyan", CYAN_4},  
    {"Magenta", MAGENTA_4},  
    {"Silver", SILVER_4},  
    {"Gray", GRAY_4},  
    {"Maroon", MAROON_4},  
    {"Olive", OLIVE_4},  
    {"Lime", LIME_4},  
    {"Aqua", AQUA_4},  
    {"Teal", TEAL_4},  
    {"Navy", NAVY_4},  
    {"Fuchsia", FUCHSIA_4},  
    {"Night Blue", NIGHT_BLUE_4},  
    {"Sky Blue", SKY_BLUE_4},  
    {"Sunset", SUNSET_4}  
}}
```

Definition at line 130 of file [Colors.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

### 7.10.2.12 COLOR\_PRESETS\_VK

```
std::array<std::pair<const char *, VkClearColorValue>, 20> ven::Colors::COLOR_PRESETS_VK  
[static], [constexpr]
```

**Initial value:**

```
= {{  
    {"White", WHITE_V},  
    {"Black", BLACK_V},  
    {"Red", RED_V},  
    {"Green", GREEN_V},  
    {"Blue", BLUE_V},  
    {"Yellow", YELLOW_V},  
    {"Cyan", CYAN_V},  
    {"Magenta", MAGENTA_V},  
    {"Silver", SILVER_V},  
    {"Gray", GRAY_V},  
    {"Maroon", MAROON_V},  
    {"Olive", OLIVE_V},  
    {"Lime", LIME_V},  
    {"Aqua", AQUA_V},  
    {"Teal", TEAL_V},  
    {"Navy", NAVY_V},  
    {"Fuchsia", FUCHSIA_V},  
    {"Night Blue", NIGHT_BLUE_V},  
    {"Sky Blue", SKY_BLUE_V},  
    {"Sunset", SUNSET_V}  
}}
```

Definition at line 153 of file [Colors.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

### 7.10.2.13 CYAN\_3

```
glm::vec3 ven::Colors::CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 50 of file [Colors.hpp](#).

### 7.10.2.14 CYAN\_4

```
glm::vec4 ven::Colors::CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 51 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.15 CYAN\_V

```
VkClearColorValue ven::Colors::CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 52 of file [Colors.hpp](#).

### 7.10.2.16 FUCHSIA\_3

```
glm::vec3 ven::Colors::FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 90 of file [Colors.hpp](#).

### 7.10.2.17 FUCHSIA\_4

```
glm::vec4 ven::Colors::FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 91 of file [Colors.hpp](#).

### 7.10.2.18 FUCHSIA\_V

```
VkClearColorValue ven::Colors::FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 92 of file [Colors.hpp](#).

### 7.10.2.19 GRAY\_3

```
glm::vec3 ven::Colors::GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 62 of file [Colors.hpp](#).

### 7.10.2.20 GRAY\_4

```
glm::vec4 ven::Colors::GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 63 of file [Colors.hpp](#).

Referenced by [ven::Gui::objectsSection\(\)](#).

### 7.10.2.21 GRAY\_V

```
VkClearColorValue ven::Colors::GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 64 of file [Colors.hpp](#).

### 7.10.2.22 GREEN\_3

```
glm::vec3 ven::Colors::GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 38 of file [Colors.hpp](#).

### 7.10.2.23 GREEN\_4

```
glm::vec4 ven::Colors::GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 39 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.24 GREEN\_V

```
VkClearColorValue ven::Colors::GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 40 of file [Colors.hpp](#).

### 7.10.2.25 LIME\_3

```
glm::vec3 ven::Colors::LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 74 of file [Colors.hpp](#).

### 7.10.2.26 LIME\_4

```
glm::vec4 ven::Colors::LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 75 of file [Colors.hpp](#).

### 7.10.2.27 LIME\_V

```
VkClearColorValue ven::Colors::LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 76 of file [Colors.hpp](#).

### 7.10.2.28 MAGENTA\_3

```
glm::vec3 ven::Colors::MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 54 of file [Colors.hpp](#).

### 7.10.2.29 MAGENTA\_4

```
glm::vec4 ven::Colors::MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 55 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.30 MAGENTA\_V

```
VkClearColorValue ven::Colors::MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 56 of file [Colors.hpp](#).

### 7.10.2.31 MAROON\_3

```
glm::vec3 ven::Colors::MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 66 of file [Colors.hpp](#).

### 7.10.2.32 MAROON\_4

```
glm::vec4 ven::Colors::MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 67 of file [Colors.hpp](#).

### 7.10.2.33 MAROON\_V

```
VkClearColorValue ven::Colors::MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 68 of file [Colors.hpp](#).

### 7.10.2.34 NAVY\_3

```
glm::vec3 ven::Colors::NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 86 of file [Colors.hpp](#).

### 7.10.2.35 NAVY\_4

```
glm::vec4 ven::Colors::NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 87 of file [Colors.hpp](#).

### 7.10.2.36 NAVY\_V

```
VkClearColorValue ven::Colors::NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 88 of file [Colors.hpp](#).

### 7.10.2.37 NIGHT\_BLUE\_3

```
glm::vec3 ven::Colors::NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 94 of file [Colors.hpp](#).

### 7.10.2.38 NIGHT\_BLUE\_4

```
glm::vec4 ven::Colors::NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F } [static], [constexpr]
```

Definition at line 95 of file [Colors.hpp](#).

### 7.10.2.39 NIGHT\_BLUE\_V

```
VkClearColorValue ven::Colors::NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } } [static], [constexpr]
```

Definition at line 96 of file [Colors.hpp](#).

### 7.10.2.40 OLIVE\_3

```
glm::vec3 ven::Colors::OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 70 of file [Colors.hpp](#).

### 7.10.2.41 OLIVE\_4

```
glm::vec4 ven::Colors::OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 71 of file [Colors.hpp](#).

### 7.10.2.42 OLIVE\_V

```
VkClearColorValue ven::Colors::OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 72 of file [Colors.hpp](#).

### 7.10.2.43 RED\_3

```
glm::vec3 ven::Colors::RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 34 of file [Colors.hpp](#).

### 7.10.2.44 RED\_4

```
glm::vec4 ven::Colors::RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 35 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.45 RED\_V

```
VkClearColorValue ven::Colors::RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 36 of file [Colors.hpp](#).

### 7.10.2.46 SILVER\_3

```
glm::vec3 ven::Colors::SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 58 of file [Colors.hpp](#).

### 7.10.2.47 SILVER\_4

```
glm::vec4 ven::Colors::SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F } [static], [constexpr]
```

Definition at line 59 of file [Colors.hpp](#).

### 7.10.2.48 SILVER\_V

```
VkClearColorValue ven::Colors::SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } } [static], [constexpr]
```

Definition at line 60 of file [Colors.hpp](#).

### 7.10.2.49 SKY\_BLUE\_3

```
glm::vec3 ven::Colors::SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 98 of file [Colors.hpp](#).

### 7.10.2.50 SKY\_BLUE\_4

```
glm::vec4 ven::Colors::SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 99 of file [Colors.hpp](#).

### 7.10.2.51 SKY\_BLUE\_V

```
VkClearColorValue ven::Colors::SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 100 of file [Colors.hpp](#).

### 7.10.2.52 SUNSET\_3

```
glm::vec3 ven::Colors::SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 102 of file [Colors.hpp](#).

### 7.10.2.53 SUNSET\_4

```
glm::vec4 ven::Colors::SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 103 of file [Colors.hpp](#).

### 7.10.2.54 SUNSET\_V

```
VkClearColorValue ven::Colors::SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 104 of file [Colors.hpp](#).

### 7.10.2.55 TEAL\_3

```
glm::vec3 ven::Colors::TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 82 of file [Colors.hpp](#).

### 7.10.2.56 TEAL\_4

```
glm::vec4 ven::Colors::TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 83 of file [Colors.hpp](#).

### 7.10.2.57 TEAL\_V

```
VkClearColorValue ven::Colors::TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 84 of file [Colors.hpp](#).

### 7.10.2.58 WHITE\_3

```
glm::vec3 ven::Colors::WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 26 of file [Colors.hpp](#).

### 7.10.2.59 WHITE\_4

```
glm::vec4 ven::Colors::WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 27 of file [Colors.hpp](#).

### 7.10.2.60 WHITE\_V

```
VkClearColorValue ven::Colors::WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 28 of file [Colors.hpp](#).

### 7.10.2.61 YELLOW\_3

```
glm::vec3 ven::Colors::YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 46 of file [Colors.hpp](#).

### 7.10.2.62 YELLOW\_4

```
glm::vec4 ven::Colors::YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 47 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

### 7.10.2.63 YELLOW\_V

```
VkClearColorValue ven::Colors::YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 48 of file [Colors.hpp](#).

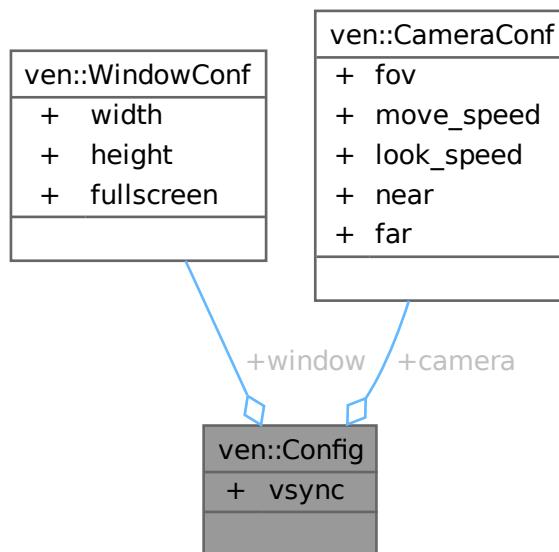
The documentation for this class was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp](#)

## 7.11 ven::Config Struct Reference

```
#include <Config.hpp>
```

Collaboration diagram for ven::Config:



### Public Attributes

- [WindowConf window](#)
- [CameraConf camera](#)
- bool `vsync` = false

### 7.11.1 Detailed Description

Definition at line 35 of file [Config.hpp](#).

## 7.11.2 Member Data Documentation

### 7.11.2.1 camera

```
CameraConf ven::Config::camera
```

Definition at line 38 of file [Config.hpp](#).

### 7.11.2.2 vsync

```
bool ven::Config::vsync = false
```

Definition at line 39 of file [Config.hpp](#).

### 7.11.2.3 window

```
WindowConf ven::Config::window
```

Definition at line 37 of file [Config.hpp](#).

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

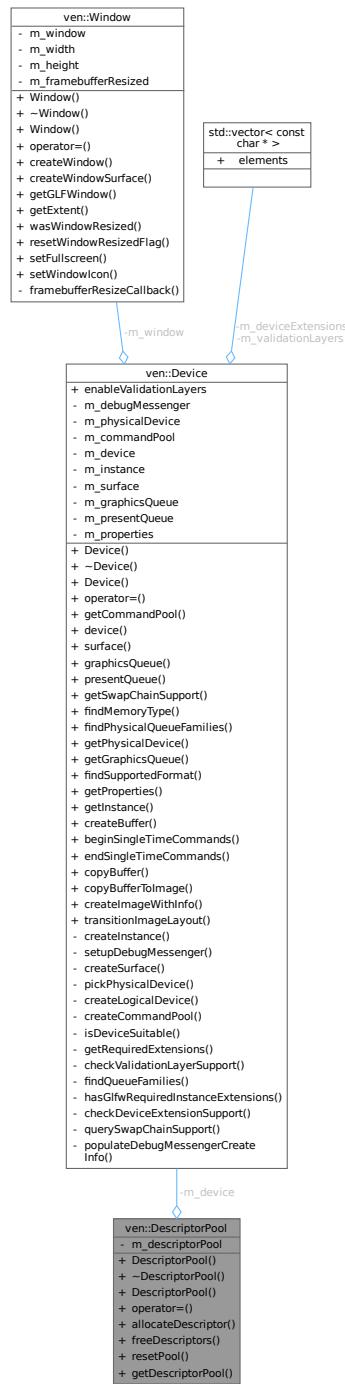
- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/[Config.hpp](#)

## 7.12 ven::DescriptorPool Class Reference

Class for descriptor pool.

```
#include <Pool.hpp>
```

Collaboration diagram for ven::DescriptorPool:



## Classes

- class [Builder](#)

## Public Member Functions

- [`DescriptorPool \(Device &device, uint32\_t maxSets, VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes\)`](#)

- `~DescriptorPool ()`
- `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`
- `VkDescriptorPool getDescriptorPool () const`

### Private Attributes

- `Device & m_device`
- `VkDescriptorPool m_descriptorPool`

### Friends

- class `DescriptorWriter`

## 7.12.1 Detailed Description

Class for descriptor pool.

Definition at line 22 of file `Pool.hpp`.

## 7.12.2 Constructor & Destructor Documentation

### 7.12.2.1 DescriptorPool() [1/2]

```
ven::DescriptorPool::DescriptorPool (
    Device & device,
    uint32_t maxSets,
    VkDescriptorPoolCreateFlags poolFlags,
    const std::vector< VkDescriptorPoolSize > & poolSizes)
```

Definition at line 5 of file `pool.cpp`.

References `ven::Device::device()`, `m_descriptorPool`, and `m_device`.

Here is the call graph for this function:



### 7.12.2.2 ~DescriptorPool()

```
ven::DescriptorPool::~DescriptorPool () [inline]
```

Definition at line 48 of file [Pool.hpp](#).

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



### 7.12.2.3 DescriptorPool() [2/2]

```
ven::DescriptorPool::DescriptorPool (
    const DescriptorPool & ) [delete]
```

## 7.12.3 Member Function Documentation

### 7.12.3.1 allocateDescriptor()

```
bool ven::DescriptorPool::allocateDescriptor (
    VkDescriptorSetLayout descriptorsetLayout,
    VkDescriptorSet & descriptor) const
```

Definition at line 20 of file [pool.cpp](#).

### 7.12.3.2 freeDescriptors()

```
void ven::DescriptorPool::freeDescriptors (
    const std::vector< VkDescriptorSet > & descriptors) const [inline]
```

Definition at line 54 of file [Pool.hpp](#).

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



### 7.12.3.3 getDescriptorPool()

```
VkDescriptorPool ven::DescriptorPool::getDescriptorPool () const [inline], [nodiscard]
```

Definition at line 57 of file [Pool.hpp](#).

References [m\\_descriptorPool](#).

### 7.12.3.4 operator=(())

```
DescriptorPool & ven::DescriptorPool::operator= (
    const DescriptorPool & ) [delete]
```

### 7.12.3.5 resetPool()

```
void ven::DescriptorPool::resetPool () const [inline]
```

Definition at line 55 of file [Pool.hpp](#).

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



## 7.12.4 Friends And Related Symbol Documentation

### 7.12.4.1 DescriptorWriter

```
friend class DescriptorWriter [friend]
```

Definition at line 63 of file [Pool.hpp](#).

## 7.12.5 Member Data Documentation

### 7.12.5.1 m\_descriptorPool

```
VkDescriptorPool ven::DescriptorPool::m_descriptorPool [private]
```

Definition at line 62 of file [Pool.hpp](#).

Referenced by [DescriptorPool\(\)](#), [freeDescriptors\(\)](#), [getDescriptorPool\(\)](#), [resetPool\(\)](#), and [~DescriptorPool\(\)](#).

### 7.12.5.2 m\_device

```
Device& ven::DescriptorPool::m_device [private]
```

Definition at line 61 of file [Pool.hpp](#).

Referenced by [DescriptorPool\(\)](#), [freeDescriptors\(\)](#), [resetPool\(\)](#), and [~DescriptorPool\(\)](#).

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

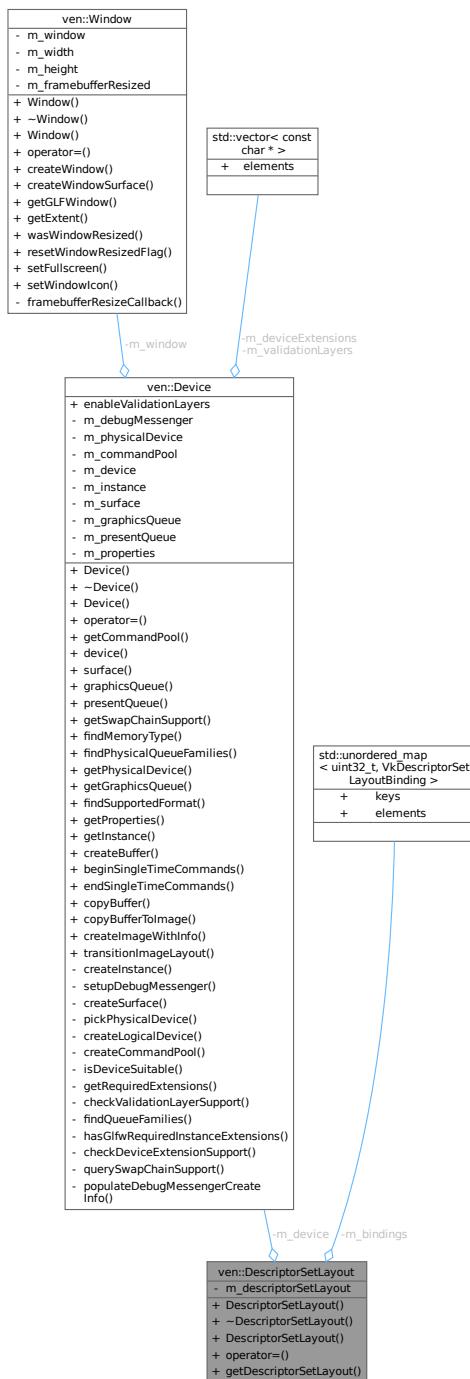
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[Pool.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[pool.cpp](#)

## 7.13 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

```
#include <SetLayout.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout:



## Classes

- class [Builder](#)

## Public Member Functions

- [`DescriptorsetLayout` \(`Device &device, const std::unordered\_map< uint32\_t, VkDescriptorSetLayoutBinding > &bindings\)`](#)

- `~DescriptorSetLayout ()`
- `DescriptorSetLayout (const DescriptorSetLayout &)=delete`
- `DescriptorSetLayout & operator= (const DescriptorSetLayout &)=delete`
- `VkDescriptorSetLayout getDescriptorSetLayout () const`

### Private Attributes

- `Device & m_device`
- `VkDescriptorSetLayout m_descriptorSetLayout`
- `std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > m_bindings`

### Friends

- class `DescriptorWriter`

## 7.13.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file `SetLayout.hpp`.

## 7.13.2 Constructor & Destructor Documentation

### 7.13.2.1 DescriptorsetLayout() [1/2]

```
ven::DescriptorsetLayout::DescriptorsetLayout (
    Device & device,
    const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > & bindings)
```

Definition at line 17 of file `setLayout.cpp`.

References `ven::Device::device()`, `m_descriptorsetLayout`, and `m_device`.

Here is the call graph for this function:



### 7.13.2.2 ~DescriptorSetLayout()

```
ven::DescriptorsetLayout::~DescriptorsetLayout () [inline]
```

Definition at line 42 of file [SetLayout.hpp](#).

References [ven::Device::device\(\)](#), [m\\_descriptorsetLayout](#), and [m\\_device](#).

Here is the call graph for this function:



### 7.13.2.3 DescriptorsetLayout() [2/2]

```
ven::DescriptorsetLayout::DescriptorsetLayout (
    const DescriptorsetLayout & ) [delete]
```

## 7.13.3 Member Function Documentation

### 7.13.3.1 getDescriptorsetLayout()

```
VkDescriptorsetLayout ven::DescriptorsetLayout::getDescriptorsetLayout () const [inline]
```

Definition at line 47 of file [SetLayout.hpp](#).

References [m\\_descriptorsetLayout](#).

### 7.13.3.2 operator=()

```
DescriptorsetLayout & ven::DescriptorsetLayout::operator= (
    const DescriptorsetLayout & ) [delete]
```

## 7.13.4 Friends And Related Symbol Documentation

### 7.13.4.1 DescriptorWriter

```
friend class DescriptorWriter [friend]
```

Definition at line 55 of file [SetLayout.hpp](#).

## 7.13.5 Member Data Documentation

### 7.13.5.1 m\_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorsetLayout::m_bindings [private]
```

Definition at line 53 of file [SetLayout.hpp](#).

Referenced by [ven::DescriptorWriter::writeBuffer\(\)](#).

### 7.13.5.2 m\_descriptorsetLayout

```
VkDescriptorSetLayout ven::DescriptorsetLayout::m_descriptorsetLayout [private]
```

Definition at line 52 of file [SetLayout.hpp](#).

Referenced by [DescriptorsetLayout\(\)](#), [getDescriptorsetLayout\(\)](#), and [~DescriptorsetLayout\(\)](#).

### 7.13.5.3 m\_device

```
Device& ven::DescriptorsetLayout::m_device [private]
```

Definition at line 51 of file [SetLayout.hpp](#).

Referenced by [DescriptorsetLayout\(\)](#), and [~DescriptorsetLayout\(\)](#).

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

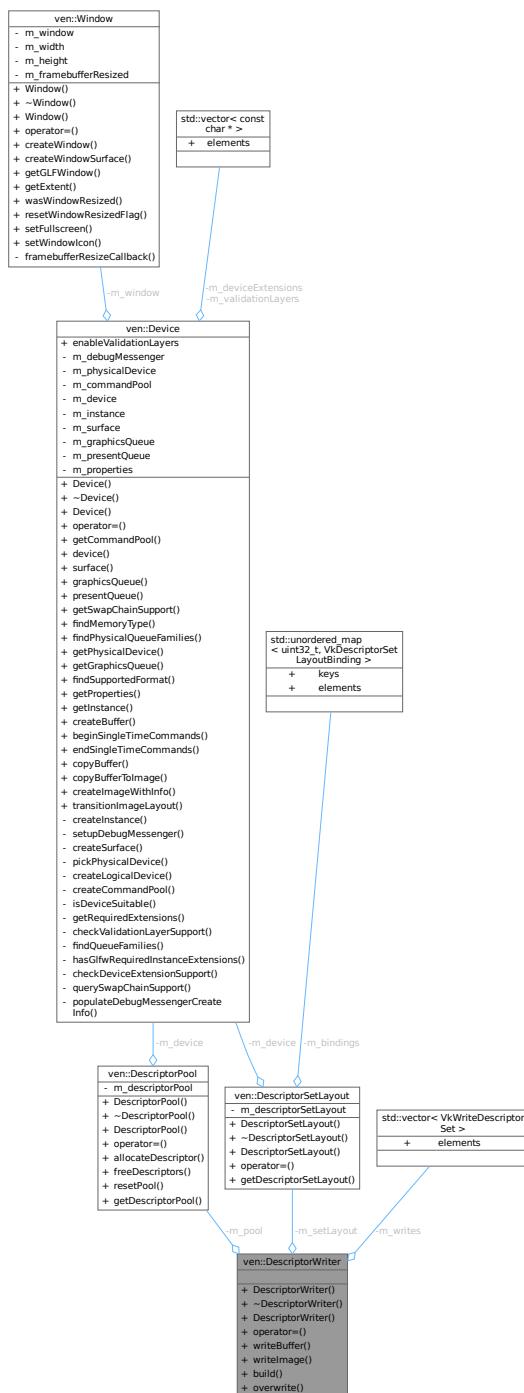
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[SetLayout.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[setLayout.cpp](#)

## 7.14 ven::DescriptorWriter Class Reference

Class for descriptor writer.

```
#include <Writer.hpp>
```

Collaboration diagram for ven::DescriptorWriter:



## Public Member Functions

- `DescriptorWriter (DescriptorsetLayout &setLayout, DescriptorPool &pool)`
- `~DescriptorWriter ()=default`
- `DescriptorWriter (const DescriptorWriter &)=delete`
- `DescriptorWriter & operator= (const DescriptorWriter &)=delete`
- `DescriptorWriter & writeBuffer (uint32_t binding, const VkDescriptorBufferInfo *bufferInfo)`

- `DescriptorWriter & writelImage (uint32_t binding, const VkDescriptorImageInfo *imageInfo)`
- `bool build (VkDescriptorSet &set)`
- `void overwrite (const VkDescriptorSet &set)`

### Private Attributes

- `DescriptorsetLayout & m_setLayout`
- `DescriptorPool & m_pool`
- `std::vector< VkWriteDescriptorSet > m_writes`

## 7.14.1 Detailed Description

Class for descriptor writer.

Definition at line 19 of file [Wwriter.hpp](#).

## 7.14.2 Constructor & Destructor Documentation

### 7.14.2.1 DescriptorWriter() [1/2]

```
ven::DescriptorWriter::DescriptorWriter (
    DescriptorsetLayout & setLayout,
    DescriptorPool & pool) [inline]
```

Definition at line 23 of file [Wwriter.hpp](#).

### 7.14.2.2 ~DescriptorWriter()

```
ven::DescriptorWriter::~DescriptorWriter () [default]
```

### 7.14.2.3 DescriptorWriter() [2/2]

```
ven::DescriptorWriter::DescriptorWriter (
    const DescriptorWriter &) [delete]
```

## 7.14.3 Member Function Documentation

### 7.14.3.1 build()

```
bool ven::DescriptorWriter::build (
    VkDescriptorSet & set)
```

Definition at line 43 of file [writer.cpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#), and [ven::ObjectRenderSystem::render\(\)](#).

Here is the caller graph for this function:



### 7.14.3.2 operator=( )

```
DescriptorWriter & ven::DescriptorWriter::operator= (
    const DescriptorWriter & ) [delete]
```

### 7.14.3.3 overwrite()

```
void ven::DescriptorWriter::overwrite (
    const VkDescriptorSet & set)
```

Definition at line 52 of file [writer.cpp](#).

### 7.14.3.4 writeBuffer()

```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
    uint32_t binding,
    const VkDescriptorBufferInfo * bufferInfo)
```

Definition at line 5 of file [writer.cpp](#).

References [ven::DescriptorsetLayout::m\\_bindings](#), [m\\_setLayout](#), and [m\\_writes](#).

Referenced by [ven::Engine::mainLoop\(\)](#), and [ven::ObjectRenderSystem::render\(\)](#).

Here is the caller graph for this function:



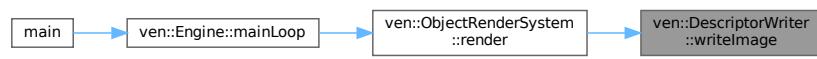
### 7.14.3.5 writeImage()

```
ven::DescriptorWriter & ven::DescriptorWriter::writeImage (
    uint32_t binding,
    const VkDescriptorImageInfo * imageInfo)
```

Definition at line 24 of file [writer.cpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#).

Here is the caller graph for this function:



## 7.14.4 Member Data Documentation

### 7.14.4.1 m\_pool

```
DescriptorPool& ven::DescriptorWriter::m_pool [private]
```

Definition at line 38 of file [Wwriter.hpp](#).

### 7.14.4.2 m\_setLayout

```
DescriptorSetLayout& ven::DescriptorWriter::m_setLayout [private]
```

Definition at line 37 of file [Wwriter.hpp](#).

Referenced by [writeBuffer\(\)](#).

### 7.14.4.3 m\_writes

```
std::vector<VkWriteDescriptorSet> ven::DescriptorWriter::m_writes [private]
```

Definition at line 39 of file [Wwriter.hpp](#).

Referenced by [writeBuffer\(\)](#).

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

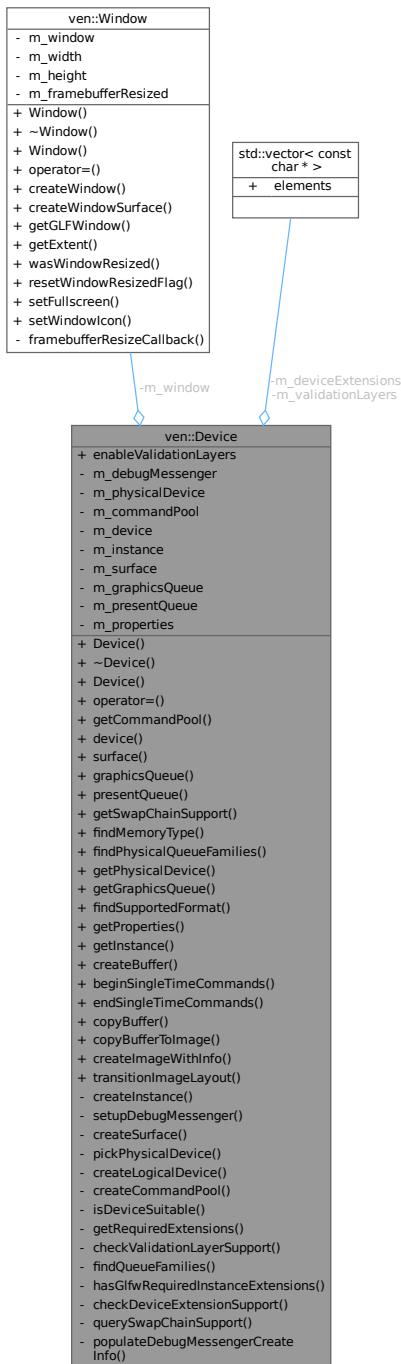
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[Wwriter.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[writer.cpp](#)

## 7.15 ven::Device Class Reference

Class for device.

```
#include <Device.hpp>
```

Collaboration diagram for ven::Device:



## Public Member Functions

- `Device (Window &window)`
- `~Device ()`
- `Device (const Device &) = delete`
- `Device & operator= (const 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 properties) const`
- `QueueFamilyIndices findPhysicalQueueFamilies () const`
- `VkPhysicalDevice getPhysicalDevice () const`
- `VkQueue getGraphicsQueue () const`
- `VkFormat findSupportedFormat (const std::vector< VkFormat > &candidates, VkImageTiling tiling, VkFormatFeatureFlags features) const`
- `VkPhysicalDeviceProperties getProperties () const`
- `VkInstance getInstance () const`
- `void createBuffer (VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags properties, 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 layerCount) const`
- `void createImageWithInfo (const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags properties, VkImage &image, VkDeviceMemory &imageMemory) const`
- `void transitionImageLayout (VkImage image, VkFormat format, VkImageLayout oldLayout, VkImageLayout newLayout, uint32_t mipLevels=1, uint32_t layerCount=1) const`

## Public Attributes

- `const bool enableValidationLayers = true`

## Private Member Functions

- `void createInstance ()`
- `void setupDebugMessenger ()`
- `void createSurface ()`
- `void pickPhysicalDevice ()`
- `void createLogicalDevice ()`
- `void createCommandPool ()`
- `bool isDeviceSuitable (VkPhysicalDevice device) const`
- `std::vector< const char * > getRequiredExtensions () const`
- `bool checkValidationLayerSupport () const`
- `QueueFamilyIndices findQueueFamilies (VkPhysicalDevice device) const`
- `void hasGlfwRequiredInstanceExtensions () const`
- `bool checkDeviceExtensionSupport (VkPhysicalDevice device) const`
- `SwapChainSupportDetails querySwapChainSupport (VkPhysicalDevice device) const`

## Static Private Member Functions

- `static void populateDebugMessengerCreateInfo (VkDebugUtilsMessengerCreateInfoEXT &createInfo)`

## Private Attributes

- `Window & m_window`
- `VkDebugUtilsMessengerEXT m_debugMessenger`
- `VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE`
- `VkCommandPool m_commandPool`
- `VkDevice m_device`
- `VkInstance m_instance`
- `VkSurfaceKHR m_surface`
- `VkQueue m_graphicsQueue`
- `VkQueue m_presentQueue`
- `VkPhysicalDeviceProperties m_properties`
- `const std::vector< const char * > m_validationLayers = {"VK_LAYER_KHRONOS_validation"}`
- `const std::vector< const char * > m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME}`

## 7.15.1 Detailed Description

Class for device.

Definition at line 35 of file `Device.hpp`.

## 7.15.2 Constructor & Destructor Documentation

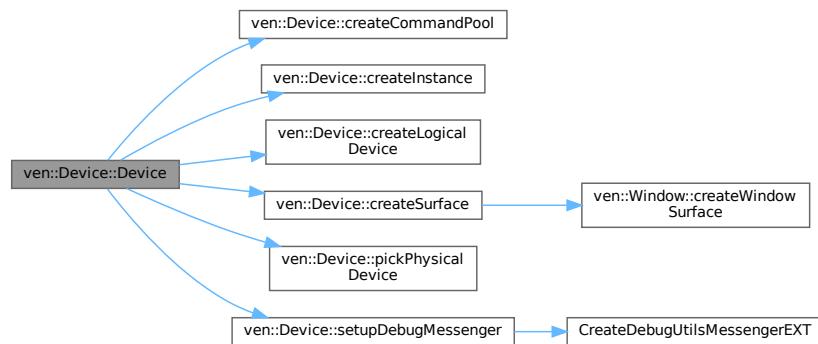
### 7.15.2.1 Device() [1/2]

```
ven::Device::Device (
    Window & window) [explicit]
```

Definition at line 32 of file `device.cpp`.

References `createCommandPool()`, `createInstance()`, `createLogicalDevice()`, `createSurface()`, `pickPhysicalDevice()`, and `setupDebugMessenger()`.

Here is the call graph for this function:



### 7.15.2.2 ~Device()

```
ven::Device::~Device ()
```

Definition at line 42 of file [device.cpp](#).

References [DestroyDebugUtilsMessengerEXT\(\)](#).

Here is the call graph for this function:



### 7.15.2.3 Device() [2/2]

```
ven::Device::Device (
    const Device & ) [delete]
```

## 7.15.3 Member Function Documentation

### 7.15.3.1 beginSingleTimeCommands()

```
VkCommandBuffer ven::Device::beginSingleTimeCommands () const [nodiscard]
```

Definition at line 413 of file [device.cpp](#).

### 7.15.3.2 checkDeviceExtensionSupport()

```
bool ven::Device::checkDeviceExtensionSupport (
    VkPhysicalDevice device) const [private]
```

Definition at line 290 of file [device.cpp](#).

### 7.15.3.3 checkValidationLayerSupport()

```
bool ven::Device::checkValidationLayerSupport () const [nodiscard], [private]
```

Definition at line 227 of file [device.cpp](#).

#### 7.15.3.4 copyBuffer()

```
void ven::Device::copyBuffer (
    VkBuffer srcBuffer,
    VkBuffer dstBuffer,
    VkDeviceSize size) const
```

Definition at line [447](#) of file [device.cpp](#).

#### 7.15.3.5 copyBufferToImage()

```
void ven::Device::copyBufferToImage (
    VkBuffer buffer,
    VkImage image,
    uint32_t width,
    uint32_t height,
    uint32_t layerCount) const
```

Definition at line [460](#) of file [device.cpp](#).

#### 7.15.3.6 createBuffer()

```
void ven::Device::createBuffer (
    VkDeviceSize size,
    VkBufferUsageFlags usage,
    VkMemoryPropertyFlags properties,
    VkBuffer & buffer,
    VkDeviceMemory & bufferMemory) const
```

Definition at line [384](#) of file [device.cpp](#).

Referenced by [ven::Buffer::Buffer\(\)](#).

Here is the caller graph for this function:



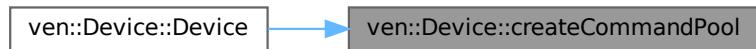
### 7.15.3.7 createCommandPool()

```
void ven::Device::createCommandPool () [private]
```

Definition at line 171 of file [device.cpp](#).

Referenced by [Device\(\)](#).

Here is the caller graph for this function:



### 7.15.3.8 createImageWithInfo()

```
void ven::Device::createImageWithInfo (
    const VkImageCreateInfo & imageInfo,
    VkMemoryPropertyFlags properties,
    VkImage & image,
    VkDeviceMemory & imageMemory) const
```

Definition at line 481 of file [device.cpp](#).

Referenced by [ven::Texture::Texture\(\)](#).

Here is the caller graph for this function:



### 7.15.3.9 `createInstance()`

```
void ven::Device::createInstance () [private]
```

Definition at line 55 of file [device.cpp](#).

Referenced by [Device\(\)](#).

Here is the caller graph for this function:



### 7.15.3.10 `createLogicalDevice()`

```
void ven::Device::createLogicalDevice () [private]
```

Definition at line 124 of file [device.cpp](#).

Referenced by [Device\(\)](#).

Here is the caller graph for this function:



### 7.15.3.11 `createSurface()`

```
void ven::Device::createSurface () [inline], [private]
```

Definition at line 80 of file [Device.hpp](#).

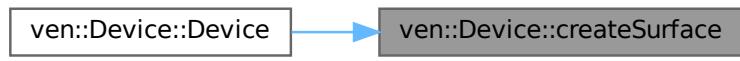
References [ven::Window::createWindowSurface\(\)](#), [m\\_instance](#), [m\\_surface](#), and [m\\_window](#).

Referenced by [Device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.15.3.12 device()

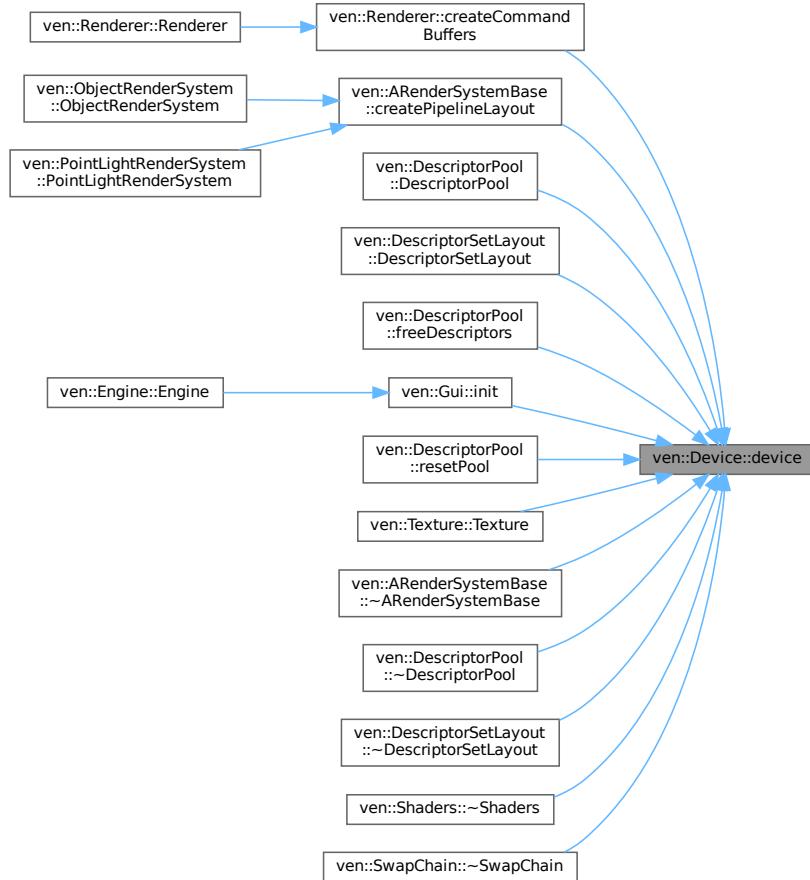
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

Definition at line 52 of file [Device.hpp](#).

References [m\\_device](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), [ven::DescriptorPool::DescriptorPool\(\)](#), [ven::DescriptorSetLayout::DescriptorsetLayout\(\)](#), [ven::DescriptorPool::freeDescriptors\(\)](#), [ven::Gui::init\(\)](#), [ven::DescriptorPool::resetPool\(\)](#), [ven::Texture::Texture\(\)](#), [ven::ARenderSystemBase::~ARenderSystemBase\(\)](#), [ven::DescriptorPool::~DescriptorPool\(\)](#), [ven::DescriptorSetLayout::~DescriptorsetLayout\(\)](#), [ven::Shaders::~Shaders\(\)](#), and [ven::SwapChain::~SwapChain\(\)](#).

Here is the caller graph for this function:



### 7.15.3.13 `endSingleTimeCommands()`

```
void ven::Device::endSingleTimeCommands (
    VkCommandBuffer commandBuffer) const
```

Definition at line 432 of file [device.cpp](#).

### 7.15.3.14 `findMemoryType()`

```
uint32_t ven::Device::findMemoryType (
    uint32_t typeFilter,
    VkMemoryPropertyFlags properties) const [nodiscard]
```

Definition at line 369 of file [device.cpp](#).

### 7.15.3.15 findPhysicalQueueFamilies()

```
QueueFamilyIndices ven::Device::findPhysicalQueueFamilies () const [inline], [nodiscard]
```

Definition at line 59 of file [Device.hpp](#).

References [findQueueFamilies\(\)](#), and [m\\_physicalDevice](#).

Here is the call graph for this function:



### 7.15.3.16 findQueueFamilies()

```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
    VkPhysicalDevice device) const [private]
```

Definition at line 306 of file [device.cpp](#).

References [ven::QueueFamilyIndices::graphicsFamily](#), [ven::QueueFamilyIndices::graphicsFamilyHasValue](#), [ven::QueueFamilyIndices::isComplete\(\)](#), [ven::QueueFamilyIndices::presentFamily](#), and [ven::QueueFamilyIndices::presentFamilyHasV](#)

Referenced by [findPhysicalQueueFamilies\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.15.3.17 `findSupportedFormat()`

```
VkFormat ven::Device::findSupportedFormat (
    const std::vector< VkFormat > & candidates,
    VkImageTiling tiling,
    VkFormatFeatureFlags features) const [nodiscard]
```

Definition at line 355 of file [device.cpp](#).

### 7.15.3.18 `getCommandPool()`

```
VkCommandPool ven::Device::getCommandPool () const [inline], [nodiscard]
```

Definition at line 51 of file [Device.hpp](#).

References [m\\_commandPool](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#).

Here is the caller graph for this function:



### 7.15.3.19 `getGraphicsQueue()`

```
VkQueue ven::Device::getGraphicsQueue () const [inline], [nodiscard]
```

Definition at line 61 of file [Device.hpp](#).

References [m\\_graphicsQueue](#).

### 7.15.3.20 `getInstance()`

```
VkInstance ven::Device::getInstance () const [inline], [nodiscard]
```

Definition at line 64 of file [Device.hpp](#).

References [m\\_instance](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



### 7.15.3.21 getPhysicalDevice()

```
VkPhysicalDevice ven::Device::getPhysicalDevice () const [inline], [nodiscard]
```

Definition at line 60 of file [Device.hpp](#).

References [m\\_physicalDevice](#).

Referenced by [ven::Gui::init\(\)](#).

Here is the caller graph for this function:



### 7.15.3.22 getProperties()

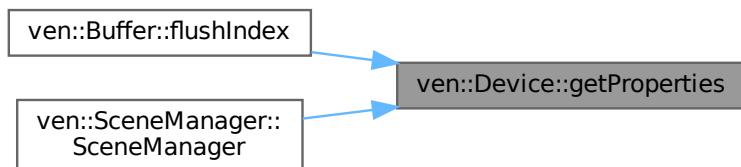
```
VkPhysicalDeviceProperties ven::Device::getProperties () const [inline], [nodiscard]
```

Definition at line 63 of file [Device.hpp](#).

References [m\\_properties](#).

Referenced by [ven::Buffer::flushIndex\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

Here is the caller graph for this function:



### 7.15.3.23 getRequiredExtensions()

```
std::vector< const char * > ven::Device::getRequiredExtensions () const [nodiscard], [private]
```

Definition at line 252 of file [device.cpp](#).

### 7.15.3.24 getSwapChainSupport()

```
SwapChainSupportDetails ven::Device::getSwapChainSupport () const [inline], [nodiscard]
```

Definition at line 57 of file [Device.hpp](#).

References [m\\_physicalDevice](#), and [querySwapChainSupport\(\)](#).

Here is the call graph for this function:



### 7.15.3.25 graphicsQueue()

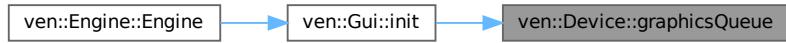
```
VkQueue ven::Device::graphicsQueue () const [inline], [nodiscard]
```

Definition at line 54 of file [Device.hpp](#).

References [m\\_graphicsQueue](#).

Referenced by [ven::Gui::init\(\)](#).

Here is the caller graph for this function:



### 7.15.3.26 hasGlfwRequiredInstanceExtensions()

```
void ven::Device::hasGlfwRequiredInstanceExtensions () const [private]
```

Definition at line 267 of file [device.cpp](#).

### 7.15.3.27 isDeviceSuitable()

```
bool ven::Device::isDeviceSuitable (
    VkPhysicalDevice device) const [private]
```

Definition at line 187 of file [device.cpp](#).

References [ven::QueueFamilyIndices::isComplete\(\)](#).

Here is the call graph for this function:



### 7.15.3.28 operator=( )

```
Device & ven::Device::operator= (
    const Device &) [delete]
```

### 7.15.3.29 pickPhysicalDevice()

```
void ven::Device::pickPhysicalDevice () [private]
```

Definition at line 98 of file [device.cpp](#).

Referenced by [Device\(\)](#).

Here is the caller graph for this function:



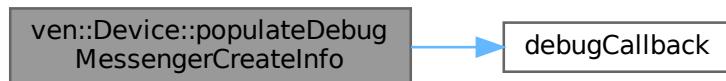
### 7.15.3.30 populateDebugMessengerCreateInfo()

```
void ven::Device::populateDebugMessengerCreateInfo (
    VkDebugUtilsMessengerCreateInfoEXT & createInfo) [static], [private]
```

Definition at line 204 of file [device.cpp](#).

References [debugCallback\(\)](#).

Here is the call graph for this function:



### 7.15.3.31 presentQueue()

```
VkQueue ven::Device::presentQueue () const [inline], [nodiscard]
```

Definition at line 55 of file [Device.hpp](#).

References [m\\_presentQueue](#).

### 7.15.3.32 querySwapChainSupport()

```
ven::SwapChainSupportDetails ven::Device::querySwapChainSupport (
    VkPhysicalDevice device) const [private]
```

Definition at line 334 of file [device.cpp](#).

References [ven::SwapChainSupportDetails::capabilities](#), [ven::SwapChainSupportDetails::formats](#), and [ven::SwapChainSupportDetails::rasterizationLayers](#).

Referenced by [getSwapChainSupport\(\)](#).

Here is the caller graph for this function:



### 7.15.3.33 setupDebugMessenger()

```
void ven::Device::setupDebugMessenger () [private]
```

Definition at line 217 of file [device.cpp](#).

References [CreateDebugUtilsMessengerEXT\(\)](#).

Referenced by [Device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.15.3.34 surface()

```
VkSurfaceKHR ven::Device::surface () const [inline], [nodiscard]
```

Definition at line 53 of file [Device.hpp](#).

References [m\\_surface](#).

### 7.15.3.35 transitionImageLayout()

```
void ven::Device::transitionImageLayout (
    VkImage image,
    VkFormat format,
    VkImageLayout oldLayout,
    VkImageLayout newLayout,
    uint32_t mipLevels = 1,
    uint32_t layerCount = 1) const
```

Definition at line 504 of file [device.cpp](#).

## 7.15.4 Member Data Documentation

### 7.15.4.1 enableValidationLayers

```
const bool ven:::Device::enableValidationLayers = true
```

Definition at line 42 of file [Device.hpp](#).

### 7.15.4.2 m\_commandPool

```
VkCommandPool ven:::Device::m_commandPool [private]
```

Definition at line 98 of file [Device.hpp](#).

Referenced by [getCommandPool\(\)](#).

### 7.15.4.3 m\_debugMessenger

```
VkDebugUtilsMessengerEXT ven:::Device::m_debugMessenger [private]
```

Definition at line 96 of file [Device.hpp](#).

### 7.15.4.4 m\_device

```
VkDevice ven:::Device::m_device [private]
```

Definition at line 99 of file [Device.hpp](#).

Referenced by [device\(\)](#).

### 7.15.4.5 m\_deviceExtensions

```
const std::vector<const char *> ven:::Device::m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME} [private]
```

Definition at line 107 of file [Device.hpp](#).

### 7.15.4.6 m\_graphicsQueue

```
VkQueue ven:::Device::m_graphicsQueue [private]
```

Definition at line 102 of file [Device.hpp](#).

Referenced by [getGraphicsQueue\(\)](#), and [graphicsQueue\(\)](#).

#### 7.15.4.7 m\_instance

```
VkInstance ven::Device::m_instance [private]
```

Definition at line 100 of file [Device.hpp](#).

Referenced by [createSurface\(\)](#), and [getInstance\(\)](#).

#### 7.15.4.8 m\_physicalDevice

```
VkPhysicalDevice ven::Device::m_physicalDevice = VK_NULL_HANDLE [private]
```

Definition at line 97 of file [Device.hpp](#).

Referenced by [findPhysicalQueueFamilies\(\)](#), [getPhysicalDevice\(\)](#), and [getSwapChainSupport\(\)](#).

#### 7.15.4.9 m\_presentQueue

```
VkQueue ven::Device::m_presentQueue [private]
```

Definition at line 103 of file [Device.hpp](#).

Referenced by [presentQueue\(\)](#).

#### 7.15.4.10 m\_properties

```
VkPhysicalDeviceProperties ven::Device::m_properties [private]
```

Definition at line 104 of file [Device.hpp](#).

Referenced by [getProperties\(\)](#).

#### 7.15.4.11 m\_surface

```
VkSurfaceKHR ven::Device::m_surface [private]
```

Definition at line 101 of file [Device.hpp](#).

Referenced by [createSurface\(\)](#), and [surface\(\)](#).

#### 7.15.4.12 m\_validationLayers

```
const std::vector<const char *> ven::Device::m_validationLayers = {"VK_LAYER_KHRONOS_validation"} [private]
```

Definition at line 106 of file [Device.hpp](#).

#### 7.15.4.13 m\_window

```
Window& ven::Device::m_window [private]
```

Definition at line 95 of file [Device.hpp](#).

Referenced by [createSurface\(\)](#).

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

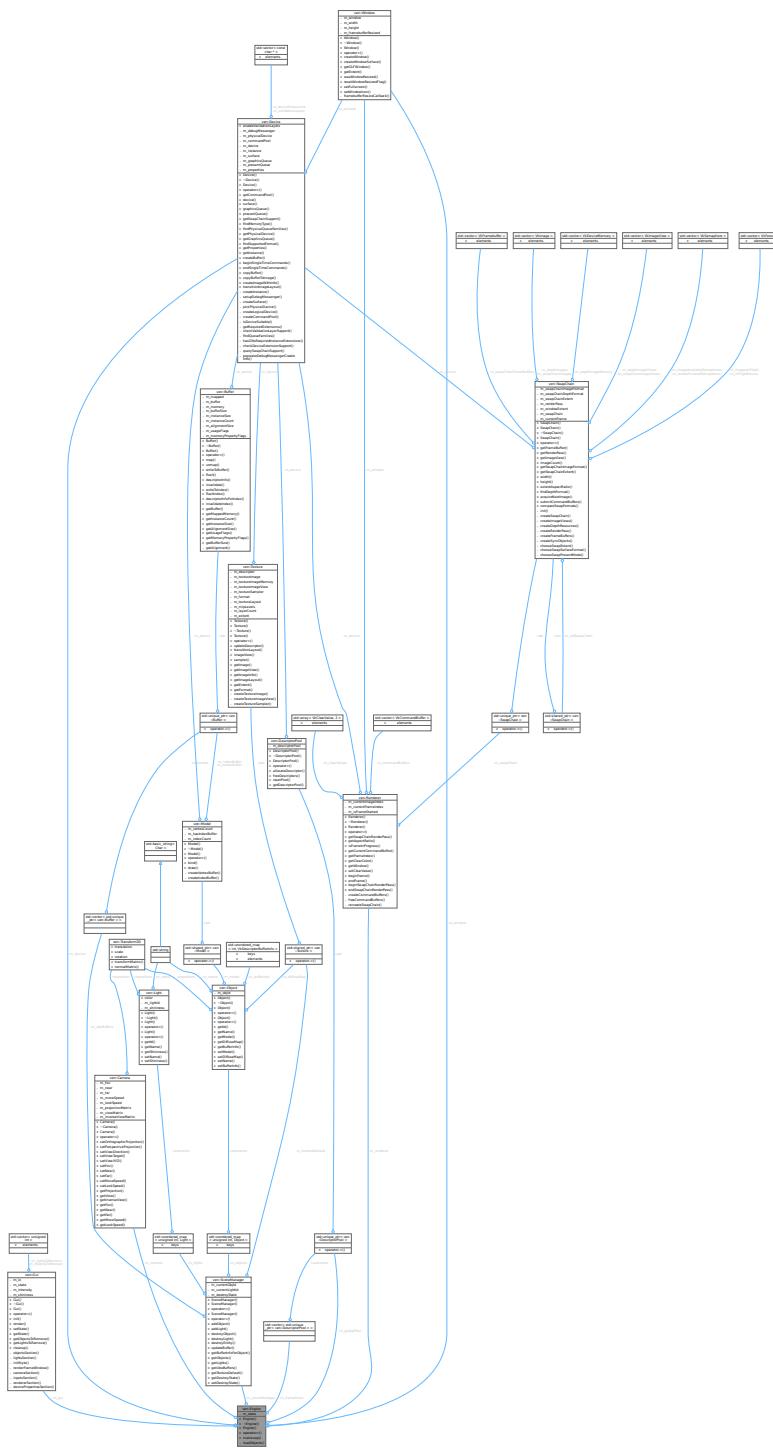
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp
- /home/runner/work/VEngine/VEngine/src/Core/[device.cpp](#)

## 7.16 ven::Engine Class Reference

Class for engine.

```
#include <Engine.hpp>
```

## Collaboration diagram for ven::Engine:



## Public Member Functions

- Engine (const Config &config)
  - ~Engine ()
  - Engine (const Engine &)=delete
  - Engine operator= (const Engine &)=delete
  - void mainLoop ()

### Private Member Functions

- void `loadObjects ()`

### Private Attributes

- `ENGINE_STATE m_state {EXIT}`
- `Window m_window`
- `Camera m_camera`
- `Gui m_gui`
- `Device m_device {m_window}`
- `SceneManager m_sceneManager {m_device}`
- `Renderer m_renderer {m_window, m_device}`
- `std::unique_ptr< DescriptorPool > m_globalPool`
- `std::vector< std::unique_ptr< DescriptorPool > > m_framePools`

## 7.16.1 Detailed Description

Class for engine.

Definition at line 23 of file `Engine.hpp`.

## 7.16.2 Constructor & Destructor Documentation

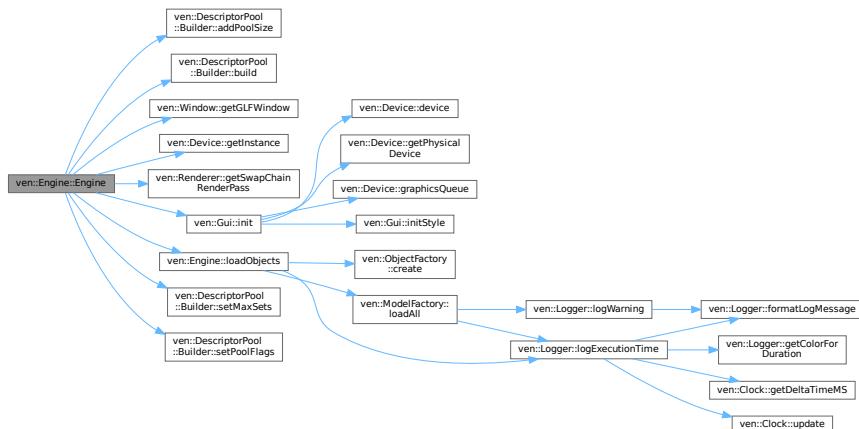
### 7.16.2.1 Engine() [1/2]

```
ven::Engine::Engine (
    const Config & config) [explicit]
```

Definition at line 13 of file `engine.cpp`.

References `ven::DescriptorPool::Builder::addPoolSize()`, `ven::DescriptorPool::Builder::build()`, `ven::EDITOR`, `ven::Window::getGLFWWindow()`, `ven::Device::getInstance()`, `ven::Renderer::getSwapChainRenderPass()`, `ven::Gui::init()`, `loadObjects()`, `m_device`, `m_framePools`, `m_globalPool`, `m_gui`, `m_renderer`, `m_window`, `ven::MAX_FRAMES_IN_FLIGHT`, `ven::DescriptorPool::Builder::setMaxSets()`, and `ven::DescriptorPool::Builder::setPoolFlags()`.

Here is the call graph for this function:



### 7.16.2.2 ~Engine()

```
ven::Engine::~Engine () [inline]
```

Definition at line 28 of file [Engine.hpp](#).

References [ven::Gui::cleanup\(\)](#).

Here is the call graph for this function:



### 7.16.2.3 Engine() [2/2]

```
ven::Engine::Engine (
    const Engine & ) [delete]
```

## 7.16.3 Member Function Documentation

### 7.16.3.1 loadObjects()

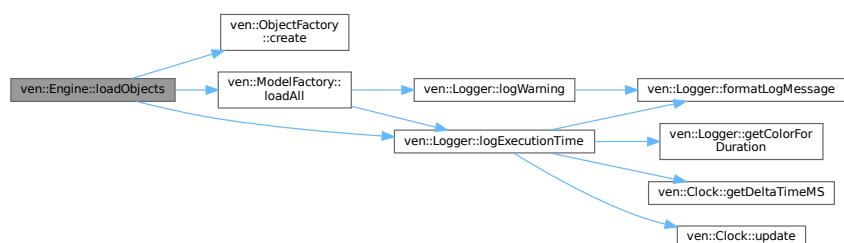
```
void ven::Engine::loadObjects () [private]
```

Definition at line 28 of file [engine.cpp](#).

References [ven::Colors::BLUE\\_4](#), [ven::ObjectFactory::create\(\)](#), [ven::Colors::CYAN\\_4](#), [ven::Colors::GREEN\\_4](#), [ven::ModelFactory::loadAll\(\)](#), [ven::Logger::logExecutionTime\(\)](#), [ven::Colors::MAGENTA\\_4](#), [ven::Colors::RED\\_4](#), and [ven::Colors::YELLOW\\_4](#).

Referenced by [Engine\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.16.3.2 mainLoop()

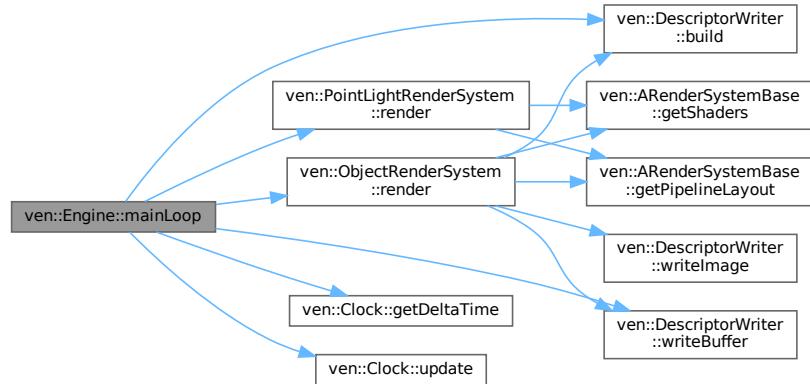
```
void ven::Engine::mainLoop ()
```

Definition at line 86 of file [engine.cpp](#).

References [ven::DescriptorWriter::build\(\)](#), [ven::EXIT](#), [ven::FrameInfo::frameIndex](#), [ven::Clock::getDeltaTime\(\)](#), [ven::HIDDEN](#), [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#), [ven::ObjectRenderSystem::render\(\)](#), [ven::PointLightRenderSystem::render\(\)](#), [ven::Clock::update\(\)](#), and [ven::DescriptorWriter::writeBuffer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.16.3.3 operator=([\(\)](#))

```
Engine ven::Engine::operator= (
    const Engine & )  [delete]
```

## 7.16.4 Member Data Documentation

### 7.16.4.1 m\_camera

```
Camera ven::Engine::m_camera  [private]
```

Definition at line [42](#) of file [Engine.hpp](#).

### 7.16.4.2 m\_device

```
Device ven::Engine::m_device {m_window}  [private]
```

Definition at line [44](#) of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

### 7.16.4.3 m\_framePools

```
std::vector<std::unique_ptr<DescriptorPool>> ven::Engine::m_framePools  [private]
```

Definition at line [48](#) of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

### 7.16.4.4 m\_globalPool

```
std::unique_ptr<DescriptorPool> ven::Engine::m_globalPool  [private]
```

Definition at line [47](#) of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

### 7.16.4.5 m\_gui

```
Gui ven::Engine::m_gui  [private]
```

Definition at line [43](#) of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

#### 7.16.4.6 m\_renderer

```
Renderer ven::Engine::m_renderer {m_window, m_device} [private]
```

Definition at line 46 of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

#### 7.16.4.7 m\_sceneManager

```
SceneManager ven::Engine::m_sceneManager {m_device} [private]
```

Definition at line 45 of file [Engine.hpp](#).

#### 7.16.4.8 m\_state

```
ENGINE_STATE ven::Engine::m_state {EXIT} [private]
```

Definition at line 39 of file [Engine.hpp](#).

#### 7.16.4.9 m\_window

```
Window ven::Engine::m_window [private]
```

Definition at line 41 of file [Engine.hpp](#).

Referenced by [Engine\(\)](#).

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

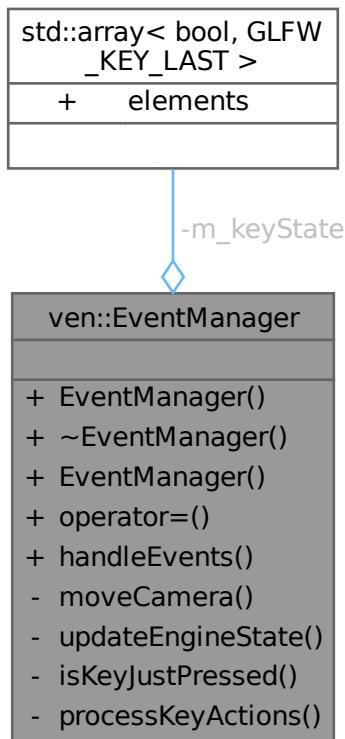
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Engine.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/[engine.cpp](#)

## 7.17 ven::EventManager Class Reference

Class for event manager.

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::EventManager:



### Public Member Functions

- `EventManager ()=default`
- `~EventManager ()=default`
- `EventManager (const EventManager &)=delete`
- `EventManager & operator= (const EventManager &)=delete`
- `void handleEvents (GLFWwindow *window, ENGINE_STATE *engineState, Camera &camera, Gui &gui, float dt) const`

### Static Private Member Functions

- `static void moveCamera (GLFWwindow *window, Camera &camera, float dt)`
- `static void updateEngineState (ENGINE_STATE *engineState, const ENGINE_STATE newState)`
- `static bool isKeyJustPressed (GLFWwindow *window, long unsigned int key, std::array< bool, GLFW_KEY_LAST > &keyStates)`
- `template<typename Iterator> static void processKeyActions (GLFWwindow *window, Iterator begin, Iterator end)`

### Private Attributes

- `std::array< bool, GLFW_KEY_LAST > m_keyState {}`

### 7.17.1 Detailed Description

Class for event manager.

Definition at line 42 of file [EventManager.hpp](#).

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 EventManager() [1/2]

```
ven::EventManager::EventManager () [default]
```

#### 7.17.2.2 ~EventManager()

```
ven::EventManager::~EventManager () [default]
```

#### 7.17.2.3 EventManager() [2/2]

```
ven::EventManager::EventManager (
    const EventManager & ) [delete]
```

### 7.17.3 Member Function Documentation

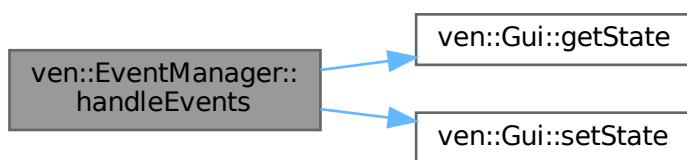
#### 7.17.3.1 handleEvents()

```
void ven::EventManager::handleEvents (
    GLFWwindow * window,
    ENGINE_STATE * engineState,
    Camera & camera,
    Gui & gui,
    float dt) const
```

Definition at line 60 of file [eventManager.cpp](#).

References [ven::DEFAULT\\_KEY\\_MAPPINGS](#), [ven::EDITOR](#), [ven::EXIT](#), [ven::Gui::getState\(\)](#), [ven::HIDDEN](#), [ven::Gui::setState\(\)](#), [ven::SHOW\\_EDITOR](#), [ven::SHOW\\_PLAYER](#), and [ven::KeyMappings::toggleGui](#).

Here is the call graph for this function:



### 7.17.3.2 isKeyJustPressed()

```
bool ven::EventManager::isKeyJustPressed (
    GLFWwindow * window,
    long unsigned int key,
    std::array< bool, GLFW_KEY_LAST > & keyStates) [static], [private]
```

Definition at line 6 of file [eventManager.cpp](#).

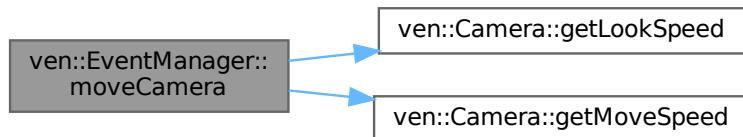
### 7.17.3.3 moveCamera()

```
void ven::EventManager::moveCamera (
    GLFWwindow * window,
    Camera & camera,
    float dt) [static], [private]
```

Definition at line 26 of file [eventManager.cpp](#).

References [ven::DEFAULT\\_KEY\\_MAPPINGS](#), [ven::EPSILON](#), [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::KeyMappings::lookDown](#), [ven::KeyMappings::lookLeft](#), [ven::KeyMappings::lookRight](#), [ven::KeyMappings::lookUp](#), [ven::KeyMappings::moveBackward](#), [ven::KeyMappings::moveDown](#), [ven::KeyMappings::moveForward](#), [ven::KeyMappings::moveLeft](#), [ven::KeyMappings::moveRight](#), [ven::KeyMappings::moveUp](#), [ven::Transform3D::rotation](#), [ven::Camera::transform](#), and [ven::Transform3D::translation](#).

Here is the call graph for this function:



### 7.17.3.4 operator=()

```
EventManager & ven::EventManager::operator= (
    const EventManager & ) [delete]
```

### 7.17.3.5 processKeyActions()

```
template<typename Iterator >
void ven::EventManager::processKeyActions (
    GLFWwindow * window,
    Iterator begin,
    Iterator end) [static], [private]
```

Definition at line 17 of file [eventManager.cpp](#).

### 7.17.3.6 updateEngineState()

```
static void ven::EventManager::updateEngineState (
    ENGINE_STATE * engineState,
    const ENGINE_STATE newState) [inline], [static], [private]
```

Definition at line [57](#) of file [EventManager.hpp](#).

## 7.17.4 Member Data Documentation

### 7.17.4.1 m\_keyState

```
std::array<bool, GLFW_KEY_LAST> ven::EventManager::m_keyState {} [mutable], [private]
```

Definition at line [63](#) of file [EventManager.hpp](#).

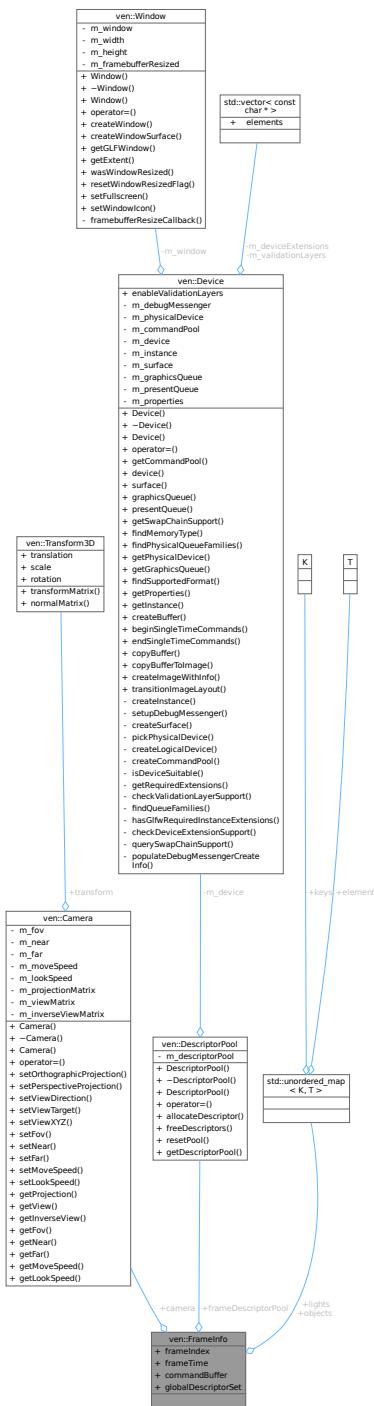
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[EventManager.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/[eventManager.cpp](#)

## 7.18 ven::FrameInfo Struct Reference

```
#include <FrameInfo.hpp>
```

## Collaboration diagram for ven::FrameInfo:



## Public Attributes

- unsigned long `frameIndex`
  - float `frameTime`
  - `VkCommandBuffer commandBuffer`
  - `Camera & camera`
  - `VkDescriptorSet globalDescriptorSet`

- `DescriptorPool` & `frameDescriptorPool`
- `Object::Map` & `objects`
- `Light::Map` & `lights`

### 7.18.1 Detailed Description

Definition at line 44 of file [FrameInfo.hpp](#).

### 7.18.2 Member Data Documentation

#### 7.18.2.1 camera

`Camera&` `ven::FrameInfo::camera`

Definition at line 49 of file [FrameInfo.hpp](#).

#### 7.18.2.2 commandBuffer

`VkCommandBuffer` `ven::FrameInfo::commandBuffer`

Definition at line 48 of file [FrameInfo.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

#### 7.18.2.3 frameDescriptorPool

`DescriptorPool&` `ven::FrameInfo::frameDescriptorPool`

Definition at line 51 of file [FrameInfo.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#).

#### 7.18.2.4 frameIndex

`unsigned long` `ven::FrameInfo::frameIndex`

Definition at line 46 of file [FrameInfo.hpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#), and [ven::ObjectRenderSystem::render\(\)](#).

#### 7.18.2.5 frameTime

`float` `ven::FrameInfo::frameTime`

Definition at line 47 of file [FrameInfo.hpp](#).

### 7.18.2.6 globalDescriptorSet

```
VkDescriptorSet ven::FrameInfo::globalDescriptorSet
```

Definition at line 50 of file [FrameInfo.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

### 7.18.2.7 lights

```
Light::Map& ven::FrameInfo::lights
```

Definition at line 53 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::render\(\)](#).

### 7.18.2.8 objects

```
Object::Map& ven::FrameInfo::objects
```

Definition at line 52 of file [FrameInfo.hpp](#).

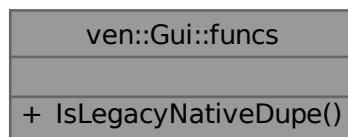
Referenced by [ven::ObjectRenderSystem::render\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[FrameInfo.hpp](#)

## 7.19 ven::Gui::funcs Struct Reference

Collaboration diagram for ven::Gui::funcs:



### Static Public Member Functions

- static bool [IsLegacyNativeDupe](#) (const ImGuiKey key)

### 7.19.1 Detailed Description

Definition at line 66 of file [Gui.hpp](#).

### 7.19.2 Member Function Documentation

#### 7.19.2.1 `IsLegacyNativeDupe()`

```
static bool ven::Gui::funcs::IsLegacyNativeDupe (
    const ImGuiKey key) [inline], [static]
```

Definition at line 66 of file [Gui.hpp](#).

References [IsLegacyNativeDupe\(\)](#).

Referenced by [IsLegacyNativeDupe\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



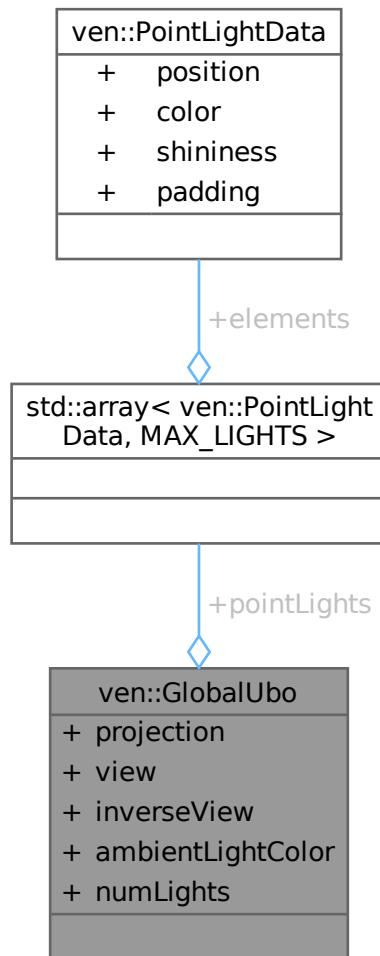
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp

## 7.20 ven::GlobalUbo Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::GlobalUbo:



### Public Attributes

- glm::mat4 `projection` {1.F}
- glm::mat4 `view` {1.F}
- glm::mat4 `inverseView` {1.F}
- glm::vec4 `ambientLightColor` {DEFAULT\_AMBIENT\_LIGHT\_COLOR}
- std::array< PointLightData, MAX\_LIGHTS > `pointLights`
- uint8\_t `numLights`

### 7.20.1 Detailed Description

Definition at line 34 of file [FrameInfo.hpp](#).

## 7.20.2 Member Data Documentation

### 7.20.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {DEFAULT_AMBIENT_LIGHT_COLOR}
```

Definition at line 39 of file [FrameInfo.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

### 7.20.2.2 inverseView

```
glm::mat4 ven::GlobalUbo::inverseView {1.F}
```

Definition at line 38 of file [FrameInfo.hpp](#).

### 7.20.2.3 numLights

```
uint8_t ven::GlobalUbo::numLights
```

Definition at line 41 of file [FrameInfo.hpp](#).

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

### 7.20.2.4 pointLights

```
std::array<PointLightData, MAX_LIGHTS> ven::GlobalUbo::pointLights
```

Definition at line 40 of file [FrameInfo.hpp](#).

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

### 7.20.2.5 projection

```
glm::mat4 ven::GlobalUbo::projection {1.F}
```

Definition at line 36 of file [FrameInfo.hpp](#).

### 7.20.2.6 view

```
glm::mat4 ven::GlobalUbo::view {1.F}
```

Definition at line 37 of file [FrameInfo.hpp](#).

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

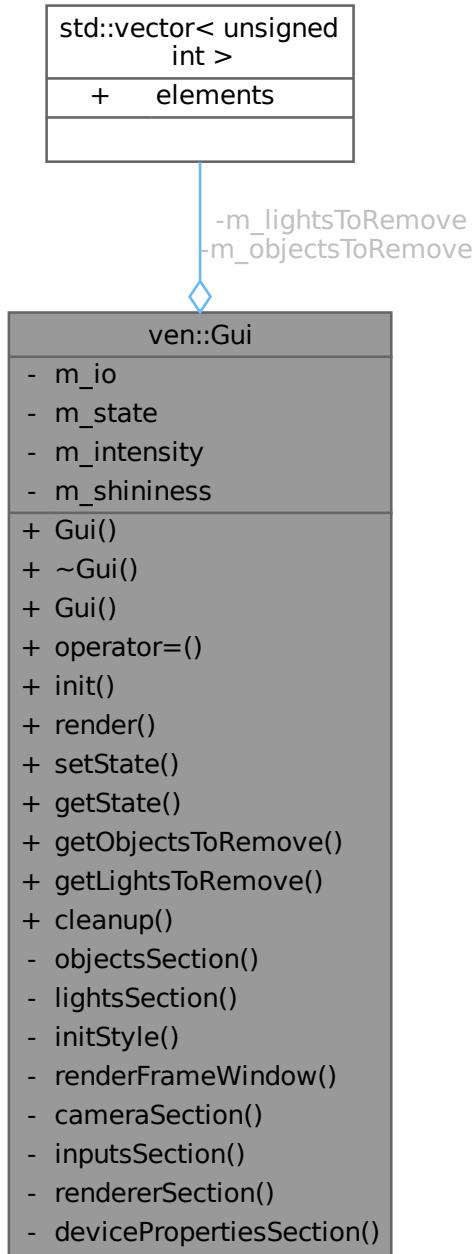
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[FrameInfo.hpp](#)

## 7.21 ven::Gui Class Reference

Class for [Gui](#).

```
#include <Gui.hpp>
```

Collaboration diagram for ven::Gui:



### Classes

- struct [ClockData](#)
- struct [funcs](#)

## Public Member Functions

- `Gui ()=default`
- `~Gui ()=default`
- `Gui (const Gui &)=delete`
- `Gui & operator= (const Gui &)=delete`
- `void init (GLFWwindow *window, VkInstance instance, const Device *device, VkRenderPass renderPass)`
- `void render (Renderer *renderer, SceneManager &sceneManager, Camera &camera, VkPhysicalDevice physicalDevice, GlobalUbo &ubo, const ClockData &clockData)`
- `void setState (const GUI_STATE state)`
- `GUI_STATE getState () const`
- `std::vector< unsigned int > * getObjectsToRemove ()`
- `std::vector< unsigned int > * getLightsToRemove ()`

## Static Public Member Functions

- `static void cleanup ()`

## Private Member Functions

- `void objectsSection (SceneManager &sceneManager)`
- `void lightsSection (SceneManager &sceneManager)`

## Static Private Member Functions

- `static void initStyle ()`
- `static void renderFrameWindow (const ClockData &clockData)`
- `static void cameraSection (Camera &camera)`
- `static void inputsSection (const ImGuilO &io)`
- `static void rendererSection (Renderer *renderer, GlobalUbo &ubo)`
- `static void devicePropertiesSection (VkPhysicalDeviceProperties deviceProperties)`

## Private Attributes

- `ImGuiIO * m_io {nullptr}`
- `GUI_STATE m_state {HIDDEN}`
- `float m_intensity {1.0F}`
- `float m_shininess {DEFAULT_SHININESS}`
- `std::vector< unsigned int > m_objectsToRemove`
- `std::vector< unsigned int > m_lightsToRemove`

### 7.21.1 Detailed Description

Class for [Gui](#).

Definition at line 30 of file [Gui.hpp](#).

## 7.21.2 Constructor & Destructor Documentation

### 7.21.2.1 Gui() [1/2]

```
ven::Gui::Gui () [default]
```

### 7.21.2.2 ~Gui()

```
ven::Gui::~Gui () [default]
```

### 7.21.2.3 Gui() [2/2]

```
ven::Gui::Gui (
    const Gui & ) [delete]
```

## 7.21.3 Member Function Documentation

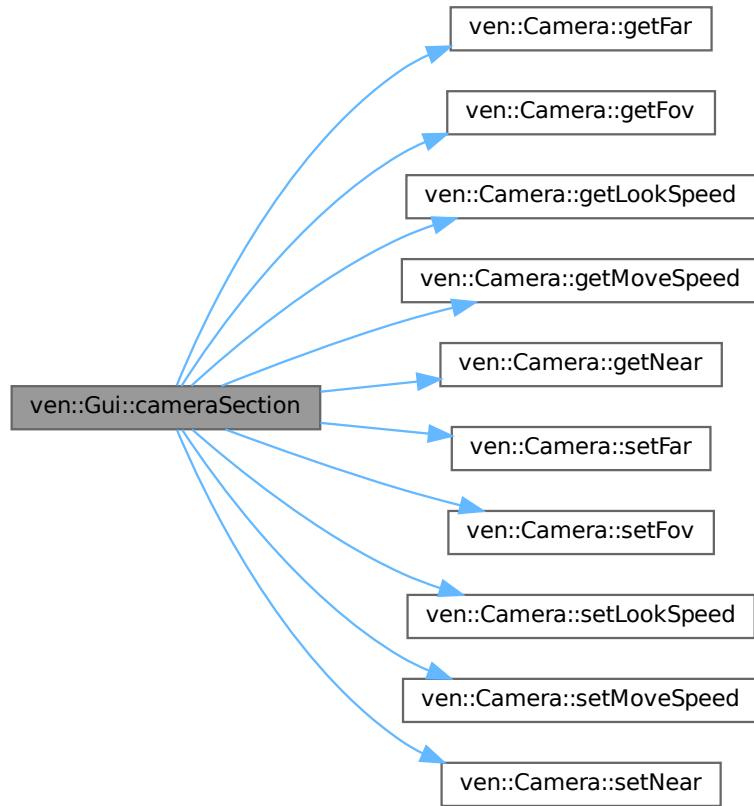
### 7.21.3.1 cameraSection()

```
void ven::Gui::cameraSection (
    Camera & camera) [static], [private]
```

Definition at line 109 of file [render.cpp](#).

References [ven::DEFAULT\\_FAR](#), [ven::DEFAULT\\_FOV](#), [ven::DEFAULT\\_LOOK\\_SPEED](#), [ven::DEFAULT\\_MOVE\\_SPEED](#), [ven::DEFAULT\\_NEAR](#), [ven::DEFAULT\\_POSITION](#), [ven::DEFAULT\\_ROTATION](#), [ven::Camera::getFar\(\)](#), [ven::Camera::getFov\(\)](#), [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::Camera::getNear\(\)](#), [ven::Transform3D::rotation](#), [ven::Camera::setFar\(\)](#), [ven::Camera::setFov\(\)](#), [ven::Camera::setLookSpeed\(\)](#), [ven::Camera::setMoveSpeed\(\)](#), [ven::Camera::setNear\(\)](#), [ven::Camera::transform](#), and [ven::Transform3D::translation](#).

Here is the call graph for this function:



### 7.21.3.2 cleanup()

```
void ven::Gui::cleanup () [static]
```

Definition at line 11 of file [render.cpp](#).

Referenced by [ven::Engine::~Engine\(\)](#).

Here is the caller graph for this function:



### 7.21.3.3 devicePropertiesSection()

```
void ven::Gui::devicePropertiesSection (
    VkPhysicalDeviceProperties deviceProperties) [static], [private]
```

Definition at line 304 of file [render.cpp](#).

### 7.21.3.4 getLightsToRemove()

```
std::vector< unsigned int > * ven::Gui::getLightsToRemove () [inline], [nodiscard]
```

Definition at line 53 of file [Gui.hpp](#).

References [m\\_lightsToRemove](#).

### 7.21.3.5 getObjectsToRemove()

```
std::vector< unsigned int > * ven::Gui::getObjectsToRemove () [inline], [nodiscard]
```

Definition at line 52 of file [Gui.hpp](#).

References [m\\_objectsToRemove](#).

### 7.21.3.6 getState()

```
GUI_STATE ven::Gui::getState () const [inline], [nodiscard]
```

Definition at line 51 of file [Gui.hpp](#).

References [m\\_state](#).

Referenced by [ven::EventManager::handleEvents\(\)](#).

Here is the caller graph for this function:



### 7.21.3.7 init()

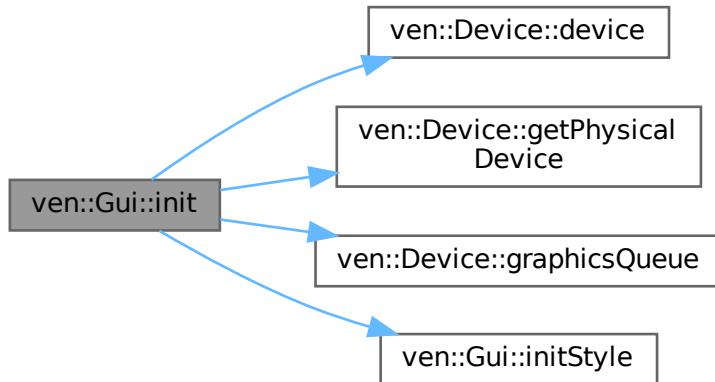
```
void ven::Gui::init (
    GLFWwindow * window,
    VkInstance instance,
    const Device * device,
    VkRenderPass renderPass)
```

Definition at line 6 of file [init.cpp](#).

References [ven::DESCRIPTOR\\_COUNT](#), [ven::Device::device\(\)](#), [ven::Device::getPhysicalDevice\(\)](#), [ven::Device::graphicsQueue\(\)](#), [initStyle\(\)](#), and [m\\_io](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.21.3.8 initStyle()

```
void ven::Gui::initStyle () [static], [private]
```

Definition at line 54 of file [init.cpp](#).

Referenced by [init\(\)](#).

Here is the caller graph for this function:



### 7.21.3.9 inputsSection()

```
void ven::Gui::inputsSection (
    const ImGuiIO & io) [static], [private]
```

Definition at line 282 of file [render.cpp](#).

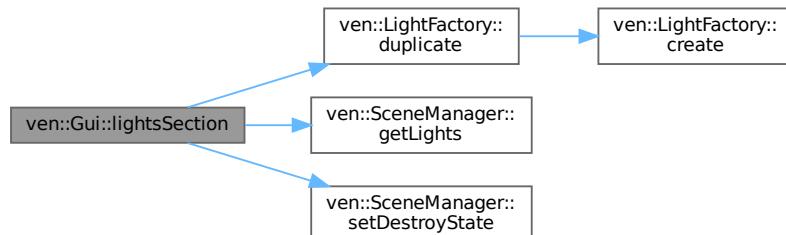
### 7.21.3.10 lightsSection()

```
void ven::Gui::lightsSection (
    SceneManager & sceneManager) [private]
```

Definition at line 185 of file [render.cpp](#).

References [ven::Colors::COLOR\\_PRESETS\\_3](#), [ven::DEFAULT\\_LIGHT\\_INTENSITY](#), [ven::DEFAULT\\_SHININESS](#), [ven::LightFactory::duplicate\(\)](#), [ven::SceneManager::getLights\(\)](#), and [ven::SceneManager::setDestroyState\(\)](#).

Here is the call graph for this function:



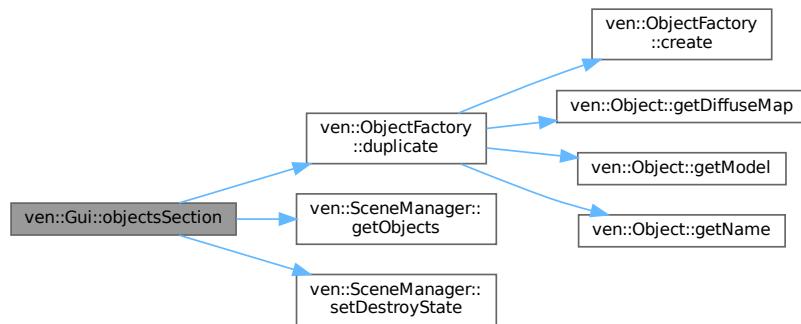
### 7.21.3.11 objectsSection()

```
void ven::Gui::objectsSection (
    SceneManager & sceneManager) [private]
```

Definition at line 158 of file [render.cpp](#).

References [ven::ObjectFactory::duplicate\(\)](#), [ven::SceneManager::getObjects\(\)](#), [ven::Colors::GRAY\\_4](#), and [ven::SceneManager::setDestroyState\(\)](#).

Here is the call graph for this function:



### 7.21.3.12 operator=()

```
Gui & ven::Gui::operator= (
    const Gui &) [delete]
```

### 7.21.3.13 render()

```
void ven::Gui::render (
    Renderer * renderer,
    SceneManager & sceneManager,
    Camera & camera,
    VkPhysicalDevice physicalDevice,
    GlobalUbo & ubo,
    const ClockData & clockData)
```

Definition at line 18 of file [render.cpp](#).

References [ven::Renderer::getCurrentCommandBuffer\(\)](#).

Here is the call graph for this function:



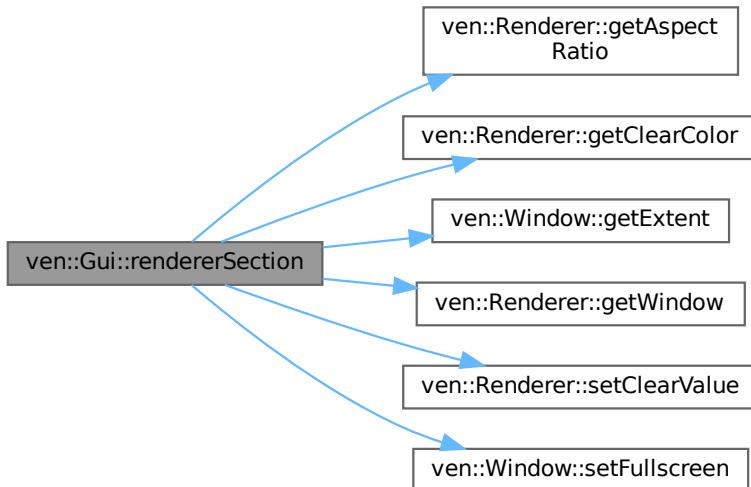
### 7.21.3.14 rendererSection()

```
void ven::Gui::rendererSection (
    Renderer * renderer,
    GlobalUbo & ubo) [static], [private]
```

Definition at line 48 of file [render.cpp](#).

References [ven::GlobalUbo::ambientLightColor](#), [ven::Colors::COLOR\\_PRESETS\\_4](#), [ven::Colors::COLOR\\_PRESETS\\_VK](#), [ven::DEFAULT\\_AMBIENT\\_LIGHT\\_INTENSITY](#), [ven::Renderer::getAspectRatio\(\)](#), [ven::Renderer::getClearColor\(\)](#), [ven::Window::getExtent\(\)](#), [ven::Renderer::getWindow\(\)](#), [ven::Renderer::setClearValue\(\)](#), and [ven::Window::setFullscreen\(\)](#).

Here is the call graph for this function:



### 7.21.3.15 renderFrameWindow()

```
void ven::Gui::renderFrameWindow (
    const ClockData & clockData) [static], [private]
```

Definition at line 39 of file [render.cpp](#).

References [ven::Gui::ClockData::deltaTimeMS](#), and [ven::Gui::ClockData::fps](#).

### 7.21.3.16 setState()

```
void ven::Gui::setState (
    const GUI_STATE state) [inline]
```

Definition at line 50 of file [Gui.hpp](#).

References [m\\_state](#).

Referenced by [ven::EventManager::handleEvents\(\)](#).

Here is the caller graph for this function:



## 7.21.4 Member Data Documentation

### 7.21.4.1 m\_intensity

```
float ven::Gui::m_intensity {1.0F} [private]
```

Definition at line [70](#) of file [Gui.hpp](#).

### 7.21.4.2 m\_io

```
ImGuiIO* ven::Gui::m_io {nullptr} [private]
```

Definition at line [68](#) of file [Gui.hpp](#).

Referenced by [init\(\)](#).

### 7.21.4.3 m\_lightsToRemove

```
std::vector<unsigned int> ven::Gui::m_lightsToRemove [private]
```

Definition at line [74](#) of file [Gui.hpp](#).

Referenced by [getLightsToRemove\(\)](#).

### 7.21.4.4 m\_objectsToRemove

```
std::vector<unsigned int> ven::Gui::m_objectsToRemove [private]
```

Definition at line [73](#) of file [Gui.hpp](#).

Referenced by [getObjectsToRemove\(\)](#).

### 7.21.4.5 m\_shininess

```
float ven::Gui::m_shininess {DEFAULT_SHININESS} [private]
```

Definition at line 71 of file [Gui.hpp](#).

### 7.21.4.6 m\_state

```
GUI_STATE ven::Gui::m_state {HIDDEN} [private]
```

Definition at line 69 of file [Gui.hpp](#).

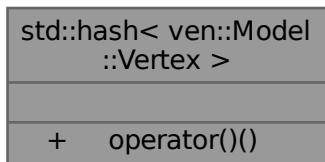
Referenced by [getState\(\)](#), and [setState\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp
- /home/runner/work/VEngine/VEngine/src/Core/GUI/[init.cpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/GUI/[render.cpp](#)

## 7.22 std::hash< ven::Model::Vertex > Struct Reference

Collaboration diagram for std::hash< ven::Model::Vertex >:



### Public Member Functions

- `size_t operator() (ven::Model::Vertex const &vertex) const noexcept`

### 7.22.1 Detailed Description

Definition at line 12 of file [model.cpp](#).

## 7.22.2 Member Function Documentation

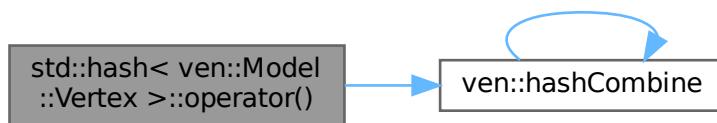
### 7.22.2.1 operator()()

```
size_t std::hash< ven::Model::Vertex >::operator() (   
    ven::Model::Vertex const & vertex) const [inline], [noexcept]
```

Definition at line 13 of file [model.cpp](#).

References [ven::hashCombine\(\)](#).

Here is the call graph for this function:



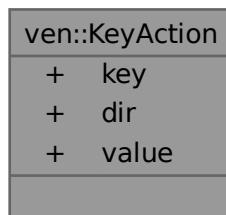
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

## 7.23 ven::KeyAction Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyAction:



## Public Attributes

- `uint16_t key`
- `glm::vec3 * dir`
- `glm::vec3 value`

### 7.23.1 Detailed Description

Definition at line 14 of file [EventManager.hpp](#).

### 7.23.2 Member Data Documentation

#### 7.23.2.1 dir

```
glm::vec3* ven::KeyAction::dir
```

Definition at line 16 of file [EventManager.hpp](#).

#### 7.23.2.2 key

```
uint16_t ven::KeyAction::key
```

Definition at line 15 of file [EventManager.hpp](#).

#### 7.23.2.3 value

```
glm::vec3 ven::KeyAction::value
```

Definition at line 17 of file [EventManager.hpp](#).

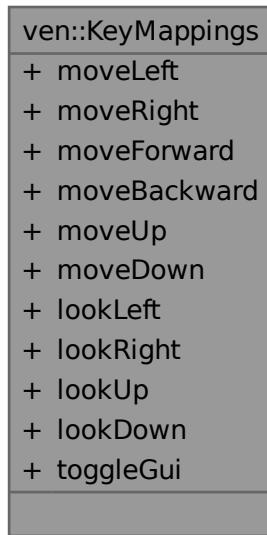
The documentation for this struct was generated from the following file:

- `/home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp`

## 7.24 ven::KeyMappings Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyMappings:



### Public Attributes

- uint16\_t **moveLeft** = GLFW\_KEY\_A
- uint16\_t **moveRight** = GLFW\_KEY\_D
- uint16\_t **moveForward** = GLFW\_KEY\_W
- uint16\_t **moveBackward** = GLFW\_KEY\_S
- uint16\_t **moveUp** = GLFW\_KEY\_SPACE
- uint16\_t **moveDown** = GLFW\_KEY\_LEFT\_SHIFT
- uint16\_t **lookLeft** = GLFW\_KEY\_LEFT
- uint16\_t **lookRight** = GLFW\_KEY\_RIGHT
- uint16\_t **lookUp** = GLFW\_KEY\_UP
- uint16\_t **lookDown** = GLFW\_KEY\_DOWN
- uint16\_t **toggleGui** = GLFW\_KEY\_0

### 7.24.1 Detailed Description

Definition at line 20 of file [EventManager.hpp](#).

## 7.24.2 Member Data Documentation

### 7.24.2.1 lookDown

```
uint16_t ven::KeyMappings::lookDown = GLFW_KEY_DOWN
```

Definition at line 30 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.2 lookLeft

```
uint16_t ven::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

Definition at line 27 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.3 lookRight

```
uint16_t ven::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

Definition at line 28 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.4 lookUp

```
uint16_t ven::KeyMappings::lookUp = GLFW_KEY_UP
```

Definition at line 29 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.5 moveBackward

```
uint16_t ven::KeyMappings::moveBackward = GLFW_KEY_S
```

Definition at line 24 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.6 moveDown

```
uint16_t ven::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

Definition at line 26 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.7 moveForward

```
uint16_t ven::KeyMappings::moveForward = GLFW_KEY_W
```

Definition at line 23 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.8 moveLeft

```
uint16_t ven::KeyMappings::moveLeft = GLFW_KEY_A
```

Definition at line 21 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.9 moveRight

```
uint16_t ven::KeyMappings::moveRight = GLFW_KEY_D
```

Definition at line 22 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.10 moveUp

```
uint16_t ven::KeyMappings::moveUp = GLFW_KEY_SPACE
```

Definition at line 25 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

### 7.24.2.11 toggleGui

```
uint16_t ven::KeyMappings::toggleGui = GLFW_KEY_0
```

Definition at line 31 of file [EventManager.hpp](#).

Referenced by [ven::EventManager::handleEvents\(\)](#).

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

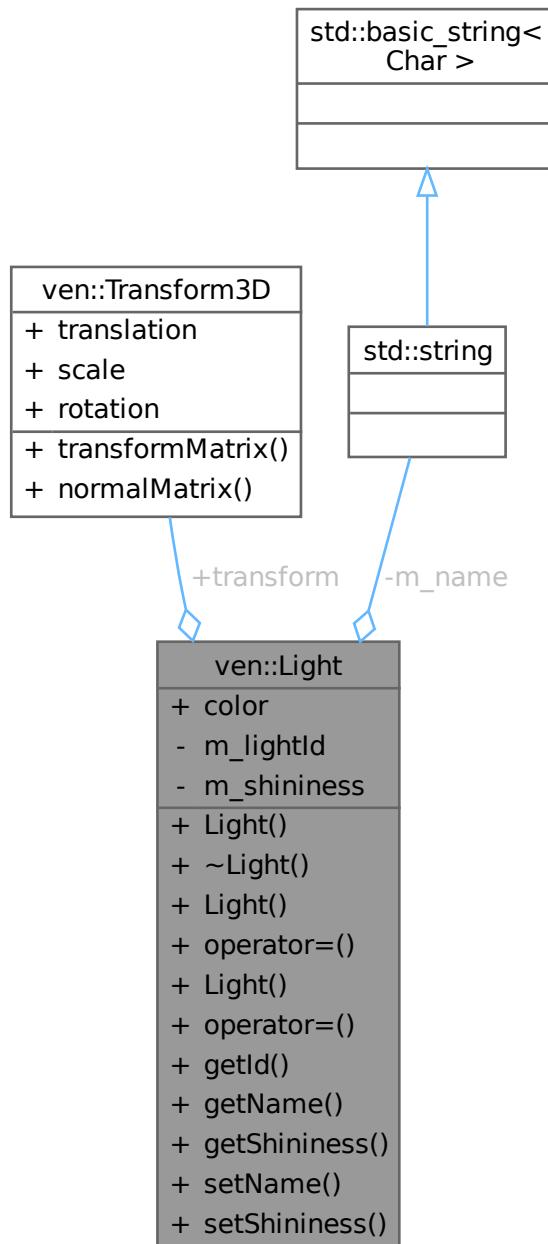
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp

## 7.25 ven::Light Class Reference

Class for light.

```
#include <Light.hpp>
```

Collaboration diagram for ven::Light:



### Public Types

- using `Map` = `std::unordered_map<unsigned int, Light>`

## Public Member Functions

- `Light (const unsigned int objId)`
- `~Light ()=default`
- `Light (const Light &)=delete`
- `Light & operator= (const Light &)=delete`
- `Light (Light &&)=default`
- `Light & operator= (Light &&)=default`
- `unsigned int getId () const`
- `std::string getName () const`
- `float getShininess () const`
- `void setName (const std::string &name)`
- `void setShininess (const float shininess)`

## Public Attributes

- `glm::vec4 color {DEFAULT_LIGHT_COLOR}`
- `Transform3D transform {}`

## Private Attributes

- `unsigned int m_lightId`
- `std::string m_name {"point light"}`
- `float m_shininess {DEFAULT_SHININESS}`

### 7.25.1 Detailed Description

Class for light.

Definition at line 28 of file [Light.hpp](#).

### 7.25.2 Member Typedef Documentation

#### 7.25.2.1 Map

```
using ven::Light::Map = std::unordered_map<unsigned int, Light>
```

Definition at line 32 of file [Light.hpp](#).

### 7.25.3 Constructor & Destructor Documentation

#### 7.25.3.1 Light() [1/3]

```
ven::Light::Light (
    const unsigned int objId) [inline], [explicit]
```

Definition at line 34 of file [Light.hpp](#).

### 7.25.3.2 ~Light()

```
ven::Light::~Light () [default]
```

### 7.25.3.3 Light() [2/3]

```
ven::Light::Light (
    const Light & ) [delete]
```

### 7.25.3.4 Light() [3/3]

```
ven::Light::Light (
    Light && ) [default]
```

## 7.25.4 Member Function Documentation

### 7.25.4.1 getId()

```
unsigned int ven::Light::getId () const [inline], [nodiscard]
```

Definition at line 43 of file [Light.hpp](#).

References [m\\_lightId](#).

### 7.25.4.2 getName()

```
std::string ven::Light::getName () const [inline], [nodiscard]
```

Definition at line 44 of file [Light.hpp](#).

References [m\\_name](#).

### 7.25.4.3 getShininess()

```
float ven::Light::getShininess () const [inline], [nodiscard]
```

Definition at line 45 of file [Light.hpp](#).

References [m\\_shininess](#).

### 7.25.4.4 operator=() [1/2]

```
Light & ven::Light::operator= (
    const Light & ) [delete]
```

#### 7.25.4.5 **operator=()** [2/2]

```
Light & ven::Light::operator= (
    Light &&) [default]
```

#### 7.25.4.6 **setName()**

```
void ven::Light::setName (
    const std::string & name) [inline]
```

Definition at line 47 of file [Light.hpp](#).

References [m\\_name](#).

#### 7.25.4.7 **setShininess()**

```
void ven::Light::setShininess (
    const float shininess) [inline]
```

Definition at line 48 of file [Light.hpp](#).

References [m\\_shininess](#).

### 7.25.5 Member Data Documentation

#### 7.25.5.1 **color**

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

Definition at line 50 of file [Light.hpp](#).

Referenced by [ven::LightFactory::duplicate\(\)](#).

#### 7.25.5.2 **m\_lightId**

```
unsigned int ven::Light::m_lightId [private]
```

Definition at line 55 of file [Light.hpp](#).

Referenced by [getId\(\)](#).

#### 7.25.5.3 **m\_name**

```
std::string ven::Light::m_name {"point light"} [private]
```

Definition at line 56 of file [Light.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

#### 7.25.5.4 m\_shininess

```
float ven::Light::m_shininess {DEFAULT_SHININESS} [private]
```

Definition at line 57 of file [Light.hpp](#).

Referenced by [getShininess\(\)](#), and [setShininess\(\)](#).

#### 7.25.5.5 transform

```
Transform3D ven::Light::transform {}
```

Definition at line 51 of file [Light.hpp](#).

Referenced by [ven::LightFactory::duplicate\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/[Light.hpp](#)

## 7.26 ven::LightFactory Class Reference

Class for light factory.

```
#include <Light.hpp>
```

Collaboration diagram for ven::LightFactory:

ven::LightFactory
- m_currentLightId
+ LightFactory()
+ ~LightFactory()
+ LightFactory()
+ operator=(())
+ operator=(())
+ create()
+ duplicate()

## Public Member Functions

- `LightFactory ()=delete`
- `~LightFactory ()=default`
- `LightFactory (const LightFactory &)=delete`
- `LightFactory & operator= (const LightFactory &)=delete`
- `LightFactory (LightFactory &&)=delete`
- `LightFactory & operator= (LightFactory &&)=delete`

## Static Public Member Functions

- `static std::unique_ptr< Light > create (const Transform3D &transform=DEFAULT_TRANSFORM, glm::vec4 color=DEFAULT_LIGHT_COLOR)`
- `static std::unique_ptr< Light > duplicate (const Light &cpyLight)`

## Static Private Attributes

- `static unsigned int m_currentLightId = 0`

### 7.26.1 Detailed Description

Class for light factory.

Definition at line 22 of file [Light.hpp](#).

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 LightFactory() [1/3]

```
ven::LightFactory::LightFactory () [delete]
```

#### 7.26.2.2 ~LightFactory()

```
ven::LightFactory::~LightFactory () [default]
```

#### 7.26.2.3 LightFactory() [2/3]

```
ven::LightFactory::LightFactory (
    const LightFactory & ) [delete]
```

#### 7.26.2.4 LightFactory() [3/3]

```
ven::LightFactory::LightFactory (
    LightFactory && ) [delete]
```

### 7.26.3 Member Function Documentation

#### 7.26.3.1 create()

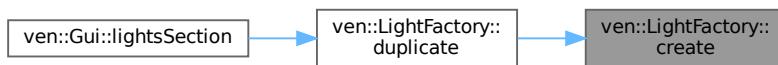
```
std::unique_ptr< ven::Light > ven::LightFactory::create (
    const Transform3D & transform = DEFAULT_TRANSFORM,
    glm::vec4 color = DEFAULT_LIGHT_COLOR) [static]
```

Definition at line 5 of file [Light.cpp](#).

References [m\\_currentLightId](#), and [ven::MAX\\_LIGHTS](#).

Referenced by [duplicate\(\)](#).

Here is the caller graph for this function:



#### 7.26.3.2 duplicate()

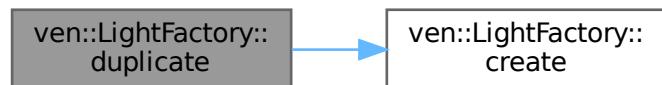
```
static std::unique_ptr< Light > ven::LightFactory::duplicate (
    const Light & cpyLight) [inline], [static]
```

Definition at line 35 of file [Light.hpp](#).

References [ven::Light::color](#), [create\(\)](#), and [ven::Light::transform](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.26.3.3 operator=() [1/2]

```
LightFactory & ven::LightFactory::operator= (
    const LightFactory & )  [delete]
```

### 7.26.3.4 operator=() [2/2]

```
LightFactory & ven::LightFactory::operator= (
    LightFactory && )  [delete]
```

## 7.26.4 Member Data Documentation

### 7.26.4.1 m\_currentLightId

unsigned int ven::LightFactory::m\_currentLightId = 0 [static], [private]

Definition at line 39 of file [Light.hpp](#).

Referenced by [create\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Light.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Light.cpp](#)

## 7.27 ven::LightPushConstantData Struct Reference

```
#include <PointLight.hpp>
```

Collaboration diagram for ven::LightPushConstantData:

ven::LightPushConstantData
+ position
+ color
+ radius

### Public Attributes

- `glm::vec4 position {}`
- `glm::vec4 color {}`
- `float radius`

### 7.27.1 Detailed Description

Definition at line 13 of file [PointLight.hpp](#).

### 7.27.2 Member Data Documentation

#### 7.27.2.1 color

```
glm::vec4 ven::LightPushConstantData::color {}
```

Definition at line 15 of file [PointLight.hpp](#).

#### 7.27.2.2 position

```
glm::vec4 ven::LightPushConstantData::position {}
```

Definition at line 14 of file [PointLight.hpp](#).

Referenced by [ven::PointLightRenderSystem::render\(\)](#).

#### 7.27.2.3 radius

```
float ven::LightPushConstantData::radius
```

Definition at line 16 of file [PointLight.hpp](#).

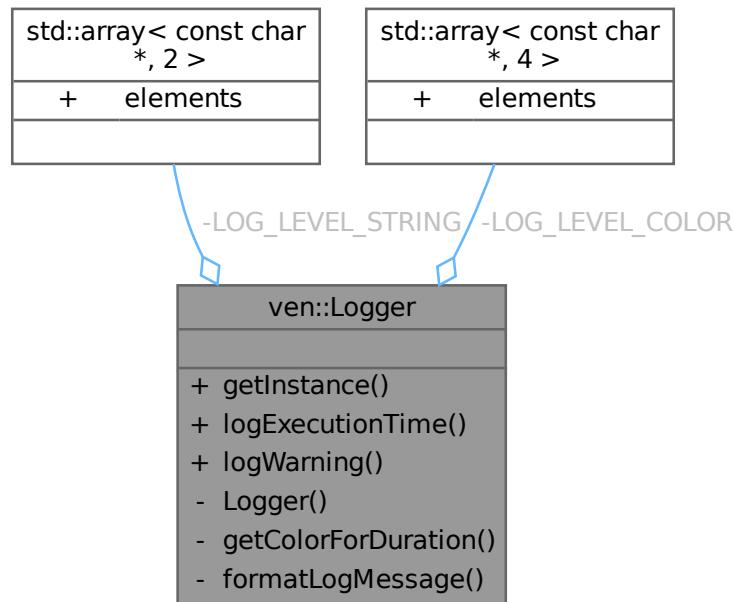
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[PointLight.hpp](#)

## 7.28 ven::Logger Class Reference

```
#include <Logger.hpp>
```

Collaboration diagram for ven::Logger:



### Static Public Member Functions

- static [Logger & getInstance \(\)](#)
- template<typename Func >  
  static void [logExecutionTime \(const std::string &message, Func &&func\)](#)
- static void [logWarning \(const std::string &message\)](#)

### Private Member Functions

- [Logger \(\)](#)

### Static Private Member Functions

- static const char \* [getColorForDuration \(const float duration\)](#)
- static std::string [formatLogMessage \(LogLevel level, const std::string &message\)](#)

### Static Private Attributes

- static constexpr std::array< const char \*, 2 > [LOG\\_LEVEL\\_STRING = {"INFO", "WARNING"}](#)
- static constexpr std::array< const char \*, 4 > [LOG\\_LEVEL\\_COLOR = {"\033\[31m", "\033\[32m", "\033\[33m", "\033\[0m\n"}](#)

## 7.28.1 Detailed Description

Definition at line 28 of file [Logger.hpp](#).

## 7.28.2 Constructor & Destructor Documentation

### 7.28.2.1 Logger()

```
ven::Logger::Logger () [private]
```

Definition at line 3 of file [logger.cpp](#).

## 7.28.3 Member Function Documentation

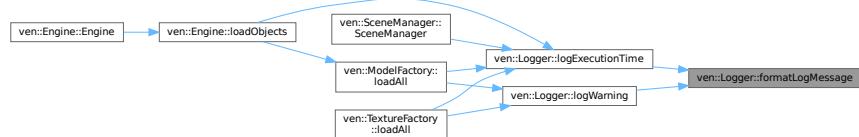
### 7.28.3.1 formatLogMessage()

```
std::string ven::Logger::formatLogMessage (
    LogLevel level,
    const std::string & message) [static], [nodiscard], [private]
```

Definition at line 14 of file [logger.cpp](#).

Referenced by [logExecutionTime\(\)](#), and [logWarning\(\)](#).

Here is the caller graph for this function:



### 7.28.3.2 getColorForDuration()

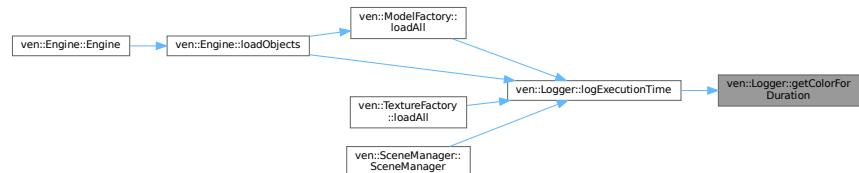
```
static const char * ven::Logger::getColorForDuration (
    const float duration) [inline], [static], [private]
```

Definition at line 54 of file [Logger.hpp](#).

References [LOG\\_LEVEL\\_COLOR](#).

Referenced by [logExecutionTime\(\)](#).

Here is the caller graph for this function:



### 7.28.3.3 getInstance()

```
static Logger & ven::Logger::getInstance () [inline], [static]
```

Definition at line 32 of file [Logger.hpp](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



### 7.28.3.4 logExecutionTime()

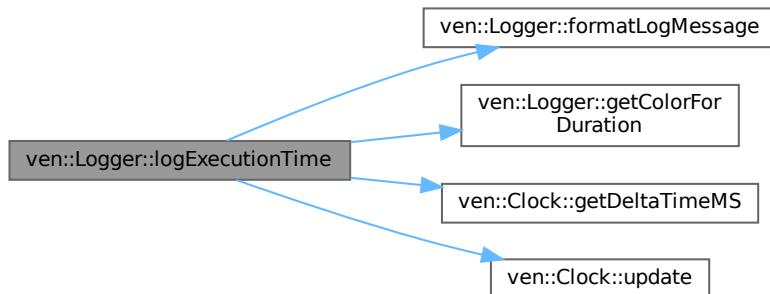
```
template<typename Func >
static void ven::Logger::logExecutionTime (
    const std::string & message,
    Func && func) [inline], [static]
```

Definition at line 35 of file [Logger.hpp](#).

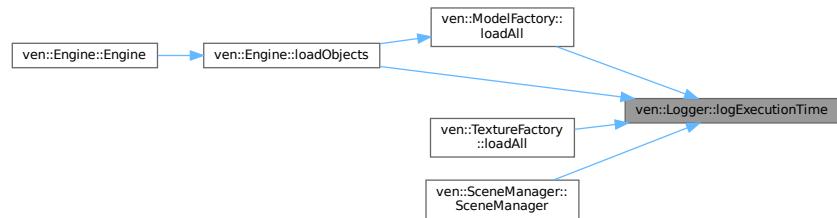
References [formatLogMessage\(\)](#), [getColorForDuration\(\)](#), [ven::Clock::getDeltaTimeMS\(\)](#), [ven::INFO](#), [LOG\\_LEVEL\\_COLOR](#), and [ven::Clock::update\(\)](#).

Referenced by [ven::ModelFactory::loadAll\(\)](#), [ven::TextureFactory::loadAll\(\)](#), [ven::Engine::loadObjects\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.28.3.5 logWarning()

```
static void ven::Logger::logWarning (
    const std::string & message) [inline], [static]
```

Definition at line 45 of file [Logger.hpp](#).

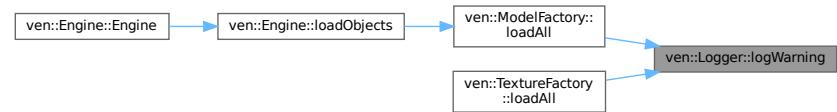
References [formatLogMessage\(\)](#), [LOG\\_LEVEL\\_COLOR](#), and [ven::WARNING](#).

Referenced by [ven::ModelFactory::loadAll\(\)](#), and [ven::TextureFactory::loadAll\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.28.4 Member Data Documentation

### 7.28.4.1 LOG\_LEVEL\_COLOR

```
std::array<const char*, 4> ven::Logger::LOG_LEVEL_COLOR = {"\033[31m", "\033[32m", "\033[33m",
"\033[0m\n"} [static], [constexpr], [private]
```

Definition at line 50 of file [Logger.hpp](#).

Referenced by [getColorForDuration\(\)](#), [logExecutionTime\(\)](#), and [logWarning\(\)](#).

#### 7.28.4.2 LOG\_LEVEL\_STRING

```
std::array<const char*, 2> ven::Logger::LOG_LEVEL_STRING = {"INFO", "WARNING"} [static],  
[constexpr], [private]
```

Definition at line 49 of file [Logger.hpp](#).

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

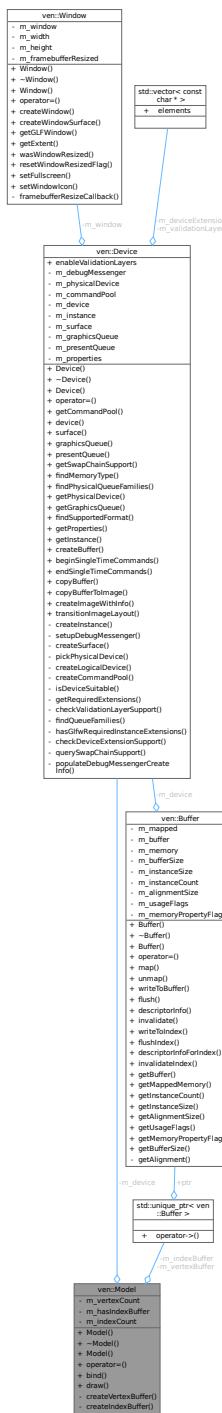
- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp
- /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp

## 7.29 ven::Model Class Reference

Class for model.

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model:



## Classes

- struct [Builder](#)
- struct [Vertex](#)

## Public Member Functions

- [Model \(Device &device, const Builder &builder\)](#)

- `~Model ()=default`
- `Model (const Model &)=delete`
- `void operator= (const Model &)=delete`
- `void bind (VkCommandBuffer commandBuffer) const`
- `void draw (VkCommandBuffer commandBuffer) const`

### Private Member Functions

- `void createVertexBuffer (const std::vector< Vertex > &vertices)`
- `void createIndexBuffer (const std::vector< uint32_t > &indices)`

### Private Attributes

- `Device & m_device`
- `std::unique_ptr< Buffer > m_vertexBuffer`
- `uint32_t m_vertexCount`
- `bool m_hasIndexBuffer {false}`
- `std::unique_ptr< Buffer > m_indexBuffer`
- `uint32_t m_indexCount`

## 7.29.1 Detailed Description

Class for model.

Definition at line 29 of file [Model.hpp](#).

## 7.29.2 Constructor & Destructor Documentation

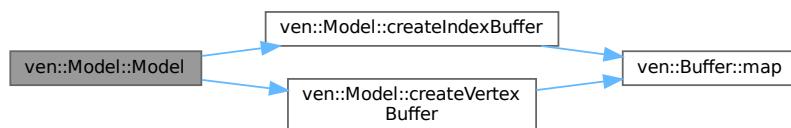
### 7.29.2.1 Model() [1/2]

```
ven::Model::Model (
    Device & device,
    const Builder & builder)
```

Definition at line 20 of file [model.cpp](#).

References [createIndexBuffer\(\)](#), [createVertexBuffer\(\)](#), [ven::Model::Builder::indices](#), and [ven::Model::Builder::vertices](#).

Here is the call graph for this function:



### 7.29.2.2 ~Model()

```
ven::Model::~Model () [default]
```

### 7.29.2.3 Model() [2/2]

```
ven::Model::Model (
    const Model & ) [delete]
```

## 7.29.3 Member Function Documentation

### 7.29.3.1 bind()

```
void ven::Model::bind (
    VkCommandBuffer commandBuffer) const
```

Definition at line 73 of file [model.cpp](#).

### 7.29.3.2 createIndexBuffer()

```
void ven::Model::createIndexBuffer (
    const std::vector< uint32_t > & indices) [private]
```

Definition at line 43 of file [model.cpp](#).

References [ven::Buffer::map\(\)](#).

Referenced by [Model\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.29.3.3 `createVertexBuffer()`

```
void ven::Model::createVertexBuffer (
    const std::vector< Vertex > & vertices) [private]
```

Definition at line 26 of file [model.cpp](#).

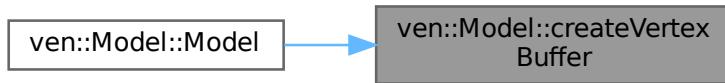
References [ven::Buffer::map\(\)](#).

Referenced by [Model\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.29.3.4 `draw()`

```
void ven::Model::draw (
    VkCommandBuffer commandBuffer) const
```

Definition at line 64 of file [model.cpp](#).

### 7.29.3.5 `operator=()`

```
void ven::Model::operator= (
    const Model &) [delete]
```

## 7.29.4 Member Data Documentation

### 7.29.4.1 m\_device

```
Device& ven::Model::m_device [private]
```

Definition at line 69 of file [Model.hpp](#).

### 7.29.4.2 m\_hasIndexBuffer

```
bool ven::Model::m_hasIndexBuffer {false} [private]
```

Definition at line 73 of file [Model.hpp](#).

### 7.29.4.3 m\_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

Definition at line 74 of file [Model.hpp](#).

### 7.29.4.4 m\_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

Definition at line 75 of file [Model.hpp](#).

### 7.29.4.5 m\_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

Definition at line 70 of file [Model.hpp](#).

### 7.29.4.6 m\_vertexCount

```
uint32_t ven::Model::m_vertexCount [private]
```

Definition at line 71 of file [Model.hpp](#).

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

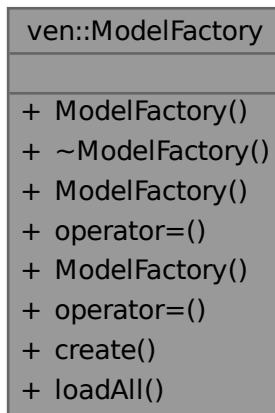
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

## 7.30 ven::ModelFactory Class Reference

Class for [Model](#) factory.

```
#include <Model.hpp>
```

Collaboration diagram for ven::ModelFactory:



### Public Member Functions

- [ModelFactory \(\)=delete](#)
- [~ModelFactory \(\)=default](#)
- [ModelFactory \(const ModelFactory &\)=delete](#)
- [ModelFactory & operator= \(const ModelFactory &\)=delete](#)
- [ModelFactory \(ModelFactory &&\)=delete](#)
- [ModelFactory & operator= \(ModelFactory &&\)=delete](#)

### Static Public Member Functions

- static std::unique\_ptr< [Model](#) > [create](#) ([Device](#) &device, const std::string &filepath)
- static std::unordered\_map< std::string, std::shared\_ptr< [Model](#) > > [loadAll](#) ([Device](#) &device, const std::string &folderPath)

### 7.30.1 Detailed Description

Class for [Model](#) factory.

Definition at line 20 of file [Model.hpp](#).

## 7.30.2 Constructor & Destructor Documentation

### 7.30.2.1 ModelFactory() [1/3]

```
ven::ModelFactory::ModelFactory () [delete]
```

### 7.30.2.2 ~ModelFactory()

```
ven::ModelFactory::~ModelFactory () [default]
```

### 7.30.2.3 ModelFactory() [2/3]

```
ven::ModelFactory::ModelFactory (
    const ModelFactory & ) [delete]
```

### 7.30.2.4 ModelFactory() [3/3]

```
ven::ModelFactory::ModelFactory (
    ModelFactory && ) [delete]
```

## 7.30.3 Member Function Documentation

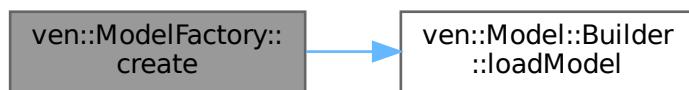
### 7.30.3.1 create()

```
std::unique_ptr< ven::Model > ven::ModelFactory::create (
    Device & device,
    const std::string & filepath) [static]
```

Definition at line 20 of file [Model.cpp](#).

References [ven::Model::Builder::loadModel\(\)](#).

Here is the call graph for this function:



### 7.30.3.2 `loadAll()`

```
std::unordered_map< std::string, std::shared_ptr< ven::Model > > ven::ModelFactory::loadAll (
    Device & device,
    const std::string & folderPath) [static]
```

Definition at line 3 of file [Model.cpp](#).

References [ven::Logger::logExecutionTime\(\)](#), and [ven::Logger::logWarning\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.30.3.3 `operator=()` [1/2]

```
ModelFactory & ven::ModelFactory::operator= (
    const ModelFactory & ) [delete]
```

### 7.30.3.4 `operator=()` [2/2]

```
ModelFactory & ven::ModelFactory::operator= (
    ModelFactory && ) [delete]
```

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Model.cpp](#)

## 7.31 ven::Object Class Reference

## Class for object.

```
#include <Object.hpp>
```

## Collaboration diagram for ven::Object:



## Public Types

- using `Map` = `std::unordered_map<unsigned int, Object>`

## Public Member Functions

- `Object (const unsigned int objId)`
- `~Object ()=default`
- `Object (const Object &)=delete`
- `Object & operator= (const Object &)=delete`
- `Object (Object &&)=default`
- `Object & operator= (Object &&)=default`
- `unsigned int getId () const`
- `std::string getName () const`
- `std::shared_ptr< Model > getModel () const`
- `std::shared_ptr< Texture > getDiffuseMap () const`
- `VkDescriptorBufferInfo getBufferInfo (const int frameIndex) const`
- `void setModel (const std::shared_ptr< Model > &model)`
- `void setDiffuseMap (const std::shared_ptr< Texture > &diffuseMap)`
- `void setName (const std::string &name)`
- `void setBufferInfo (const int frameIndex, const VkDescriptorBufferInfo &info)`

## Public Attributes

- `Transform3D transform {}`

## Private Attributes

- `unsigned int m_objId`
- `std::string m_name`
- `std::shared_ptr< Model > m_model = nullptr`
- `std::shared_ptr< Texture > m_diffuseMap = nullptr`
- `std::unordered_map< int, VkDescriptorBufferInfo > m_bufferInfo`

## 7.31.1 Detailed Description

Class for object.

Definition at line 24 of file [Object.hpp](#).

## 7.31.2 Member Typedef Documentation

### 7.31.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

Definition at line 28 of file [Object.hpp](#).

## 7.31.3 Constructor & Destructor Documentation

### 7.31.3.1 Object() [1/3]

```
ven::Object::Object (
    const unsigned int objId) [inline], [explicit]
```

Definition at line 30 of file [Object.hpp](#).

### 7.31.3.2 ~Object()

```
ven::Object::~Object () [default]
```

### 7.31.3.3 Object() [2/3]

```
ven::Object::Object (
    const Object & ) [delete]
```

### 7.31.3.4 Object() [3/3]

```
ven::Object::Object (
    Object && ) [default]
```

## 7.31.4 Member Function Documentation

### 7.31.4.1 getBufferInfo()

```
VkDescriptorBufferInfo ven::Object::getBufferInfo (
    const int frameIndex) const [inline], [nodiscard]
```

Definition at line 43 of file [Object.hpp](#).

References [m\\_bufferInfo](#).

### 7.31.4.2 getDiffuseMap()

```
std::shared_ptr< Texture > ven::Object::getDiffuseMap () const [inline], [nodiscard]
```

Definition at line 42 of file [Object.hpp](#).

References [m\\_diffuseMap](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



### 7.31.4.3 getId()

```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

Definition at line 39 of file [Object.hpp](#).

References [m\\_objId](#).

### 7.31.4.4 getModel()

```
std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]
```

Definition at line 41 of file [Object.hpp](#).

References [m\\_model](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



### 7.31.4.5 getName()

```
std::string ven::Object::getName () const [inline], [nodiscard]
```

Definition at line 40 of file [Object.hpp](#).

References [m\\_name](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



### 7.31.4.6 operator=() [1/2]

```
Object & ven::Object::operator= (
    const Object & ) [delete]
```

#### 7.31.4.7 operator=() [2/2]

```
Object & ven::Object::operator= (
    Object &&) [default]
```

#### 7.31.4.8 setBufferInfo()

```
void ven::Object::setBufferInfo (
    const int frameIndex,
    const VkDescriptorBufferInfo & info) [inline]
```

Definition at line 47 of file [Object.hpp](#).

References [m\\_bufferInfo](#).

#### 7.31.4.9 setDiffuseMap()

```
void ven::Object::setDiffuseMap (
    const std::shared_ptr< Texture > & diffuseMap) [inline]
```

Definition at line 45 of file [Object.hpp](#).

References [m\\_diffuseMap](#).

#### 7.31.4.10 setModel()

```
void ven::Object::setModel (
    const std::shared_ptr< Model > & model) [inline]
```

Definition at line 44 of file [Object.hpp](#).

References [m\\_model](#).

#### 7.31.4.11 setName()

```
void ven::Object::setName (
    const std::string & name) [inline]
```

Definition at line 46 of file [Object.hpp](#).

References [m\\_name](#).

### 7.31.5 Member Data Documentation

#### 7.31.5.1 m\_bufferInfo

```
std::unordered_map<int, VkDescriptorBufferInfo> ven::Object::m_bufferInfo [private]
```

Definition at line 57 of file [Object.hpp](#).

Referenced by [getBufferInfo\(\)](#), and [setBufferInfo\(\)](#).

### 7.31.5.2 m\_diffuseMap

```
std::shared_ptr<Texture> ven::Object::m_diffuseMap = nullptr [private]
```

Definition at line 56 of file [Object.hpp](#).

Referenced by [getDiffuseMap\(\)](#), and [setDiffuseMap\(\)](#).

### 7.31.5.3 m\_model

```
std::shared_ptr<Model> ven::Object::m_model = nullptr [private]
```

Definition at line 55 of file [Object.hpp](#).

Referenced by [getModel\(\)](#), and [setModel\(\)](#).

### 7.31.5.4 m\_name

```
std::string ven::Object::m_name [private]
```

Definition at line 54 of file [Object.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

### 7.31.5.5 m\_objId

```
unsigned int ven::Object::m_objId [private]
```

Definition at line 53 of file [Object.hpp](#).

Referenced by [getId\(\)](#).

### 7.31.5.6 transform

```
Transform3D ven::Object::transform {}
```

Definition at line 49 of file [Object.hpp](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

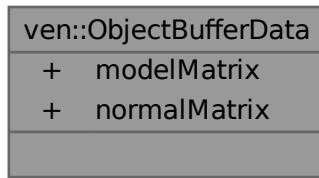
The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/[Object.hpp](#)

## 7.32 ven::ObjectBufferData Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::ObjectBufferData:



### Public Attributes

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

### 7.32.1 Detailed Description

Definition at line 29 of file [FrameInfo.hpp](#).

### 7.32.2 Member Data Documentation

#### 7.32.2.1 modelMatrix

```
glm::mat4 ven::ObjectBufferData::modelMatrix {1.F}
```

Definition at line 30 of file [FrameInfo.hpp](#).

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

#### 7.32.2.2 normalMatrix

```
glm::mat4 ven::ObjectBufferData::normalMatrix {1.F}
```

Definition at line 31 of file [FrameInfo.hpp](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp](#)

## 7.33 ven::ObjectFactory Class Reference

Class for object factory.

```
#include <Object.hpp>
```

Collaboration diagram for ven::ObjectFactory:

ven::ObjectFactory
- m_currentObjId
+ ObjectFactory()
+ ~ObjectFactory()
+ ObjectFactory()
+ operator=(())
+ ObjectFactory()
+ operator=(())
+ create()
+ duplicate()

### Public Member Functions

- [ObjectFactory \(\)=delete](#)
- [~ObjectFactory \(\)=default](#)
- [ObjectFactory \(const ObjectFactory &\)=delete](#)
- [ObjectFactory & operator= \(const ObjectFactory &\)=delete](#)
- [ObjectFactory \(ObjectFactory &&\)=delete](#)
- [ObjectFactory & operator= \(ObjectFactory &&\)=delete](#)

### Static Public Member Functions

- static std::unique\_ptr<[Object](#)> [create](#) (const std::shared\_ptr<[Texture](#)> &texture, const std::shared\_ptr<[Model](#)> &model, const std::string &name, const [Transform3D](#) &transform)
- static std::unique\_ptr<[Object](#)> [duplicate](#) (const [Object](#) &objSrc)

### Static Private Attributes

- static unsigned int [m\\_currentObjId](#) = 0

#### 7.33.1 Detailed Description

Class for object factory.

Definition at line 18 of file [Object.hpp](#).

## 7.33.2 Constructor & Destructor Documentation

### 7.33.2.1 ObjectFactory() [1/3]

```
ven::ObjectFactory::ObjectFactory () [delete]
```

### 7.33.2.2 ~ObjectFactory()

```
ven::ObjectFactory::~ObjectFactory () [default]
```

### 7.33.2.3 ObjectFactory() [2/3]

```
ven::ObjectFactory::ObjectFactory (
    const ObjectFactory & ) [delete]
```

### 7.33.2.4 ObjectFactory() [3/3]

```
ven::ObjectFactory::ObjectFactory (
    ObjectFactory && ) [delete]
```

## 7.33.3 Member Function Documentation

### 7.33.3.1 create()

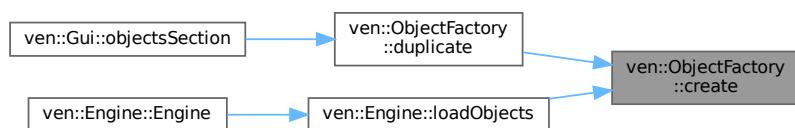
```
std::unique_ptr< ven::Object > ven::ObjectFactory::create (
    const std::shared_ptr< Texture > & texture,
    const std::shared_ptr< Model > & model,
    const std::string & name,
    const Transform3D & transform) [static]
```

Definition at line 5 of file [Object.cpp](#).

References [m\\_currentObjId](#), and [ven::MAX\\_OBJECTS](#).

Referenced by [duplicate\(\)](#), and [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



### 7.33.3.2 `duplicate()`

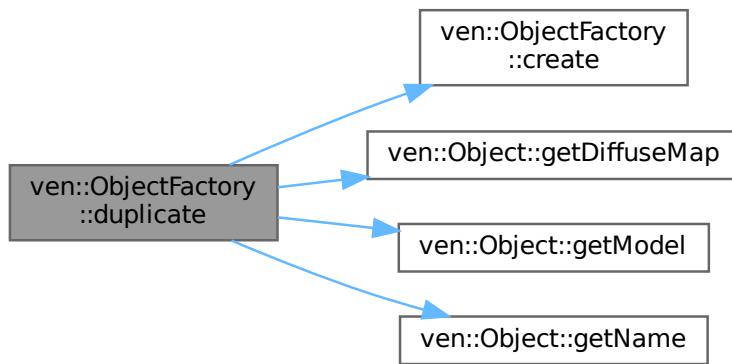
```
static std::unique_ptr< Object > ven::ObjectFactory::duplicate (
    const Object & objSrc) [inline], [static]
```

Definition at line 31 of file [Object.hpp](#).

References [create\(\)](#), [ven::Object::getDiffuseMap\(\)](#), [ven::Object::getModel\(\)](#), [ven::Object::getName\(\)](#), and [ven::Object::transform](#).

Referenced by [ven::Gui::objectsSection\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.33.3.3 `operator=()` [1/2]

```
ObjectFactory & ven::ObjectFactory::operator= (
    const ObjectFactory & ) [delete]
```

### 7.33.3.4 operator=() [2/2]

```
ObjectFactory & ven::ObjectFactory::operator= (
    ObjectFactory && ) [delete]
```

## 7.33.4 Member Data Documentation

### 7.33.4.1 m\_currentObjId

```
unsigned int ven::ObjectFactory::m_currentObjId = 0 [static], [private]
```

Definition at line 35 of file [Object.hpp](#).

Referenced by [create\(\)](#).

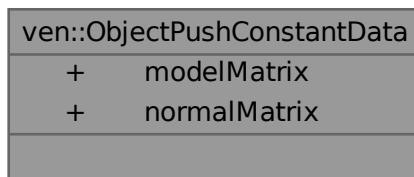
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Object.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Object.cpp](#)

## 7.34 ven::ObjectPushConstantData Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::ObjectPushConstantData:



### Public Attributes

- `glm::mat4 modelMatrix {}`
- `glm::mat4 normalMatrix {}`

### 7.34.1 Detailed Description

Definition at line 13 of file [Object.hpp](#).

## 7.34.2 Member Data Documentation

### 7.34.2.1 modelMatrix

```
glm::mat4 ven::ObjectPushConstantData::modelMatrix {}
```

Definition at line 14 of file [Object.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#).

### 7.34.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

Definition at line 15 of file [Object.hpp](#).

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

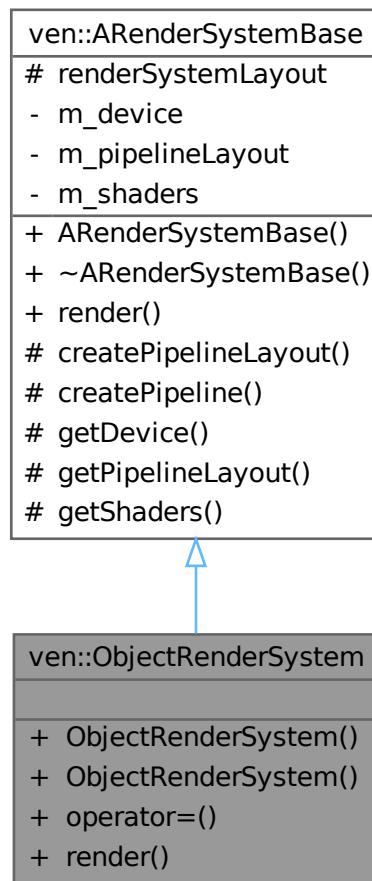
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[Object.hpp](#)

## 7.35 ven::ObjectRenderSystem Class Reference

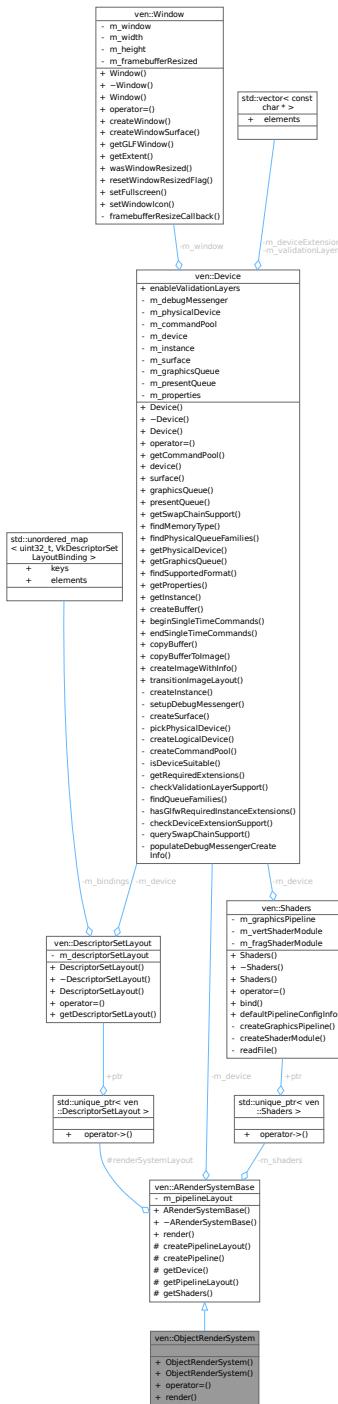
Class for object render system.

```
#include <Object.hpp>
```

Inheritance diagram for ven::ObjectRenderSystem:



Collaboration diagram for ven::ObjectRenderSystem:



## Public Member Functions

- `ObjectRenderSystem (Device &device, const VkRenderPass renderPass, const VkDescriptorSetLayout globalsetLayout)`
- `ObjectRenderSystem (const ObjectRenderSystem &)=delete`
- `ObjectRenderSystem & operator= (const ObjectRenderSystem &)=delete`
- `void render (const FrameInfo &frameInfo) const override`

## Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase \(Device &device\)](#)
- virtual [~ARenderSystemBase \(\)](#)

### Additional Inherited Members

## Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout \(VkDescriptorSetLayout globalsetLayout, uint32\\_t pushConstantSize\)](#)
- void [createPipeline \(VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight\)](#)
- [Device & getDevice \(\) const](#)
- [VkPipelineLayout getPipelineLayout \(\) const](#)
- const std::unique\_ptr<[Shaders](#)> & [getShaders \(\) const](#)

## Protected Attributes inherited from [ven::ARenderSystemBase](#)

- std::unique\_ptr<[DescriptorSetLayout](#)> [renderSystemLayout](#)

### 7.35.1 Detailed Description

Class for object render system.

Definition at line 23 of file [Object.hpp](#).

### 7.35.2 Constructor & Destructor Documentation

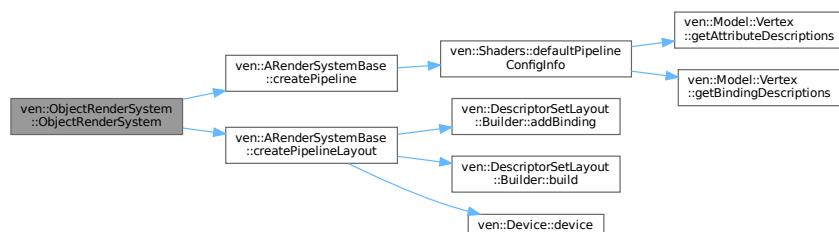
#### 7.35.2.1 ObjectRenderSystem() [1/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalsetLayout) [inline], [explicit]
```

Definition at line 27 of file [Object.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS\\_BIN\\_PATH](#).

Here is the call graph for this function:



### 7.35.2.2 ObjectRenderSystem() [2/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    const ObjectRenderSystem & ) [delete]
```

## 7.35.3 Member Function Documentation

### 7.35.3.1 operator=(\*)

```
ObjectRenderSystem & ven::ObjectRenderSystem::operator= (
    const ObjectRenderSystem & ) [delete]
```

### 7.35.3.2 render()

```
void ven::ObjectRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

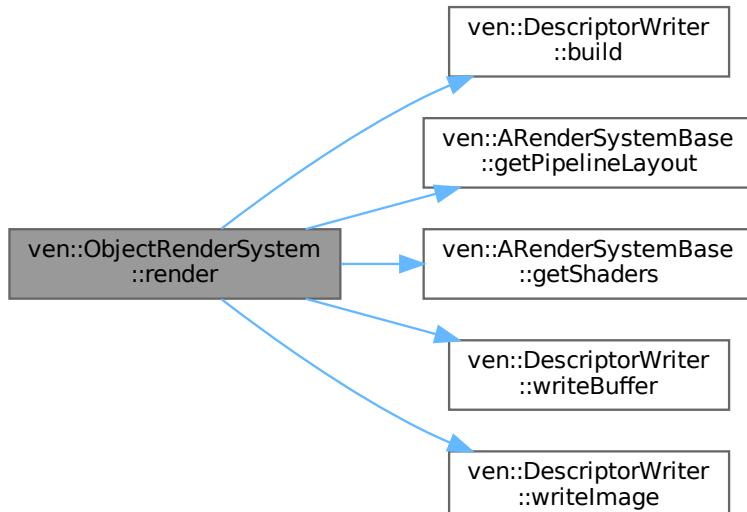
Implements [ven::ARenderSystemBase](#).

Definition at line 6 of file [object.cpp](#).

References [ven::DescriptorWriter::build\(\)](#), [ven::FrameInfo::commandBuffer](#), [ven::FrameInfo::frameDescriptorPool](#), [ven::FrameInfo::frameIndex](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders\(\)](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::ObjectPushConstantData::modelMatrix](#), [ven::FrameInfo::objects](#), [ven::ARenderSystemBase::renderSystemLayout](#), [ven::DescriptorWriter::writeBuffer\(\)](#), and [ven::DescriptorWriter::writeImage\(\)](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

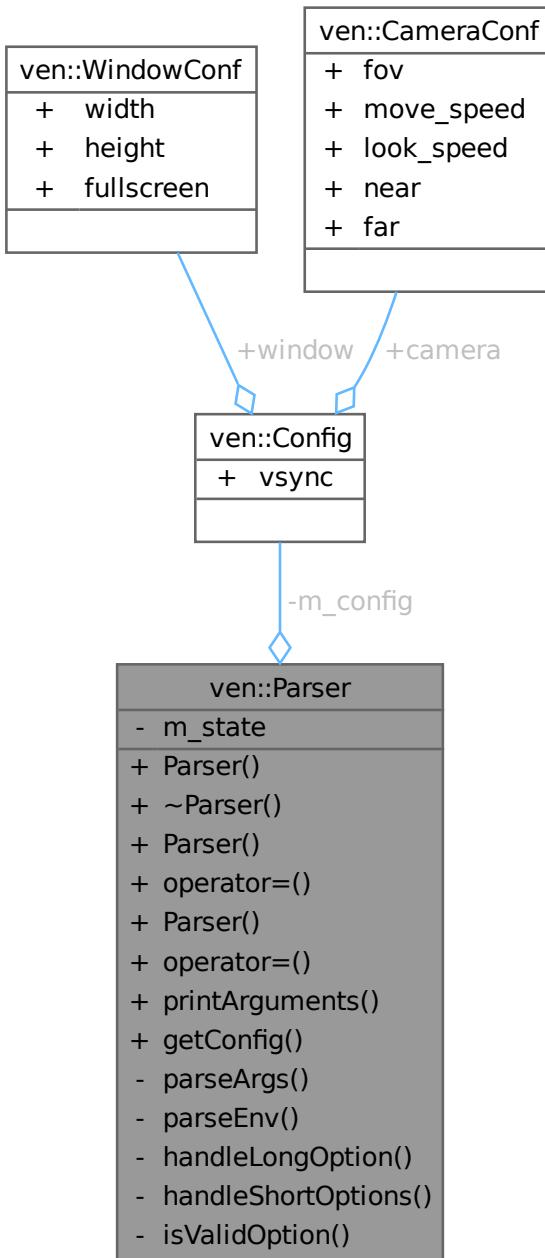
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[Object.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/[object.cpp](#)

## 7.36 ven::Parser Class Reference

Class for [Parser](#).

```
#include <Parser.hpp>
```

Collaboration diagram for ven::Parser:



### Public Member Functions

- `Parser (int argc, char *argv[], char *envp[])`
- `~Parser ()=default`
- `Parser (const Parser &)=delete`
- `Parser & operator=(const Parser &)=delete`
- `Parser (Parser &&)=delete`

- `Parser & operator= (Parser &&) = delete`
- `void printArguments () const`
- `Config getConfig () const`

### Private Member Functions

- `void parseArgs (const std::vector< std::string_view > &argv)`
- `void parseEnv (const std::unordered_map< std::string, std::string > &envp)`
- `void handleLongOption (const std::string_view &arg, const std::vector< std::string_view > &argv, size_t &index)`
- `void handleShortOptions (const std::string_view &arg, const std::vector< std::string_view > &argv, size_t &index)`

### Static Private Member Functions

- `static bool isValidOption (const std::string_view &option)`

### Private Attributes

- `Config m_config`
- `bool m_state = true`

## 7.36.1 Detailed Description

Class for [Parser](#).

Definition at line 134 of file [Parser.hpp](#).

## 7.36.2 Constructor & Destructor Documentation

### 7.36.2.1 Parser() [1/3]

```
ven::Parser::Parser (
    int argc,
    char * argv[ ],
    char * envp[ ])
```

Definition at line 55 of file [parser.cpp](#).

### 7.36.2.2 ~Parser()

```
ven::Parser::~Parser () [default]
```

### 7.36.2.3 Parser() [2/3]

```
ven::Parser::Parser (
    const Parser & ) [delete]
```

#### 7.36.2.4 Parser() [3/3]

```
ven::Parser::Parser (
    Parser && )  [delete]
```

### 7.36.3 Member Function Documentation

#### 7.36.3.1 getConfig()

```
Config ven::Parser::getConfig () const  [inline], [nodiscard]
```

Definition at line 148 of file [Parser.hpp](#).

References [m\\_config](#).

#### 7.36.3.2 handleLongOption()

```
void ven::Parser::handleLongOption (
    const std::string_view & arg,
    const std::vector< std::string_view > & argv,
    size_t & index)  [private]
```

Definition at line 6 of file [parser.cpp](#).

References [ven::FUNCTION\\_MAP\\_OPT\\_LONG](#), [isValidOption\(\)](#), and [m\\_config](#).

Here is the call graph for this function:



#### 7.36.3.3 handleShortOptions()

```
void ven::Parser::handleShortOptions (
    const std::string_view & arg,
    const std::vector< std::string_view > & argv,
    size_t & index)  [private]
```

Definition at line 15 of file [parser.cpp](#).

References [ven::FUNCTION\\_MAP\\_OPT\\_SHORT](#).

#### 7.36.3.4 isValidOption()

```
static bool ven::Parser::isValidOption (
    const std::string_view & option) [inline], [static], [nodiscard], [private]
```

Definition at line 161 of file [Parser.hpp](#).

References [ven::FUNCTION\\_MAP\\_OPT\\_LONG](#), and [ven::FUNCTION\\_MAP\\_OPT\\_SHORT](#).

Referenced by [handleLongOption\(\)](#).

Here is the caller graph for this function:



#### 7.36.3.5 operator=() [1/2]

```
Parser & ven::Parser::operator= (
    const Parser & ) [delete]
```

#### 7.36.3.6 operator=() [2/2]

```
Parser & ven::Parser::operator= (
    Parser && ) [delete]
```

#### 7.36.3.7 parseArgs()

```
void ven::Parser::parseArgs (
    const std::vector< std::string_view > & argv) [private]
```

Definition at line 39 of file [parser.cpp](#).

#### 7.36.3.8 parseEnv()

```
void ven::Parser::parseEnv (
    const std::unordered_map< std::string, std::string > & envp) [private]
```

Definition at line 27 of file [parser.cpp](#).

#### 7.36.3.9 printArguments()

```
void ven::Parser::printArguments () const
```

## 7.36.4 Member Data Documentation

### 7.36.4.1 m\_config

`Config` `ven::Parser::m_config` [private]

Definition at line 152 of file [Parser.hpp](#).

Referenced by [getConfig\(\)](#), and [handleLongOption\(\)](#).

### 7.36.4.2 m\_state

`bool` `ven::Parser::m_state = true` [private]

Definition at line 153 of file [Parser.hpp](#).

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

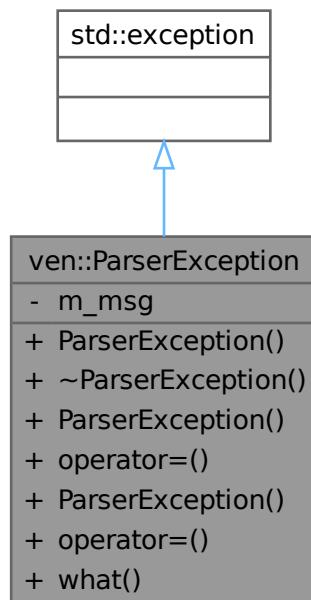
- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Parser.hpp
- /home/runner/work/VEngine/VEngine/src/Utils/parser.cpp

## 7.37 ven::ParserException Class Reference

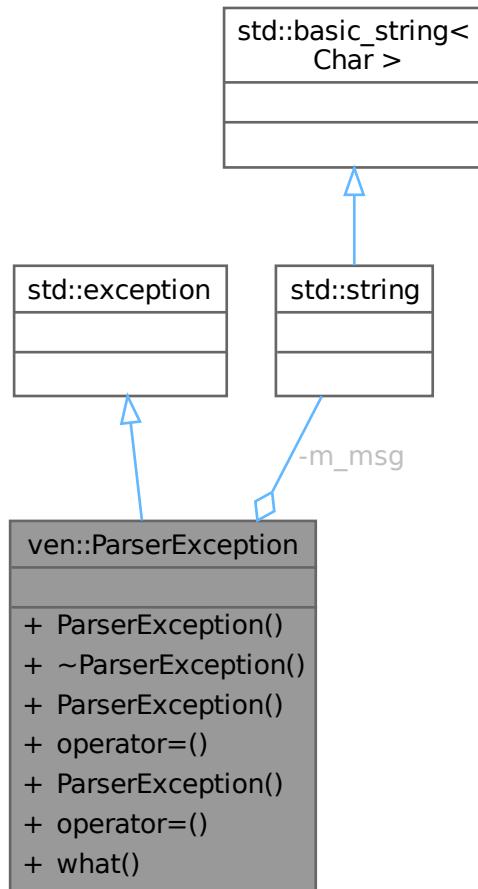
Custom exception class for parsing errors.

```
#include <Parser.hpp>
```

Inheritance diagram for ven::ParserException:



Collaboration diagram for ven::ParserException:



### Public Member Functions

- `ParserException (std::string msg)`
- `~ParserException () override=default`
- `ParserException (const ParserException &)=delete`
- `ParserException & operator=(const ParserException &)=delete`
- `ParserException (ParserException &&)=delete`
- `ParserException & operator=(ParserException &&)=delete`
- `const char * what () const noexcept override`

### Private Attributes

- `std::string m_msg {0}`

### 7.37.1 Detailed Description

Custom exception class for parsing errors.

Definition at line 24 of file [Parser.hpp](#).

## 7.37.2 Constructor & Destructor Documentation

### 7.37.2.1 ParserException() [1/3]

```
ven::ParserException::ParserException (
    std::string msg) [inline], [explicit]
```

Definition at line 28 of file [Parser.hpp](#).

### 7.37.2.2 ~ParserException()

```
ven::ParserException::~ParserException () [override], [default]
```

### 7.37.2.3 ParserException() [2/3]

```
ven::ParserException::ParserException (
    const ParserException &) [delete]
```

### 7.37.2.4 ParserException() [3/3]

```
ven::ParserException::ParserException (
    ParserException &&) [delete]
```

## 7.37.3 Member Function Documentation

### 7.37.3.1 operator=(()) [1/2]

```
ParserException & ven::ParserException::operator= (
    const ParserException &) [delete]
```

### 7.37.3.2 operator=(()) [2/2]

```
ParserException & ven::ParserException::operator= (
    ParserException &&) [delete]
```

### 7.37.3.3 what()

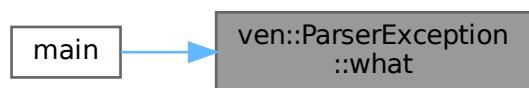
```
const char * ven::ParserException::what () const [inline], [nodiscard], [override], [noexcept]
```

Definition at line 36 of file [Parser.hpp](#).

References [m\\_msg](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



### 7.37.4 Member Data Documentation

#### 7.37.4.1 m\_msg

```
std::string ven::ParserException::m_msg {0} [private]
```

Definition at line 40 of file [Parser.hpp](#).

Referenced by [what\(\)](#).

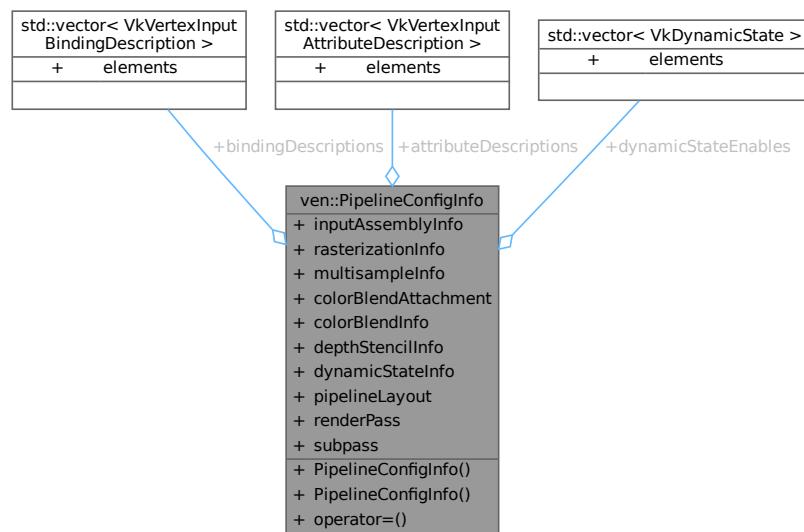
The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/[Parser.hpp](#)

## 7.38 ven::PipelineConfigInfo Struct Reference

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::PipelineConfigInfo:



### Public Member Functions

- [PipelineConfigInfo \(\)=default](#)
- [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

### 7.38.1 Detailed Description

Definition at line 15 of file [Shaders.hpp](#).

### 7.38.2 Constructor & Destructor Documentation

#### 7.38.2.1 PipelineConfigInfo() [1/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo () [default]
```

#### 7.38.2.2 PipelineConfigInfo() [2/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo (
    const PipelineConfigInfo & ) [delete]
```

### 7.38.3 Member Function Documentation

#### 7.38.3.1 operator=()

```
PipelineConfigInfo & ven::PipelineConfigInfo::operator= (
    const PipelineConfigInfo & ) [delete]
```

### 7.38.4 Member Data Documentation

#### 7.38.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

Definition at line 21 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

Definition at line 20 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

Definition at line 25 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

Definition at line 26 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

Definition at line 27 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

Definition at line 28 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

Definition at line 29 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.8 **inputAssemblyInfo**

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

Definition at line 22 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.9 **multisampleInfo**

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

Definition at line 24 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.10 **pipelineLayout**

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

Definition at line 30 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

#### 7.38.4.11 **rasterizationInfo**

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

Definition at line 23 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 7.38.4.12 **renderPass**

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

Definition at line 31 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

#### 7.38.4.13 **subpass**

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 32 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Shaders.hpp](#)

## 7.39 ven::PointLightData Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::PointLightData:



### Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [shininess](#) {32.F}
- float [padding](#) [3]

### 7.39.1 Detailed Description

Definition at line [21](#) of file [FrameInfo.hpp](#).

### 7.39.2 Member Data Documentation

#### 7.39.2.1 color

```
glm::vec4 ven::PointLightData::color {}
```

Definition at line [24](#) of file [FrameInfo.hpp](#).

#### 7.39.2.2 padding

```
float ven::PointLightData::padding[3]
```

Definition at line [26](#) of file [FrameInfo.hpp](#).

### 7.39.2.3 position

```
glm::vec4 ven::PointLightData::position {}
```

Definition at line 23 of file [FrameInfo.hpp](#).

### 7.39.2.4 shininess

```
float ven::PointLightData::shininess {32.F}
```

Definition at line 25 of file [FrameInfo.hpp](#).

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

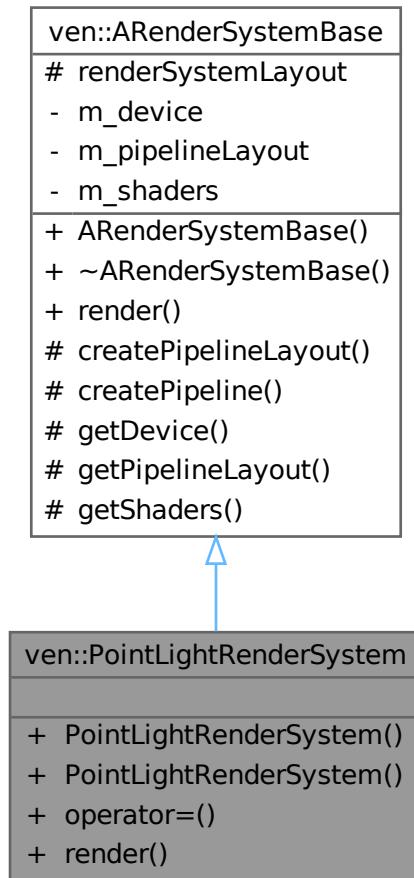
- [/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp](#)

## 7.40 ven::PointLightRenderSystem Class Reference

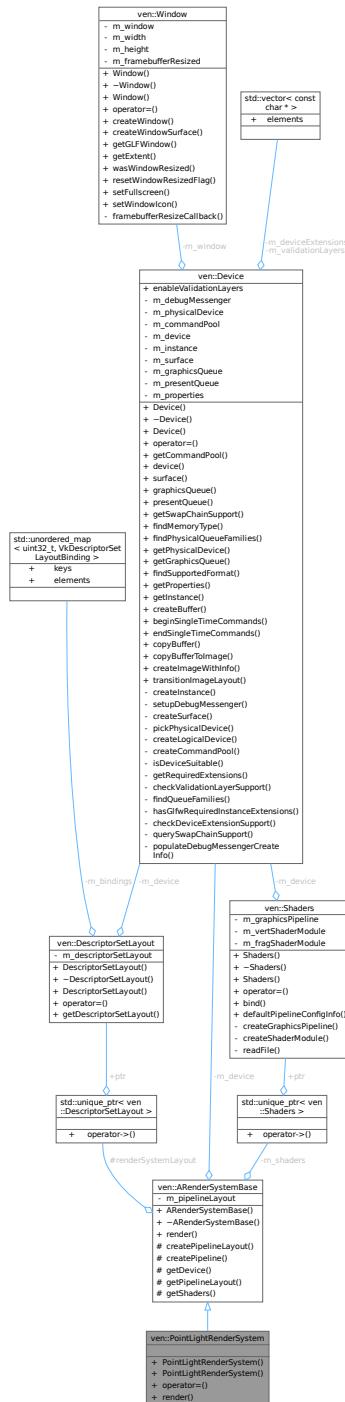
Class for point light system.

```
#include <PointLight.hpp>
```

Inheritance diagram for ven::PointLightRenderSystem:



Collaboration diagram for ven::PointLightRenderSystem:



## Public Member Functions

- [PointLightRenderSystem \(Device &device, const VkRenderPass renderPass, const VkDescriptorSetLayout globalsetLayout\)](#)
- [PointLightRenderSystem \(const PointLightRenderSystem &\)=delete](#)
- [PointLightRenderSystem & operator= \(const PointLightRenderSystem &\)=delete](#)
- void [render \(const FrameInfo &frameInfo\) const override](#)

## Public Member Functions inherited from `ven::ARenderSystemBase`

- `ARenderSystemBase (Device &device)`
- virtual `~ARenderSystemBase ()`

### Additional Inherited Members

## Protected Member Functions inherited from `ven::ARenderSystemBase`

- void `createPipelineLayout (VkDescriptorSetLayout globalsetLayout, uint32_t pushConstantSize)`
- void `createPipeline (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)`
- `Device & getDevice () const`
- `VkPipelineLayout getPipelineLayout () const`
- `const std::unique_ptr< Shaders > & getShaders () const`

## Protected Attributes inherited from `ven::ARenderSystemBase`

- `std::unique_ptr< DescriptorSetLayout > renderSystemLayout`

### 7.40.1 Detailed Description

Class for point light system.

Definition at line 24 of file `PointLight.hpp`.

### 7.40.2 Constructor & Destructor Documentation

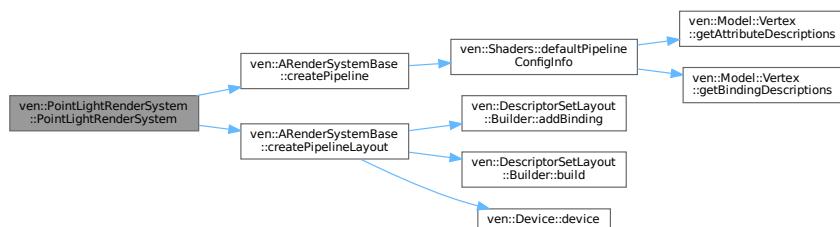
#### 7.40.2.1 `PointLightRenderSystem() [1/2]`

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalsetLayout) [inline], [explicit]
```

Definition at line 28 of file `PointLight.hpp`.

References `ven::ARenderSystemBase::createPipeline()`, `ven::ARenderSystemBase::createPipelineLayout()`, and `ven::SHADERS_BIN_PATH`.

Here is the call graph for this function:



### 7.40.2.2 PointLightRenderSystem() [2/2]

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    const PointLightRenderSystem & ) [delete]
```

## 7.40.3 Member Function Documentation

### 7.40.3.1 operator=()

```
PointLightRenderSystem & ven::PointLightRenderSystem::operator= (
    const PointLightRenderSystem & ) [delete]
```

### 7.40.3.2 render()

```
void ven::PointLightRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

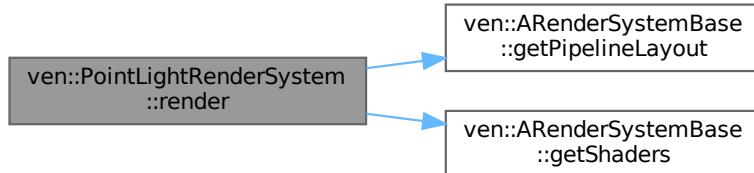
Implements [ven::ARenderSystemBase](#).

Definition at line 5 of file [pointLight.cpp](#).

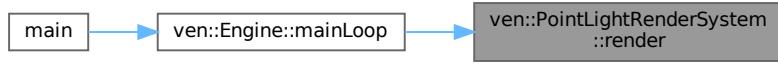
References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders\(\)](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::FrameInfo::lights](#), and [ven::LightPushConstantData::position](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



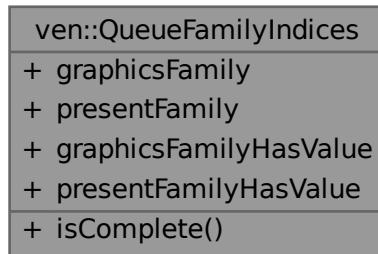
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[PointLight.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/[pointLight.cpp](#)

## 7.41 ven::QueueFamilyIndices Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::QueueFamilyIndices:



### Public Member Functions

- bool [isComplete \(\) const](#)

### Public Attributes

- uint32\_t [graphicsFamily](#) {}
- uint32\_t [presentFamily](#) {}
- bool [graphicsFamilyHasValue](#) = false
- bool [presentFamilyHasValue](#) = false

### 7.41.1 Detailed Description

Definition at line [22](#) of file [Device.hpp](#).

### 7.41.2 Member Function Documentation

#### 7.41.2.1 [isComplete\(\)](#)

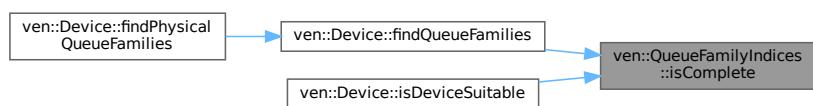
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

Definition at line [27](#) of file [Device.hpp](#).

References [graphicsFamilyHasValue](#), and [presentFamilyHasValue](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [ven::Device::isDeviceSuitable\(\)](#).

Here is the caller graph for this function:



### 7.41.3 Member Data Documentation

#### 7.41.3.1 `graphicsFamily`

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

Definition at line 23 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

#### 7.41.3.2 `graphicsFamilyHasValue`

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 25 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

#### 7.41.3.3 `presentFamily`

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

Definition at line 24 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

#### 7.41.3.4 `presentFamilyHasValue`

```
bool ven::QueueFamilyIndices::presentFamilyHasValue = false
```

Definition at line 26 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

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

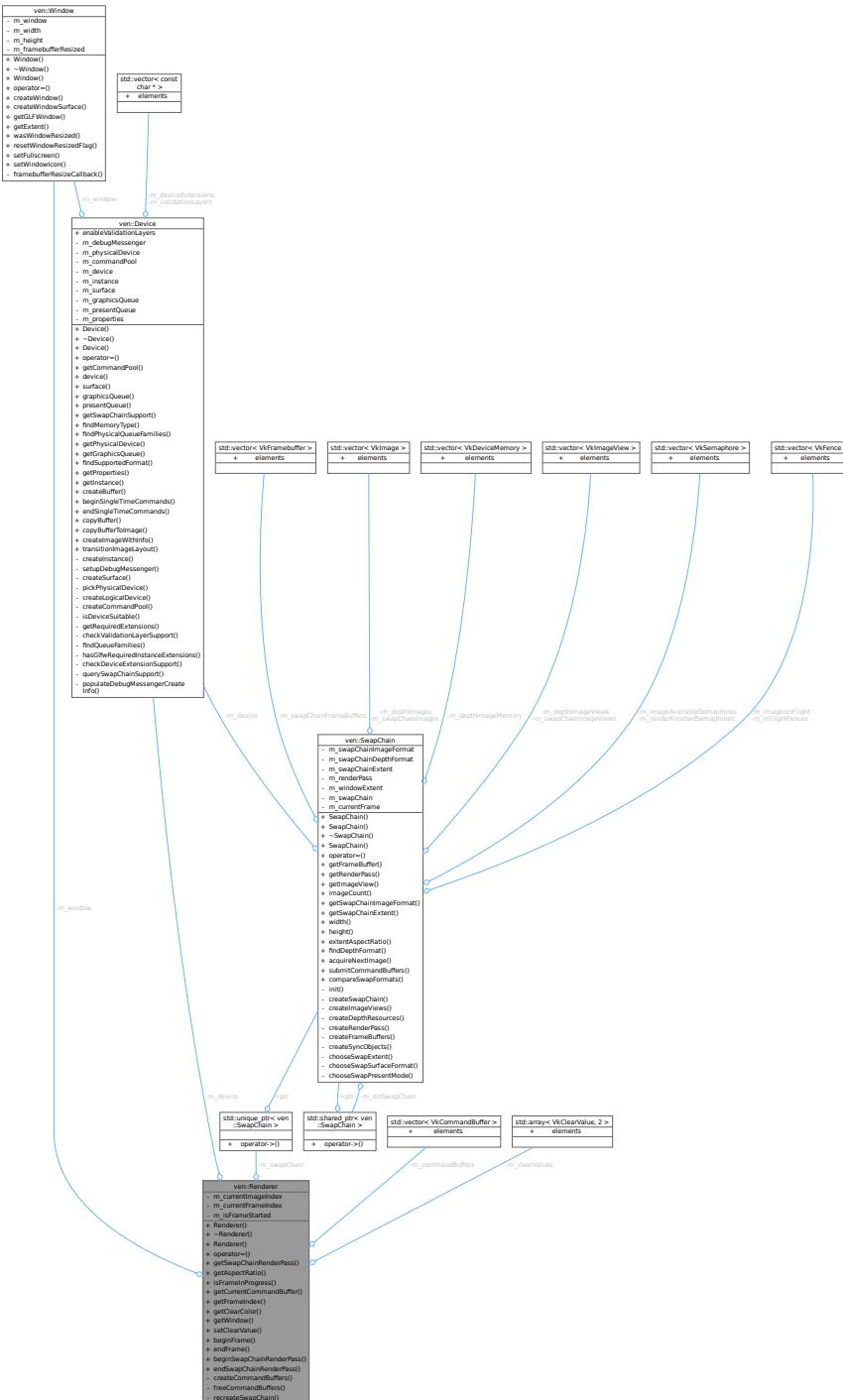
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Device.hpp](#)

## 7.42 ven::Renderer Class Reference

Class for renderer.

```
#include <Renderer.hpp>
```

Collaboration diagram for ven::Renderer:



## Public Member Functions

- `Renderer (Window &window, Device &device)`
- `~Renderer ()`
- `Renderer (const Renderer &)=delete`
- `Renderer & operator= (const Renderer &)=delete`
- `VkRenderPass getSwapChainRenderPass () const`
- `float getAspectRatio () const`
- `bool isFrameInProgress () const`
- `VkCommandBuffer getCurrentCommandBuffer () const`
- `unsigned long getFrameIndex () const`
- `std::array< float, 4 > getClearColor () const`
- `Window & getWindow () const`
- `void setClearColor (const VkClearColorValue clearColorValue=DEFAULT_CLEAR_COLOR, const VkClearDepthStencilValue clearDepthValue=DEFAULT_CLEAR_DEPTH)`
- `VkCommandBuffer beginFrame ()`
- `void endFrame ()`
- `void beginSwapChainRenderPass (VkCommandBuffer commandBuffer) const`
- `void endSwapChainRenderPass (VkCommandBuffer commandBuffer) const`

## Private Member Functions

- `void createCommandBuffers ()`
- `void freeCommandBuffers ()`
- `void recreateSwapChain ()`

## Private Attributes

- `Window & m_window`
- `Device & m_device`
- `std::unique_ptr< SwapChain > m_swapChain`
- `std::vector< VkCommandBuffer > m_commandBuffers`
- `std::array< VkClearColorValue, 2 > m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F}`
- `uint32_t m_currentImageIndex {0}`
- `unsigned long m_currentFrameIndex {0}`
- `bool m_isFrameStarted {false}`

### 7.42.1 Detailed Description

Class for renderer.

Definition at line 24 of file [Renderer.hpp](#).

## 7.42.2 Constructor & Destructor Documentation

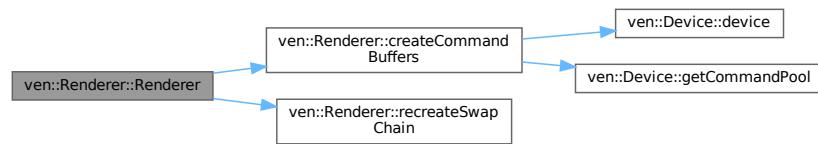
### 7.42.2.1 Renderer() [1/2]

```
ven::Renderer::Renderer (
    Window & window,
    Device & device) [inline]
```

Definition at line 28 of file [Renderer.hpp](#).

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



### 7.42.2.2 ~Renderer()

```
ven::Renderer::~Renderer () [inline]
```

Definition at line 29 of file [Renderer.hpp](#).

References [freeCommandBuffers\(\)](#).

Here is the call graph for this function:



### 7.42.2.3 Renderer() [2/2]

```
ven::Renderer::Renderer (
    const Renderer & ) [delete]
```

### 7.42.3 Member Function Documentation

#### 7.42.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

Definition at line 43 of file [renderer.cpp](#).

#### 7.42.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

Definition at line 89 of file [renderer.cpp](#).

#### 7.42.3.3 createCommandBuffers()

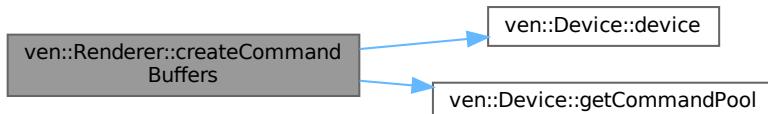
```
void ven::Renderer::createCommandBuffers () [private]
```

Definition at line 3 of file [renderer.cpp](#).

References [ven::Device::device\(\)](#), [ven::Device::getCommandPool\(\)](#), [m\\_commandBuffers](#), [m\\_device](#), and [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

Referenced by [Renderer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.42.3.4 `endFrame()`

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

#### 7.42.3.5 `endSwapChainRenderPass()`

```
void ven::Renderer::endSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

Definition at line 119 of file [renderer.cpp](#).

#### 7.42.3.6 `freeCommandBuffers()`

```
void ven::Renderer::freeCommandBuffers () [private]
```

Definition at line 17 of file [renderer.cpp](#).

Referenced by [~Renderer\(\)](#).

Here is the caller graph for this function:



#### 7.42.3.7 `getAspectRatio()`

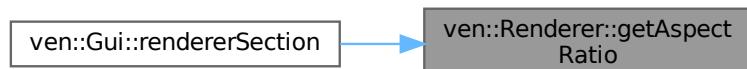
```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

Definition at line 35 of file [Renderer.hpp](#).

References [m\\_swapChain](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



#### 7.42.3.8 getClearColor()

```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

Definition at line 40 of file [Renderer.hpp](#).

References [m\\_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



#### 7.42.3.9 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

Definition at line 37 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), [m\\_commandBuffers](#), and [m\\_currentFrameIndex](#).

Referenced by [ven::Gui::render\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.42.3.10 `getFrameIndex()`

```
unsigned long ven::Renderer::getFrameIndex () const [inline], [nodiscard]
```

Definition at line 39 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), and [m\\_currentFrameIndex](#).

Here is the call graph for this function:



#### 7.42.3.11 `getSwapChainRenderPass()`

```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

Definition at line 34 of file [Renderer.hpp](#).

References [m\\_swapChain](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



#### 7.42.3.12 `getWindow()`

```
Window & ven::Renderer::getWindow () const [inline], [nodiscard]
```

Definition at line 47 of file [Renderer.hpp](#).

References [m\\_window](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



### 7.42.3.13 isFrameInProgress()

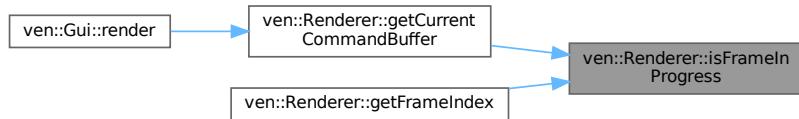
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

Definition at line 36 of file [Renderer.hpp](#).

References [m\\_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



### 7.42.3.14 operator=(\*)

```
Renderer & ven::Renderer::operator= (
    const Renderer & ) [delete]
```

### 7.42.3.15 recreateSwapChain()

```
void ven::Renderer::recreateSwapChain () [private]
```

Definition at line 23 of file [renderer.cpp](#).

Referenced by [Renderer\(\)](#).

Here is the caller graph for this function:



### 7.42.3.16 setClearValue()

```
void ven::Renderer::setClearValue (
    const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

Definition at line 49 of file [Renderer.hpp](#).

References [m\\_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



## 7.42.4 Member Data Documentation

### 7.42.4.1 m\_clearValues

```
std::array<VkClearColorValue, 2> ven::Renderer::m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F} [private]
```

Definition at line 65 of file [Renderer.hpp](#).

Referenced by [getClearColor\(\)](#), and [setClearValue\(\)](#).

### 7.42.4.2 m\_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

Definition at line 64 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

### 7.42.4.3 m\_currentFrameIndex

```
unsigned long ven::Renderer::m_currentFrameIndex {0} [private]
```

Definition at line 68 of file [Renderer.hpp](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

#### 7.42.4.4 m\_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

Definition at line 67 of file [Renderer.hpp](#).

#### 7.42.4.5 m\_device

```
Device& ven::Renderer::m_device [private]
```

Definition at line 62 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#).

#### 7.42.4.6 m\_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

Definition at line 69 of file [Renderer.hpp](#).

Referenced by [isFrameInProgress\(\)](#).

#### 7.42.4.7 m\_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]
```

Definition at line 63 of file [Renderer.hpp](#).

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

#### 7.42.4.8 m\_window

```
Window& ven::Renderer::m_window [private]
```

Definition at line 61 of file [Renderer.hpp](#).

Referenced by [getWindow\(\)](#).

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

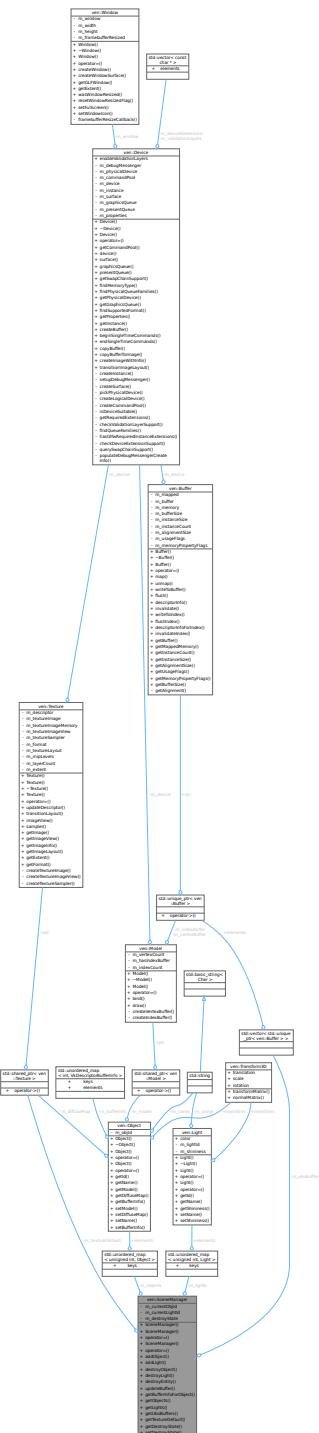
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Renderer.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[renderer.cpp](#)

## 7.43 ven::SceneManager Class Reference

Class for object manager.

```
#include <Manager.hpp>
```

Collaboration diagram for ven::SceneManager:



## Public Member Functions

- `SceneManager (Device &device)`
- `SceneManager (const SceneManager &)=delete`
- `SceneManager & operator= (const SceneManager &)=delete`
- `SceneManager (SceneManager &&)=delete`
- `SceneManager & operator= (SceneManager &&)=delete`
- `void addObject (const std::unique_ptr< Object > &object)`
- `void addLight (const std::unique_ptr< Light > &light)`
- `void destroyObject (const unsigned int objectId)`
- `void destroyLight (const unsigned int lightId)`
- `void destroyEntity (std::vector< unsigned int > *objectIds, std::vector< unsigned int > *lightIds)`
- `void updateBuffer (GlobalUbo &ubo, unsigned long frameIndex, float frameTime)`
- `VkDescriptorBufferInfo getBufferInfoForObject (const int frameIndex, const unsigned int objectId) const`
- `Object::Map & getObjects ()`
- `Light::Map & getLights ()`
- `std::vector< std::unique_ptr< Buffer > > & getUboBuffers ()`
- `std::shared_ptr< Texture > getTextureDefault ()`
- `bool getDestroyState () const`
- `void setDestroyState (const bool state)`

## Private Attributes

- `unsigned int m_currentObjId {0}`
- `unsigned int m_currentLightId {0}`
- `std::shared_ptr< Texture > m_textureDefault`
- `Object::Map m_objects`
- `Light::Map m_lights`
- `std::vector< std::unique_ptr< Buffer > > m_uboBuffers {MAX_FRAMES_IN_FLIGHT}`
- `bool m_destroyState {false}`

### 7.43.1 Detailed Description

Class for object manager.

Definition at line 19 of file [Manager.hpp](#).

### 7.43.2 Constructor & Destructor Documentation

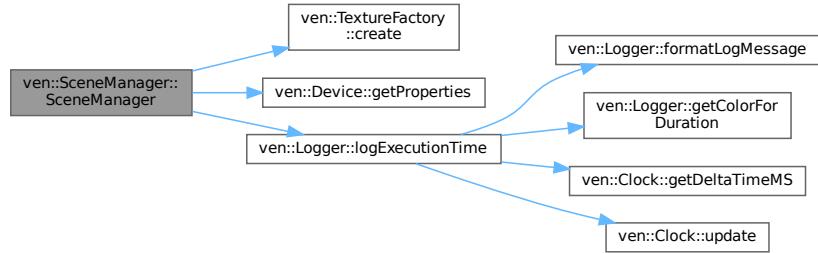
#### 7.43.2.1 SceneManager() [1/3]

```
ven::SceneManager::SceneManager (
    Device & device) [explicit]
```

Definition at line 7 of file [manager.cpp](#).

References `ven::TextureFactory::create()`, `ven::Device::getProperties()`, `ven::Logger::logExecutionTime()`, `m_textureDefault`, `m_uboBuffers`, and `ven::MAX_OBJECTS`.

Here is the call graph for this function:



#### 7.43.2.2 SceneManager() [2/3]

```
ven::SceneManager::SceneManager (
    const SceneManager & ) [delete]
```

#### 7.43.2.3 SceneManager() [3/3]

```
ven::SceneManager::SceneManager (
    SceneManager && ) [delete]
```

### 7.43.3 Member Function Documentation

#### 7.43.3.1 addLight()

```
void ven::SceneManager::addLight (
    const std::unique_ptr< Light > & light) [inline]
```

Definition at line 31 of file [Manager.hpp](#).

References [m\\_lights](#).

#### 7.43.3.2 addObject()

```
void ven::SceneManager::addObject (
    const std::unique_ptr< Object > & object) [inline]
```

Definition at line 30 of file [Manager.hpp](#).

References [m\\_objects](#).

### 7.43.3.3 `destroyEntity()`

```
void ven::SceneManager::destroyEntity (
    std::vector< unsigned int > * objectsIds,
    std::vector< unsigned int > * lightsIds)
```

Definition at line 54 of file [manager.cpp](#).

### 7.43.3.4 `destroyLight()`

```
void ven::SceneManager::destroyLight (
    const unsigned int lightId) [inline]
```

Definition at line 34 of file [Manager.hpp](#).

References [m\\_lights](#).

### 7.43.3.5 `destroyObject()`

```
void ven::SceneManager::destroyObject (
    const unsigned int objectId) [inline]
```

Definition at line 33 of file [Manager.hpp](#).

References [m\\_objects](#).

### 7.43.3.6 `getBufferInfoForObject()`

```
VkDescriptorBufferInfo ven::SceneManager::getBufferInfoForObject (
    const int frameIndex,
    const unsigned int objectId) const [inline], [nodiscard]
```

Definition at line 39 of file [Manager.hpp](#).

References [m\\_uboBuffers](#).

### 7.43.3.7 `getDestroyState()`

```
bool ven::SceneManager::getDestroyState () const [inline], [nodiscard]
```

Definition at line 44 of file [Manager.hpp](#).

References [m\\_destroyState](#).

#### 7.43.3.8 getLights()

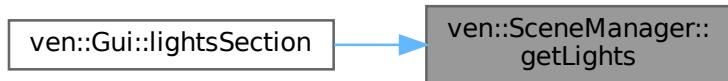
```
Light::Map & ven::SceneManager::getLights () [inline]
```

Definition at line 41 of file [Manager.hpp](#).

References [m\\_lights](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

Here is the caller graph for this function:



#### 7.43.3.9 getObjects()

```
Object::Map & ven::SceneManager::getObjects () [inline]
```

Definition at line 40 of file [Manager.hpp](#).

References [m\\_objects](#).

Referenced by [ven::Gui::objectsSection\(\)](#).

Here is the caller graph for this function:



#### 7.43.3.10 getTextureDefault()

```
std::shared_ptr< Texture > ven::SceneManager::getTextureDefault () [inline]
```

Definition at line 43 of file [Manager.hpp](#).

References [m\\_textureDefault](#).

**7.43.3.11 getUboBuffers()**

```
std::vector< std::unique_ptr< Buffer > > & ven::SceneManager::getUboBuffers () [inline]
```

Definition at line 42 of file [Manager.hpp](#).

References [m\\_uboBuffers](#).

**7.43.3.12 operator=(1/2)**

```
SceneManager & ven::SceneManager::operator= (
    const SceneManager & ) [delete]
```

**7.43.3.13 operator=(2/2)**

```
SceneManager & ven::SceneManager::operator= (
    SceneManager && ) [delete]
```

**7.43.3.14 setDestroyState()**

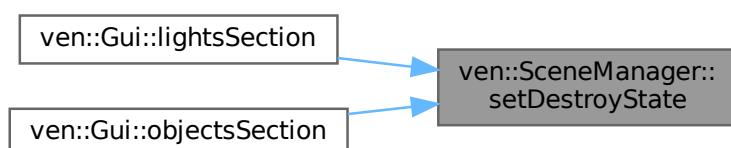
```
void ven::SceneManager::setDestroyState (
    const bool state) [inline]
```

Definition at line 46 of file [Manager.hpp](#).

References [m\\_destroyState](#).

Referenced by [ven::Gui::lightsSection\(\)](#), and [ven::Gui::objectsSection\(\)](#).

Here is the caller graph for this function:

**7.43.3.15 updateBuffer()**

```
void ven::SceneManager::updateBuffer (
    GlobalUbo & ubo,
    unsigned long frameIndex,
    float frameTime)
```

Definition at line 29 of file [manager.cpp](#).

References [ven::ObjectBufferData::modelMatrix](#), [ven::GlobalUbo::numLights](#), and [ven::GlobalUbo::pointLights](#).

## 7.43.4 Member Data Documentation

### 7.43.4.1 m\_currentLightId

```
unsigned int ven::SceneManager::m_currentLightId {0} [private]
```

Definition at line 51 of file [Manager.hpp](#).

### 7.43.4.2 m\_currentObjId

```
unsigned int ven::SceneManager::m_currentObjId {0} [private]
```

Definition at line 50 of file [Manager.hpp](#).

### 7.43.4.3 m\_destroyState

```
bool ven::SceneManager::m_destroyState {false} [private]
```

Definition at line 56 of file [Manager.hpp](#).

Referenced by [getDestroyState\(\)](#), and [setDestroyState\(\)](#).

### 7.43.4.4 m\_lights

```
Light::Map ven::SceneManager::m_lights [private]
```

Definition at line 54 of file [Manager.hpp](#).

Referenced by [addLight\(\)](#), [destroyLight\(\)](#), and [getLights\(\)](#).

### 7.43.4.5 m\_objects

```
Object::Map ven::SceneManager::m_objects [private]
```

Definition at line 53 of file [Manager.hpp](#).

Referenced by  [addObject\(\)](#),  [destroyObject\(\)](#), and  [getObjects\(\)](#).

### 7.43.4.6 m\_textureDefault

```
std::shared_ptr<Texture> ven::SceneManager::m_textureDefault [private]
```

Definition at line 52 of file [Manager.hpp](#).

Referenced by [getTextureDefault\(\)](#), and [SceneManager\(\)](#).

#### 7.43.4.7 m\_uboBuffers

```
std::vector<std::unique_ptr<Buffer>> ven::SceneManager::m_uboBuffers {MAX_FRAMES_IN_FLIGHT}  
[private]
```

Definition at line 55 of file [Manager.hpp](#).

Referenced by [getBufferInfoForObject\(\)](#), [getUboBuffers\(\)](#), and [SceneManager\(\)](#).

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

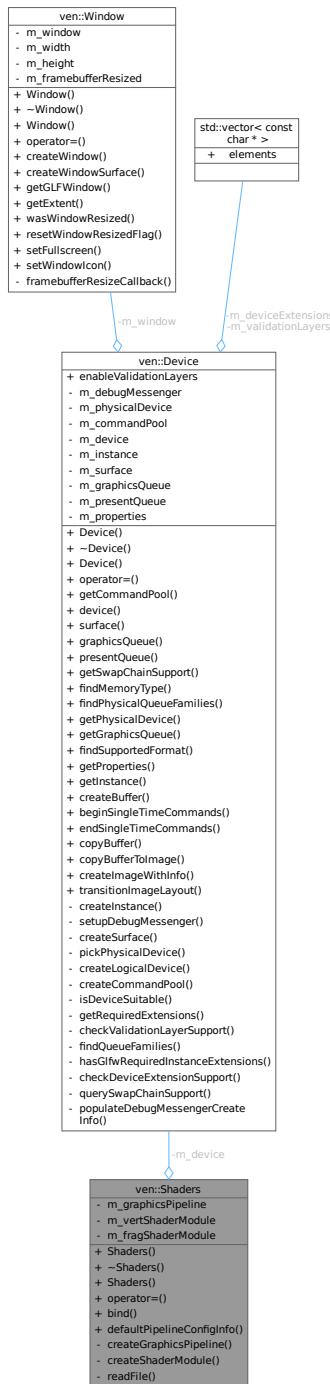
- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/[Manager.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Scene/[manager.cpp](#)

## 7.44 ven::Shaders Class Reference

Class for shaders.

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::Shaders:



## Public Member Functions

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

### Static Public Member Functions

- static void [defaultPipelineConfigInfo \(PipelineConfigInfo &configInfo\)](#)

### Private Member Functions

- void [createGraphicsPipeline \(const std::string &vertFilepath, const std::string &fragFilepath, const PipelineConfigInfo &configInfo\)](#)
- void [createShaderModule \(const std::vector< char > &code, VkShaderModule \\*shaderModule\) const](#)

### Static Private Member Functions

- static std::vector< char > [readFile \(const std::string &filename\)](#)

### Private Attributes

- [Device & m\\_device](#)
- [VkPipeline m\\_graphicsPipeline {nullptr}](#)
- [VkShaderModule m\\_vertShaderModule {nullptr}](#)
- [VkShaderModule m\\_fragShaderModule {nullptr}](#)

## 7.44.1 Detailed Description

Class for shaders.

Definition at line [40](#) of file [Shaders.hpp](#).

## 7.44.2 Constructor & Destructor Documentation

### 7.44.2.1 Shaders() [1/2]

```
ven::Shaders::Shaders (
    Device & device,
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [inline]
```

Definition at line [44](#) of file [Shaders.hpp](#).

References [createGraphicsPipeline\(\)](#).

Here is the call graph for this function:



#### 7.44.2.2 ~Shaders()

```
ven::Shaders::~Shaders ()
```

Definition at line 7 of file [shaders.cpp](#).

References [ven::Device::device\(\)](#), [m\\_device](#), [m\\_fragShaderModule](#), [m\\_graphicsPipeline](#), and [m\\_vertShaderModule](#).

Here is the call graph for this function:



#### 7.44.2.3 Shaders() [2/2]

```
ven::Shaders::Shaders (
    const Shaders & ) [delete]
```

### 7.44.3 Member Function Documentation

#### 7.44.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

Definition at line 51 of file [Shaders.hpp](#).

References [m\\_graphicsPipeline](#).

#### 7.44.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [private]
```

Definition at line 27 of file [shaders.cpp](#).

References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorAttachmentCount](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), [ven::PipelineConfigInfo::pipelineLayout](#), [ven::PipelineConfigInfo::rasterizationInfo](#), [ven::PipelineConfigInfo::renderPass](#), and [ven::PipelineConfigInfo::subpass](#).

Referenced by [Shaders\(\)](#).

Here is the caller graph for this function:



#### 7.44.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

Definition at line 96 of file [shaders.cpp](#).

#### 7.44.3.4 defaultPipelineConfigInfo()

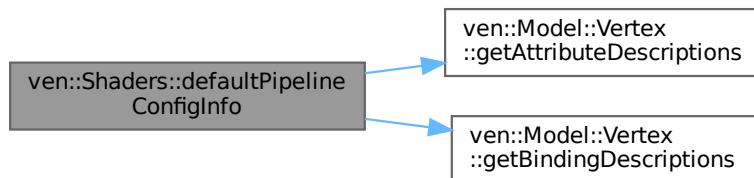
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 108 of file [shaders.cpp](#).

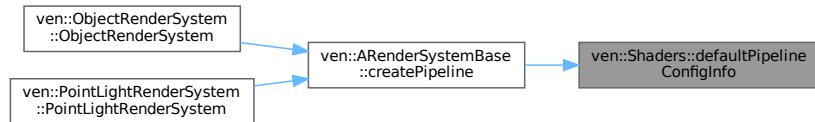
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions\(\)](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::ARenderSystemBase::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.44.3.5 operator=()

```
Shaders & ven::Shaders::operator= (
    const Shaders & ) [delete]
```

#### 7.44.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

Definition at line 14 of file [shaders.cpp](#).

### 7.44.4 Member Data Documentation

#### 7.44.4.1 m\_device

```
Device& ven::Shaders::m_device [private]
```

Definition at line 59 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

#### 7.44.4.2 m\_fragShaderModule

```
VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]
```

Definition at line 62 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

#### 7.44.4.3 m\_graphicsPipeline

```
VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]
```

Definition at line 60 of file [Shaders.hpp](#).

Referenced by [bind\(\)](#), and [~Shaders\(\)](#).

#### 7.44.4.4 m\_vertShaderModule

```
VkShaderModule ven::Shaders::m_vertShaderModule {nullptr} [private]
```

Definition at line 61 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

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

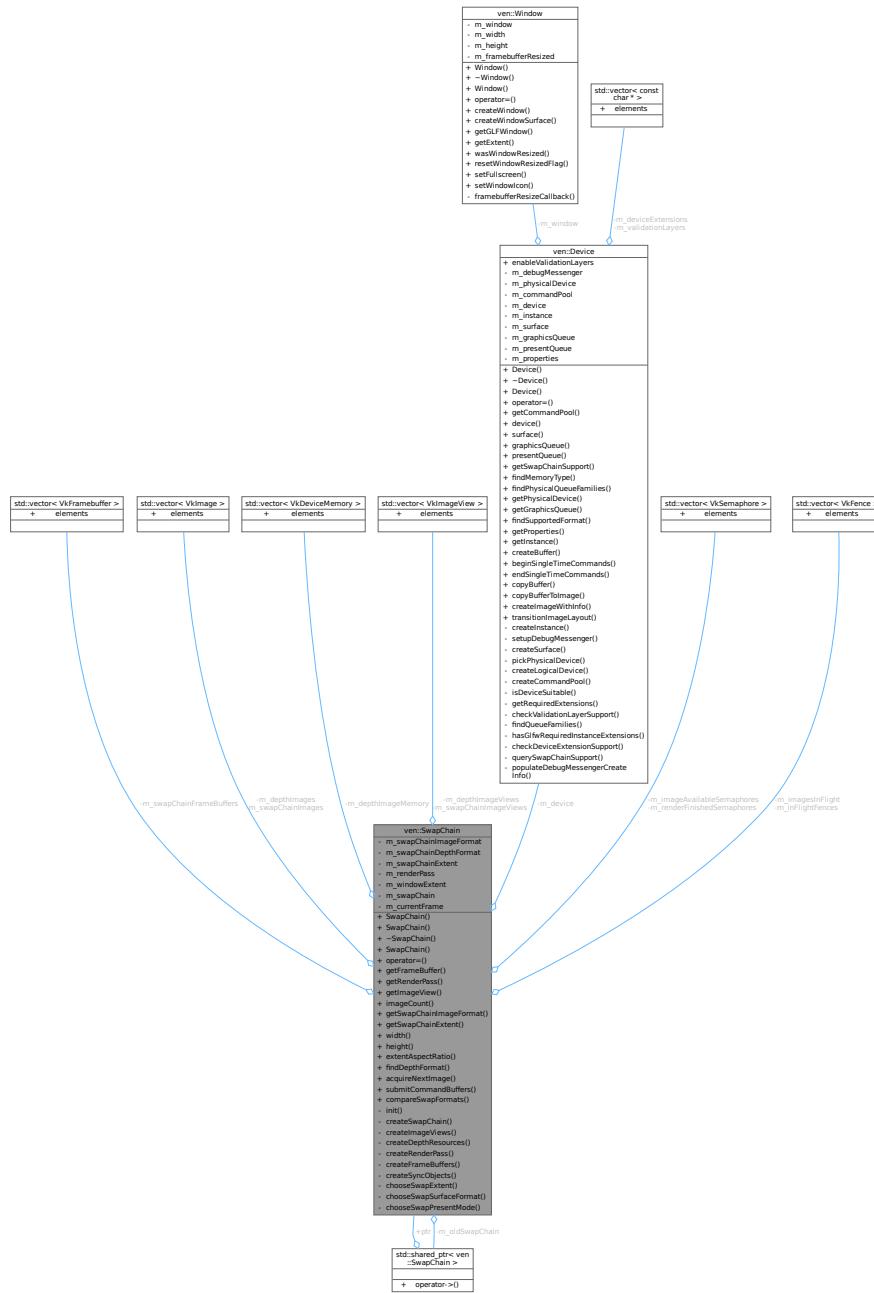
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Shaders.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[shaders.cpp](#)

## 7.45 ven::SwapChain Class Reference

Class for swap chain.

```
#include <SwapChain.hpp>
```

## Collaboration diagram for ven::SwapChain:



## Public Member Functions

- `SwapChain (Device &deviceRef, const VkExtent2D windowExtentRef)`
  - `SwapChain (Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr< SwapChain > previous)`
  - `~SwapChain ()`
  - `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 &swapChain) const`

### Private Member Functions

- `void init ()`
- `void createSwapChain ()`
- `void createImageViews ()`
- `void createDepthResources ()`
- `void createRenderPass ()`
- `void createFrameBuffers ()`
- `void createSyncObjects ()`
- `VkExtent2D chooseSwapExtent (const VkSurfaceCapabilitiesKHR &capabilities) const`

### Static Private Member Functions

- `static VkSurfaceFormatKHR chooseSwapSurfaceFormat (const std::vector< VkSurfaceFormatKHR > &availableFormats)`
- `static VkPresentModeKHR chooseSwapPresentMode (const std::vector< VkPresentModeKHR > &availablePresentModes)`

### Private Attributes

- `VkFormat m_swapChainImageFormat {}`
- `VkFormat m_swapChainDepthFormat {}`
- `VkExtent2D m_swapChainExtent {}`
- `std::vector< VkFramebuffer > m_swapChainFrameBuffers`
- `VkRenderPass m_renderPass {}`
- `std::vector< VkImage > m_depthImages`
- `std::vector< VkDeviceMemory > m_depthImageMemory`
- `std::vector< VkImageView > m_depthImageViews`
- `std::vector< VkImage > m_swapChainImages`
- `std::vector< VkImageView > m_swapChainImageViews`
- `Device & m_device`
- `VkExtent2D m_windowExtent`
- `VkSwapchainKHR m_swapChain {}`
- `std::shared_ptr< SwapChain > m_oldSwapChain`
- `std::vector< VkSemaphore > m_imageAvailableSemaphores`
- `std::vector< VkSemaphore > m_renderFinishedSemaphores`
- `std::vector< VkFence > m_inFlightFences`
- `std::vector< VkFence > m_imagesInFlight`
- `size_t m_currentFrame {0}`

### 7.45.1 Detailed Description

Class for swap chain.

Definition at line 22 of file [SwapChain.hpp](#).

### 7.45.2 Constructor & Destructor Documentation

#### 7.45.2.1 SwapChain() [1/3]

```
ven::SwapChain::SwapChain (
    Device & deviceRef,
    const VkExtent2D windowExtentRef) [inline]
```

Definition at line 26 of file [SwapChain.hpp](#).

References [init\(\)](#).

Here is the call graph for this function:



#### 7.45.2.2 SwapChain() [2/3]

```
ven::SwapChain::SwapChain (
    Device & deviceRef,
    const VkExtent2D windowExtentRef,
    std::shared_ptr< SwapChain > previous) [inline]
```

Definition at line 27 of file [SwapChain.hpp](#).

References [init\(\)](#), and [m\\_oldSwapChain](#).

Here is the call graph for this function:



### 7.45.2.3 ~SwapChain()

```
ven::SwapChain::~SwapChain ()
```

Definition at line 7 of file [swapChain.cpp](#).

References [ven::Device::device\(\)](#), [m\\_depthImageMemory](#), [m\\_depthImages](#), [m\\_depthImageViews](#), [m\\_device](#), [m\\_imageAvailableSemaphores](#), [m\\_inFlightFences](#), [m\\_renderFinishedSemaphores](#), [m\\_renderPass](#), [m\\_swapChain](#), [m\\_swapChainFrameBuffers](#), [m\\_swapChainImageViews](#), and [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

Here is the call graph for this function:



### 7.45.2.4 SwapChain() [3/3]

```
ven::SwapChain::SwapChain (
    const SwapChain & ) [delete]
```

## 7.45.3 Member Function Documentation

### 7.45.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

### 7.45.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

### 7.45.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

#### 7.45.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

Definition at line 331 of file [swapChain.cpp](#).

#### 7.45.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (
    const SwapChain & swapChain) const [inline], [nodiscard]
```

Definition at line 48 of file [SwapChain.hpp](#).

References [m\\_swapChainDepthFormat](#), and [m\\_swapChainImageFormat](#).

#### 7.45.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

#### 7.45.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

#### 7.45.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

#### 7.45.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

#### 7.45.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

Definition at line 103 of file [swapChain.cpp](#).

#### 7.45.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

References [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

#### 7.45.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

Definition at line 42 of file [SwapChain.hpp](#).

References [m\\_swapChainExtent](#).

#### 7.45.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

#### 7.45.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (
    const unsigned long index) const [inline], [nodiscard]
```

Definition at line 33 of file [SwapChain.hpp](#).

References [m\\_swapChainFrameBuffers](#).

#### 7.45.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (
    const int index) const [inline], [nodiscard]
```

Definition at line 35 of file [SwapChain.hpp](#).

References [m\\_swapChainImageViews](#).

#### 7.45.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

Definition at line 34 of file [SwapChain.hpp](#).

References [m\\_renderPass](#).

#### 7.45.3.17 `getSwapChainExtent()`

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

Definition at line 38 of file [SwapChain.hpp](#).

References [m\\_swapChainExtent](#).

#### 7.45.3.18 `getSwapChainImageFormat()`

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

Definition at line 37 of file [SwapChain.hpp](#).

References [m\\_swapChainImageFormat](#).

#### 7.45.3.19 `height()`

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

Definition at line 40 of file [SwapChain.hpp](#).

References [m\\_swapChainExtent](#).

#### 7.45.3.20 `imageCount()`

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

Definition at line 36 of file [SwapChain.hpp](#).

References [m\\_swapChainImages](#).

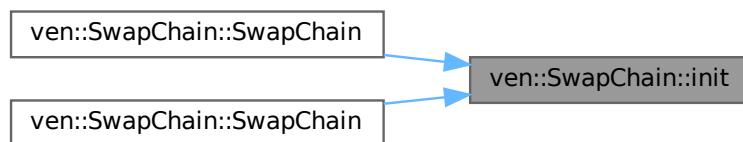
#### 7.45.3.21 `init()`

```
void ven::SwapChain::init () [private]
```

Definition at line 39 of file [swapChain.cpp](#).

Referenced by [SwapChain\(\)](#), and [SwapChain\(\)](#).

Here is the caller graph for this function:



### 7.45.3.22 operator=( )

```
SwapChain & ven::SwapChain::operator= (
    const SwapChain & ) [delete]
```

### 7.45.3.23 submitCommandBuffers( )

```
VkResult ven::SwapChain::submitCommandBuffers (
    const VkCommandBuffer * buffers,
    const uint32_t * imageIndex)
```

Definition at line 56 of file [swapChain.cpp](#).

References [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

### 7.45.3.24 width( )

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

Definition at line 39 of file [SwapChain.hpp](#).

References [m\\_swapChainExtent](#).

## 7.45.4 Member Data Documentation

### 7.45.4.1 m\_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0} [private]
```

Definition at line 87 of file [SwapChain.hpp](#).

### 7.45.4.2 m\_depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::m_depthImageMemory [private]
```

Definition at line 72 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

### 7.45.4.3 m\_depthImages

```
std::vector<VkImage> ven::SwapChain::m_depthImages [private]
```

Definition at line 71 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.4 m\_depthImageViews

```
std::vector<VkImageView> ven::SwapChain::m_depthImageViews [private]
```

Definition at line 73 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.5 m\_device

```
Device& ven::SwapChain::m_device [private]
```

Definition at line 77 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.6 m\_imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_imageAvailableSemaphores [private]
```

Definition at line 83 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.7 m\_imagesInFlight

```
std::vector<VkFence> ven::SwapChain::m_imagesInFlight [private]
```

Definition at line 86 of file [SwapChain.hpp](#).

#### 7.45.4.8 m\_inFlightFences

```
std::vector<VkFence> ven::SwapChain::m_inFlightFences [private]
```

Definition at line 85 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.9 m\_oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::m_oldSwapChain [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

Referenced by [SwapChain\(\)](#).

#### 7.45.4.10 m\_renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_renderFinishedSemaphores [private]
```

Definition at line 84 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.11 m\_renderPass

```
VkRenderPass ven::SwapChain::m_renderPass {} [private]
```

Definition at line 69 of file [SwapChain.hpp](#).

Referenced by [getRenderPass\(\)](#), and [~SwapChain\(\)](#).

#### 7.45.4.12 m\_swapChain

```
VkSwapchainKHR ven::SwapChain::m_swapChain {} [private]
```

Definition at line 80 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

#### 7.45.4.13 m\_swapChainDepthFormat

```
VkFormat ven::SwapChain::m_swapChainDepthFormat {} [private]
```

Definition at line 65 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#).

#### 7.45.4.14 m\_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

Definition at line 66 of file [SwapChain.hpp](#).

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

#### 7.45.4.15 m\_swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::m_swapChainFrameBuffers [private]
```

Definition at line 68 of file [SwapChain.hpp](#).

Referenced by [getFrameBuffer\(\)](#), and [~SwapChain\(\)](#).

#### 7.45.4.16 m\_swapChainImageFormat

```
VkFormat ven::SwapChain::m_swapChainImageFormat {} [private]
```

Definition at line 64 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

#### 7.45.4.17 m\_swapChainImages

```
std::vector<VkImage> ven::SwapChain::m_swapChainImages [private]
```

Definition at line 74 of file [SwapChain.hpp](#).

Referenced by [imageCount\(\)](#).

#### 7.45.4.18 m\_swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::m_swapChainImageViews [private]
```

Definition at line 75 of file [SwapChain.hpp](#).

Referenced by [getImageView\(\)](#), and [~SwapChain\(\)](#).

#### 7.45.4.19 m\_windowExtent

```
VkExtent2D ven::SwapChain::m_windowExtent [private]
```

Definition at line 78 of file [SwapChain.hpp](#).

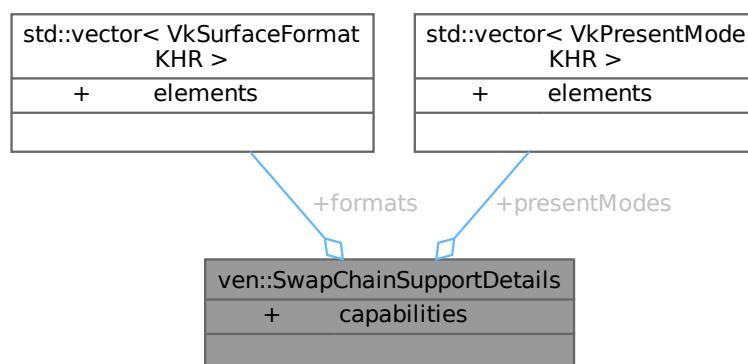
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[SwapChain.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[swapChain.cpp](#)

### 7.46 ven::SwapChainSupportDetails Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::SwapChainSupportDetails:



## Public Attributes

- VkSurfaceCapabilitiesKHR [capabilities](#)
- std::vector< VkSurfaceFormatKHR > [formats](#)
- std::vector< VkPresentModeKHR > [presentModes](#)

### 7.46.1 Detailed Description

Definition at line 16 of file [Device.hpp](#).

### 7.46.2 Member Data Documentation

#### 7.46.2.1 capabilities

VkSurfaceCapabilitiesKHR [ven::SwapChainSupportDetails::capabilities](#)

Definition at line 17 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

#### 7.46.2.2 formats

std::vector<VkSurfaceFormatKHR> [ven::SwapChainSupportDetails::formats](#)

Definition at line 18 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

#### 7.46.2.3 presentModes

std::vector<VkPresentModeKHR> [ven::SwapChainSupportDetails::presentModes](#)

Definition at line 19 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

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

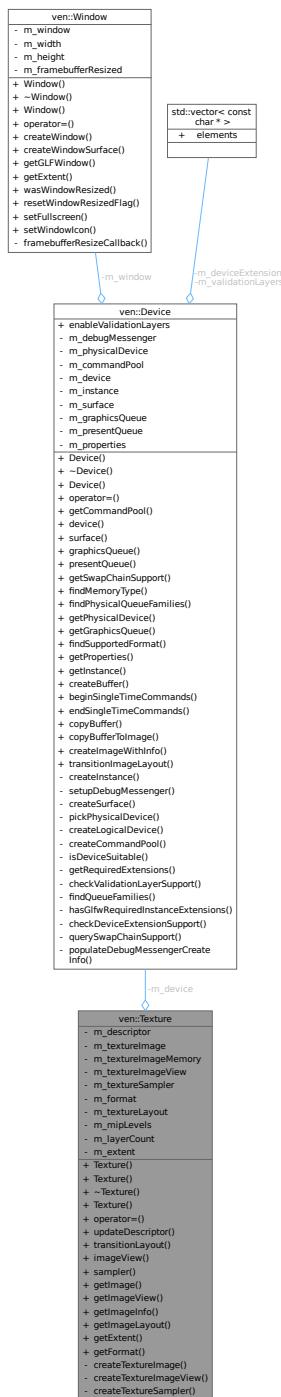
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp

## 7.47 ven::Texture Class Reference

Class for texture.

```
#include <Texture.hpp>
```

Collaboration diagram for ven::Texture:



## Public Member Functions

- `Texture (Device &device, const std::string &textureFilepath)`
- `Texture (Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage, VkSampleCountFlagBits sampleCount)`
- `~Texture ()`
- `Texture (const Texture &) = delete`
- `Texture & operator= (const Texture &) = delete`
- `void updateDescriptor ()`
- `void transitionLayout (VkCommandBuffer commandBuffer, VkImageLayout oldLayout, VkImageLayout newLayout) const`
- `VkImageView imageView () const`
- `VkSampler sampler () const`
- `VkImage getImage () const`
- `VkImageView getImageView () const`
- `VkDescriptorImageInfo getImageInfo () const`
- `VkImageLayout getImageLayout () const`
- `VkExtent3D getExtent () const`
- `VkFormat getFormat () const`

## Private Member Functions

- `void createTextureImage (const std::string &filepath)`
- `void createTextureImageView (VkImageType viewType)`
- `void createTextureSampler ()`

## Private Attributes

- `VkDescriptorImageInfo m_descriptor {}`
- `Device & m_device`
- `VkImage m_textureImage = nullptr`
- `VkDeviceMemory m_textureImageMemory = nullptr`
- `VkImageView m_textureImageView = nullptr`
- `VkSampler m_textureSampler = nullptr`
- `VkFormat m_format`
- `VkImageLayout m_textureLayout {}`
- `uint32_t m_mipLevels {1}`
- `uint32_t m_layerCount {1}`
- `VkExtent3D m_extent {}`

### 7.47.1 Detailed Description

Class for texture.

Definition at line 20 of file [Texture.hpp](#).

## 7.47.2 Constructor & Destructor Documentation

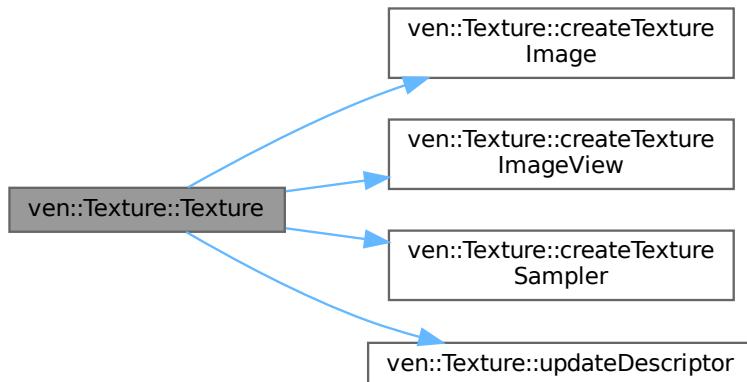
### 7.47.2.1 Texture() [1/3]

```
ven::Texture::Texture (
    Device & device,
    const std::string & textureFilepath)
```

Definition at line 7 of file [texture.cpp](#).

References [createTextureImage\(\)](#), [createTextureImageView\(\)](#), [createTextureSampler\(\)](#), and [updateDescriptor\(\)](#).

Here is the call graph for this function:



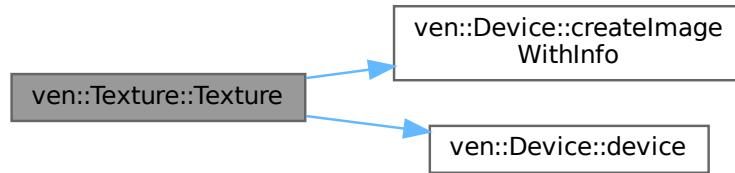
### 7.47.2.2 Texture() [2/3]

```
ven::Texture::Texture (
    Device & device,
    VkFormat format,
    VkExtent3D extent,
    VkImageUsageFlags usage,
    VkSampleCountFlagBits sampleCount)
```

Definition at line 15 of file [texture.cpp](#).

References [ven::Device::createImageWithInfo\(\)](#), [ven::Device::device\(\)](#), [m\\_descriptor](#), [m\\_textureImage](#), [m\\_textureImageMemory](#), [m\\_textureImageView](#), and [m\\_textureSampler](#).

Here is the call graph for this function:



#### 7.47.2.3 ~Texture()

```
ven::Texture::~Texture ()
```

Definition at line 89 of file [texture.cpp](#).

#### 7.47.2.4 Texture() [3/3]

```
ven::Texture::Texture (
    const Texture & ) [delete]
```

### 7.47.3 Member Function Documentation

#### 7.47.3.1 createTextureImage()

```
void ven::Texture::createTextureImage (
    const std::string & filepath) [private]
```

Definition at line 104 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



#### 7.47.3.2 `createTextureImageView()`

```
void ven::Texture::createTextureImageView (
    VkImageViewType viewType) [private]
```

Definition at line 191 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



#### 7.47.3.3 `createTextureSampler()`

```
void ven::Texture::createTextureSampler () [private]
```

Definition at line 209 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



#### 7.47.3.4 `getExtent()`

```
VkExtent3D ven::Texture::getExtent () const [inline], [nodiscard]
```

Definition at line 40 of file [Texture.hpp](#).

References [m\\_extent](#).

### 7.47.3.5 getFormat()

```
VkFormat ven::Texture::getFormat () const [inline], [nodiscard]
```

Definition at line 41 of file [Texture.hpp](#).

References [m\\_format](#).

### 7.47.3.6 getImage()

```
VkImage ven::Texture::getImage () const [inline], [nodiscard]
```

Definition at line 36 of file [Texture.hpp](#).

References [m\\_textureImage](#).

### 7.47.3.7 getImageInfo()

```
VkDescriptorImageInfo ven::Texture::getImageInfo () const [inline], [nodiscard]
```

Definition at line 38 of file [Texture.hpp](#).

References [m\\_descriptor](#).

### 7.47.3.8 getImageLayout()

```
VkImageLayout ven::Texture::getImageLayout () const [inline], [nodiscard]
```

Definition at line 39 of file [Texture.hpp](#).

References [m\\_textureLayout](#).

### 7.47.3.9 getImageView()

```
VkImageView ven::Texture::getImageView () const [inline], [nodiscard]
```

Definition at line 37 of file [Texture.hpp](#).

References [m\\_textureImageView](#).

### 7.47.3.10 imageView()

```
VkImageView ven::Texture::imageView () const [inline], [nodiscard]
```

Definition at line 34 of file [Texture.hpp](#).

References [m\\_textureImageView](#).

#### 7.47.3.11 operator=( )

```
Texture & ven::Texture::operator= (
    const Texture & ) [delete]
```

#### 7.47.3.12 sampler()

```
VkSampler ven::Texture::sampler () const [inline], [nodiscard]
```

Definition at line 35 of file [Texture.hpp](#).

References [m\\_textureSampler](#).

#### 7.47.3.13 transitionLayout()

```
void ven::Texture::transitionLayout (
    VkCommandBuffer commandBuffer,
    VkImageLayout oldLayout,
    VkImageLayout newLayout) const
```

Definition at line 239 of file [texture.cpp](#).

#### 7.47.3.14 updateDescriptor()

```
void ven::Texture::updateDescriptor ()
```

Definition at line 97 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



### 7.47.4 Member Data Documentation

#### 7.47.4.1 m\_descriptor

```
VkDescriptorImageInfo ven::Texture::m_descriptor {} [private]
```

Definition at line 49 of file [Texture.hpp](#).

Referenced by [getImageInfo\(\)](#), and [Texture\(\)](#).

#### 7.47.4.2 m\_device

```
Device& ven::Texture::m_device [private]
```

Definition at line 50 of file [Texture.hpp](#).

#### 7.47.4.3 m\_extent

```
VkExtent3D ven::Texture::m_extent {} [private]
```

Definition at line 59 of file [Texture.hpp](#).

Referenced by [getExtent\(\)](#).

#### 7.47.4.4 m\_format

```
VkFormat ven::Texture::m_format [private]
```

Definition at line 55 of file [Texture.hpp](#).

Referenced by [getFormat\(\)](#).

#### 7.47.4.5 m\_layerCount

```
uint32_t ven::Texture::m_layerCount {1} [private]
```

Definition at line 58 of file [Texture.hpp](#).

#### 7.47.4.6 m\_mipLevels

```
uint32_t ven::Texture::m_mipLevels {1} [private]
```

Definition at line 57 of file [Texture.hpp](#).

#### 7.47.4.7 m\_textureImage

```
VkImage ven::Texture::m_textureImage = nullptr [private]
```

Definition at line 51 of file [Texture.hpp](#).

Referenced by [getImage\(\)](#), and [Texture\(\)](#).

#### 7.47.4.8 m\_textureImageMemory

```
VkDeviceMemory ven::Texture::m_textureImageMemory = nullptr [private]
```

Definition at line 52 of file [Texture.hpp](#).

Referenced by [Texture\(\)](#).

#### 7.47.4.9 m\_textureImageView

```
VkImageView ven::Texture::m_textureImageView = nullptr [private]
```

Definition at line 53 of file [Texture.hpp](#).

Referenced by [getImageView\(\)](#), [imageView\(\)](#), and [Texture\(\)](#).

#### 7.47.4.10 m\_textureLayout

```
VkImageLayout ven::Texture::m_textureLayout {} [private]
```

Definition at line 56 of file [Texture.hpp](#).

Referenced by [getImageLayout\(\)](#).

#### 7.47.4.11 m\_textureSampler

```
VkSampler ven::Texture::m_textureSampler = nullptr [private]
```

Definition at line 54 of file [Texture.hpp](#).

Referenced by [sampler\(\)](#), and [Texture\(\)](#).

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

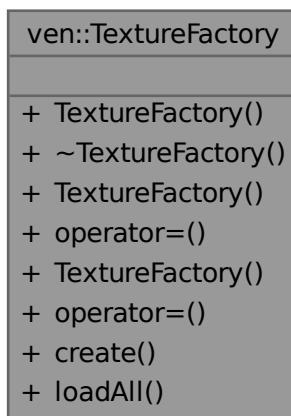
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Texture.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[texture.cpp](#)

## 7.48 ven::TextureFactory Class Reference

Class for [Texture](#) factory.

```
#include <Texture.hpp>
```

Collaboration diagram for ven::TextureFactory:



## Public Member Functions

- `TextureFactory ()=delete`
- `~TextureFactory ()=default`
- `TextureFactory (const TextureFactory &)=delete`
- `TextureFactory & operator= (const TextureFactory &)=delete`
- `TextureFactory (TextureFactory &&)=delete`
- `TextureFactory & operator= (TextureFactory &&)=delete`

## Static Public Member Functions

- `static std::unique_ptr< Texture > create (Device &device, const std::string &filepath)`
- `static std::unordered_map< std::string, std::shared_ptr< Texture > > loadAll (Device &device, const std::string &folderPath)`

## 7.48.1 Detailed Description

Class for `Texture` factory.

Definition at line 21 of file `Texture.hpp`.

## 7.48.2 Constructor & Destructor Documentation

### 7.48.2.1 TextureFactory() [1/3]

```
ven::TextureFactory::TextureFactory () [delete]
```

### 7.48.2.2 ~TextureFactory()

```
ven::TextureFactory::~TextureFactory () [default]
```

### 7.48.2.3 TextureFactory() [2/3]

```
ven::TextureFactory::TextureFactory (
    const TextureFactory & ) [delete]
```

### 7.48.2.4 TextureFactory() [3/3]

```
ven::TextureFactory::TextureFactory (
    TextureFactory && ) [delete]
```

### 7.48.3 Member Function Documentation

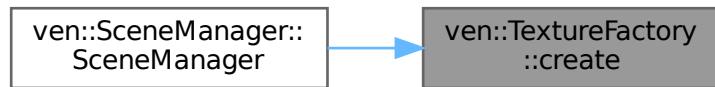
#### 7.48.3.1 `create()`

```
static std::unique_ptr< Texture > ven::TextureFactory::create (
    Device & device,
    const std::string & filepath) [inline], [static]
```

Definition at line 33 of file [Texture.hpp](#).

Referenced by [ven::SceneManager::SceneManager\(\)](#).

Here is the caller graph for this function:



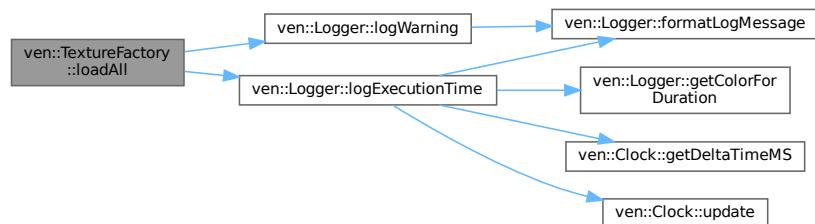
#### 7.48.3.2 `loadAll()`

```
std::unordered_map< std::string, std::shared_ptr< ven::Texture > > ven::TextureFactory::loadAll (
    Device & device,
    const std::string & folderPath) [static]
```

Definition at line 6 of file [Texture.cpp](#).

References [ven::Logger::logExecutionTime\(\)](#), and [ven::Logger::logWarning\(\)](#).

Here is the call graph for this function:



### 7.48.3.3 operator=() [1/2]

```
TextureFactory & ven::TextureFactory::operator= (
    const TextureFactory & ) [delete]
```

### 7.48.3.4 operator=() [2/2]

```
TextureFactory & ven::TextureFactory::operator= (
    TextureFactory && ) [delete]
```

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

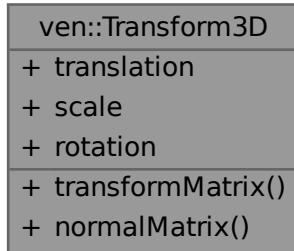
- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Texture.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Texture.cpp](#)

## 7.49 ven::Transform3D Class Reference

Class for 3D transformation.

```
#include <Transform3D.hpp>
```

Collaboration diagram for ven::Transform3D:



### Public Member Functions

- `glm::mat4 transformMatrix () const`
- `glm::mat3 normalMatrix () const`

### Public Attributes

- `glm::vec3 translation {}`
- `glm::vec3 scale {}`
- `glm::vec3 rotation {}`

### 7.49.1 Detailed Description

Class for 3D transformation.

Definition at line 18 of file [Transform3D.hpp](#).

### 7.49.2 Member Function Documentation

#### 7.49.2.1 normalMatrix()

```
glm::mat3 ven::Transform3D::normalMatrix () const [inline], [nodiscard]
```

Definition at line 34 of file [Transform3D.hpp](#).

References [transformMatrix\(\)](#).

Here is the call graph for this function:



#### 7.49.2.2 transformMatrix()

```
glm::mat4 ven::Transform3D::transformMatrix () const [inline], [nodiscard]
```

Definition at line 22 of file [Transform3D.hpp](#).

References [rotation](#), [scale](#), and [translation](#).

Referenced by [normalMatrix\(\)](#).

Here is the caller graph for this function:



### 7.49.3 Member Data Documentation

#### 7.49.3.1 rotation

```
glm::vec3 ven::Transform3D::rotation {}
```

Definition at line 38 of file [Transform3D.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::EventManager::moveCamera\(\)](#), and [transformMatrix\(\)](#).

#### 7.49.3.2 scale

```
glm::vec3 ven::Transform3D::scale {}
```

Definition at line 37 of file [Transform3D.hpp](#).

Referenced by [transformMatrix\(\)](#).

#### 7.49.3.3 translation

```
glm::vec3 ven::Transform3D::translation {}
```

Definition at line 36 of file [Transform3D.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::EventManager::moveCamera\(\)](#), and [transformMatrix\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/[Transform3D.hpp](#)

## 7.50 ven::Model::Vertex Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model::Vertex:

ven::Model::Vertex
+ position
+ color
+ normal
+ uv
+ operator==( )
+ getBindingDescriptions( )
+ getAttributeDescriptions( )

## 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 {}`

### 7.50.1 Detailed Description

Definition at line 33 of file `Model.hpp`.

### 7.50.2 Member Function Documentation

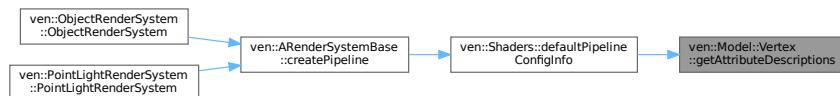
#### 7.50.2.1 `getAttributeDescriptions()`

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions()
() [static]
```

Definition at line 93 of file `model.cpp`.

Referenced by `ven::Shaders::defaultPipelineConfigInfo()`.

Here is the caller graph for this function:



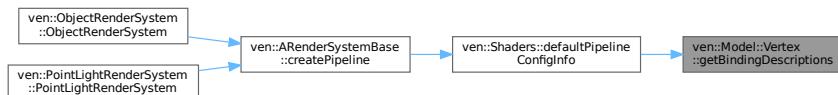
### 7.50.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()  
[static]
```

Definition at line 84 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



### 7.50.2.3 operator==( )

```
bool ven::Model::Vertex::operator== (const Vertex & other) const [inline]
```

Definition at line 42 of file [Model.hpp](#).

References [color](#), [normal](#), [position](#), and [uv](#).

## 7.50.3 Member Data Documentation

### 7.50.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 35 of file [Model.hpp](#).

Referenced by [operator==\( \)](#).

### 7.50.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 36 of file [Model.hpp](#).

Referenced by [operator==\( \)](#).

### 7.50.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 34 of file [Model.hpp](#).

Referenced by [operator==\(\)](#), and [ven::Model::Builder::processMesh\(\)](#).

### 7.50.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 37 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

## 7.51 ven::Window Class Reference

Class for window.

```
#include <Window.hpp>
```

Collaboration diagram for ven::Window:

ven::Window	
-	m_window
-	m_width
-	m_height
-	m_framebufferResized
+	Window()
+	~Window()
+	Window()
+	operator=(=)
+	createWindow()
+	createWindowSurface()
+	getGLFWWindow()
+	getExtent()
+	wasWindowResized()
+	resetWindowResizedFlag()
+	setFullscreen()
+	setWindowIcon()
-	framebufferResizeCallback()

### Public Member Functions

- `Window (const uint32_t width=DEFAULT_WIDTH, const uint32_t height=DEFAULT_HEIGHT)`
- `~Window ()`
- `Window (const Window &)=delete`
- `Window & operator= (const Window &)=delete`
- `GLFWwindow * createWindow (uint32_t width, uint32_t height, const std::string &title)`
- `void createWindowSurface (VkInstance instance, VkSurfaceKHR *surface) const`
- `GLFWwindow * getGLFWWindow () const`
- `VkExtent2D getExtent () const`
- `bool wasWindowResized () const`
- `void resetWindowResizedFlag ()`
- `void setFullscreen (bool fullscreen, uint32_t width, uint32_t height)`
- `void setWindowIcon (const std::string &path)`

### Static Private Member Functions

- static void `framebufferResizeCallback (GLFWwindow *window, int width, int height)`

### Private Attributes

- `GLFWwindow * m_window {nullptr}`
- `uint32_t m_width {0}`
- `uint32_t m_height {0}`
- `bool m_framebufferResized = false`

## 7.51.1 Detailed Description

Class for window.

Definition at line 26 of file `Window.hpp`.

## 7.51.2 Constructor & Destructor Documentation

### 7.51.2.1 `Window()` [1/2]

```
ven::Window::Window (
    const uint32_t width = DEFAULT_WIDTH,
    const uint32_t height = DEFAULT_HEIGHT) [inline], [explicit]
```

Definition at line 30 of file `Window.hpp`.

References `setWindowIcon()`.

Here is the call graph for this function:



### 7.51.2.2 ~Window()

```
ven::Window::~Window () [inline]
```

Definition at line 31 of file [Window.hpp](#).

References [m\\_window](#).

### 7.51.2.3 Window() [2/2]

```
ven::Window::Window (
    const Window & ) [delete]
```

## 7.51.3 Member Function Documentation

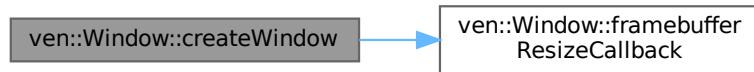
### 7.51.3.1 createWindow()

```
GLFWwindow * ven::Window::createWindow (
    uint32_t width,
    uint32_t height,
    const std::string & title) [nodiscard]
```

Definition at line 8 of file [window.cpp](#).

References [framebufferResizeCallback\(\)](#).

Here is the call graph for this function:



### 7.51.3.2 createWindowSurface()

```
void ven::Window::createWindowSurface (
    VkInstance instance,
    VkSurfaceKHR * surface) const
```

Definition at line 27 of file [window.cpp](#).

Referenced by [ven::Device::createSurface\(\)](#).

Here is the caller graph for this function:



### 7.51.3.3 framebufferResizeCallback()

```
void ven::Window::framebufferResizeCallback (
    GLFWwindow * window,
    int width,
    int height) [static], [private]
```

Definition at line 34 of file [window.cpp](#).

References [m\\_framebufferResized](#).

Referenced by [createWindow\(\)](#).

Here is the caller graph for this function:



### 7.51.3.4 getExtent()

```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

Definition at line 41 of file [Window.hpp](#).

References [m\\_height](#), and [m\\_width](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



### 7.51.3.5 getGLFWWindow()

```
GLFWwindow * ven::Window::getGLFWWindow () const [inline], [nodiscard]
```

Definition at line 39 of file [Window.hpp](#).

References [m\\_window](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



### 7.51.3.6 operator=()

```
Window & ven::Window::operator= (
    const Window & ) [delete]
```

### 7.51.3.7 resetWindowResizedFlag()

```
void ven::Window::resetWindowResizedFlag () [inline]
```

Definition at line 43 of file [Window.hpp](#).

References [m\\_framebufferResized](#).

### 7.51.3.8 setFullscreen()

```
void ven::Window::setFullscreen (
    bool fullscreen,
    uint32_t width,
    uint32_t height)
```

Definition at line 42 of file [window.cpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



### 7.51.3.9 setWindowIcon()

```
void ven::Window::setWindowIcon (
    const std::string & path)
```

Definition at line 62 of file [window.cpp](#).

Referenced by [Window\(\)](#).

Here is the caller graph for this function:



### 7.51.3.10 wasWindowResized()

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

Definition at line 42 of file [Window.hpp](#).

References [m\\_framebufferResized](#).

## 7.51.4 Member Data Documentation

### 7.51.4.1 m\_framebufferResized

```
bool ven::Window::m_framebufferResized = false [private]
```

Definition at line 56 of file [Window.hpp](#).

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).

### 7.51.4.2 m\_height

```
uint32_t ven::Window::m_height {0} [private]
```

Definition at line 54 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

#### 7.51.4.3 m\_width

```
uint32_t ven::Window::m_width {0} [private]
```

Definition at line 53 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

#### 7.51.4.4 m\_window

```
GLFWwindow* ven::Window::m_window {nullptr} [private]
```

Definition at line 52 of file [Window.hpp](#).

Referenced by [getGLFWWindow\(\)](#), and [~Window\(\)](#).

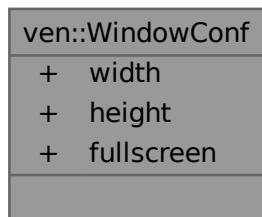
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Window.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/[window.cpp](#)

## 7.52 ven::WindowConf Struct Reference

```
#include <Config.hpp>
```

Collaboration diagram for ven::WindowConf:



### Public Attributes

- `uint16_t width = DEFAULT_WIDTH`
- `uint16_t height = DEFAULT_HEIGHT`
- `bool fullscreen = false`

### 7.52.1 Detailed Description

Definition at line 19 of file [Config.hpp](#).

### 7.52.2 Member Data Documentation

#### 7.52.2.1 fullscreen

```
bool ven::WindowConf::fullscreen = false
```

Definition at line 23 of file [Config.hpp](#).

#### 7.52.2.2 height

```
uint16_t ven::WindowConf::height = DEFAULT_HEIGHT
```

Definition at line 22 of file [Config.hpp](#).

#### 7.52.2.3 width

```
uint16_t ven::WindowConf::width = DEFAULT_WIDTH
```

Definition at line 21 of file [Config.hpp](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/[Config.hpp](#)



# Chapter 8

## File Documentation

### 8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment\_point\_light.frag File Reference

#### 8.2 fragment\_point\_light.frag

[Go to the documentation of this file.](#)

```
00001 #version 450
00002
00003 layout(location = 0) in vec2 fragOffset;
00004 layout(location = 0) out vec4 outColor;
00005
00006 struct PointLight {
00007     vec4 position; // ignore w
00008     vec4 color; // w is intensity
00009     float shininess;
00010 };
00011
00012 layout(set = 0, binding = 0) uniform GlobalUbo {
00013     mat4 projection;
00014     mat4 view;
00015     mat4 invView;
00016     vec4 ambientLightColor; // w is intensity
00017     PointLight pointLights[10];
00018     int numLights;
00019 } ubo;
00020
00021 layout(push_constant) uniform Push {
00022     vec4 position;
00023     vec4 color;
00024     float radius;
00025 } push;
00026
00027 const float M_PI = 3.1415926538;
00028
00029 void main() {
00030     float dis = length(fragOffset);
00031     if (dis >= 1.0) {
00032         discard;
00033     }
00034     float cosDis = 0.5 * (cos(dis * M_PI) + 1.0);
00035     outColor = vec4(push.color.rgb + 0.5 * cosDis, cosDis);
00036 }
00037 }
```

### 8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment\_shader.frag File Reference

#### 8.4 fragment\_shader.frag

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 fragColor;
00004 layout(location = 1) in vec3 fragPosWorld;
00005 layout(location = 2) in vec3 fragNormalWorld;
00006 layout(location = 3) in vec2 fragUv;
00007
00008 layout(location = 0) out vec4 outColor;
00009
00010 struct PointLight {
00011     vec4 position; // ignore w
00012     vec4 color; // w is intensity
00013     float shininess;
00014 };
00015
00016 layout(set = 0, binding = 0) uniform GlobalUbo {
00017     mat4 projection;
00018     mat4 view;
00019     mat4 invView;
00020     vec4 ambientLightColor; // w is intensity
00021     PointLight pointLights[10];
00022     int numLights;
00023 } ubo;
00024
00025 layout(set = 1, binding = 1) uniform sampler2D diffuseMap;
00026
00027 layout(push_constant) uniform Push {
00028     mat4 modelMatrix;
00029     mat4 normalMatrix;
00030 } push;
00031
00032 void main() {
00033     vec3 specularLight = vec3(0.0);
00034     vec3 surfaceNormal = normalize(gl_FrontFacing ? fragNormalWorld : -fragNormalWorld);
00035     vec3 diffuseLight = ubo.ambientLightColor.rgb * ubo.ambientLightColor.a;
00036
00037     vec3 cameraPosWorld = ubo.invView[3].xyz;
00038     vec3 viewDirection = normalize(cameraPosWorld - fragPosWorld);
00039
00040     for (int i = 0; i < ubo.numLights; i++) {
00041         PointLight light = ubo.pointLights[i];
00042         vec3 directionToLight = light.position.xyz - fragPosWorld;
00043         float distanceSquared = dot(directionToLight, directionToLight);
00044         float attenuation = distanceSquared > 0.001 ? (light.position.w + 1.0) / distanceSquared : 0.0;
00045         directionToLight = normalize(directionToLight);
00046
00047         float cosAngIncidence = max(dot(surfaceNormal, directionToLight), 0);
00048         vec3 intensity = light.color.rgb * light.color.a * attenuation;
00049
00050         if (cosAngIncidence > 0) {
00051             vec3 halfVector = normalize(directionToLight + viewDirection);
00052             float cosAngHalf = max(dot(surfaceNormal, halfVector), 0);
00053
00054             float specular = pow(cosAngHalf, light.shininess);
00055
00056             diffuseLight += intensity * cosAngIncidence;
00057             specularLight += intensity * specular;
00058         }
00059     }
00060
00061     vec3 color = texture(diffuseMap, fragUv).xyz;
00062     outColor = vec4(diffuseLight * color + specularLight, 1.0);
00063 }

```

## 8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex\_point\_light.vert File Reference

## 8.6 vertex\_point\_light.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 const vec2 OFFSETS[6] = vec2[](
00004     vec2(-1.0, -1.0),
00005     vec2(-1.0, 1.0),
00006     vec2(1.0, -1.0),
00007     vec2(1.0, 1.0),

```

```

00008 vec2(-1.0, 1.0),
00009 vec2(1.0, 1.0)
00010 );
00011
00012 layout(location = 0) out vec2 fragOffset;
00013
00014 struct PointLight {
00015     vec4 position; // ignore w
00016     vec4 color; // w is intensity
00017     float shininess;
00018 };
00019
00020 layout(set = 0, binding = 0) uniform GlobalUbo {
00021     mat4 projection;
00022     mat4 view;
00023     mat4 invView;
00024     vec4 ambientLightColor; // w is intensity
00025     PointLight pointLights[10];
00026     int numLights;
00027 } ubo;
00028
00029 layout(push_constant) uniform Push {
00030     vec4 position;
00031     vec4 color;
00032     float radius;
00033 } push;
00034
00035 void main() {
00036     fragOffset = OFFSETS[gl_VertexIndex];
00037     vec3 cameraRightWorld = vec3(ubo.view[0][0], ubo.view[1][0], ubo.view[2][0]);
00038     vec3 cameraUpWorld = vec3(ubo.view[0][1], ubo.view[1][1], ubo.view[2][1]);
00039
00040     vec3 positionWorld = push.position.xyz
00041     + push.radius * fragOffset.x * cameraRightWorld
00042     + push.radius * fragOffset.y * cameraUpWorld;
00043
00044     gl_Position = ubo.projection * ubo.view * vec4(positionWorld, 1.0);
00045 }

```

## 8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex\_shader.vert File Reference

### 8.8 vertex\_shader.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 position;
00004 layout(location = 1) in vec3 color;
00005 layout(location = 2) in vec3 normal;
00006 layout(location = 3) in vec2 uv;
00007
00008 layout(location = 0) out vec3 fragColor;
00009 layout(location = 1) out vec3 fragPosWorld;
00010 layout(location = 2) out vec3 fragNormalWorld;
00011 layout(location = 3) out vec2 fragUv;
00012
00013 struct PointLight {
00014     vec4 position; // ignore w
00015     vec4 color; // w is intensity
00016     float shininess;
00017 };
00018
00019 layout(set = 0, binding = 0) uniform GlobalUbo {
00020     mat4 projection;
00021     mat4 view;
00022     mat4 invView;
00023     vec4 ambientLightColor; // w is intensity
00024     PointLight pointLights[10];
00025     int numLights;
00026 } ubo;
00027
00028 layout(set = 1, binding = 0) uniform ObjectBufferData {
00029     mat4 modelMatrix;
00030     mat4 normalMatrix;
00031 } object;
00032

```

```

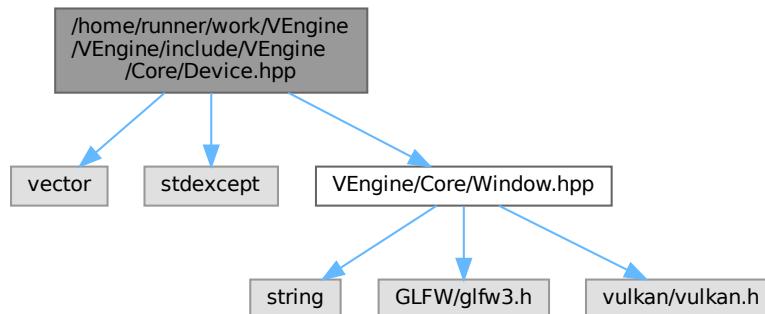
00033 layout(push_constant) uniform Push {
00034     mat4 modelMatrix;
00035     mat4 normalMatrix;
00036 } push;
00037
00038 void main() {
00039     vec4 positionWorld = object.modelMatrix * vec4(position, 1.0);
00040     gl_Position = ubo.projection * ubo.view * positionWorld;
00041     fragNormalWorld = normalize(mat3(object.normalMatrix) * normal);
00042     fragPosWorld = positionWorld.xyz;
00043     fragColor = color;
00044     fragUV = uv;
00045 }

```

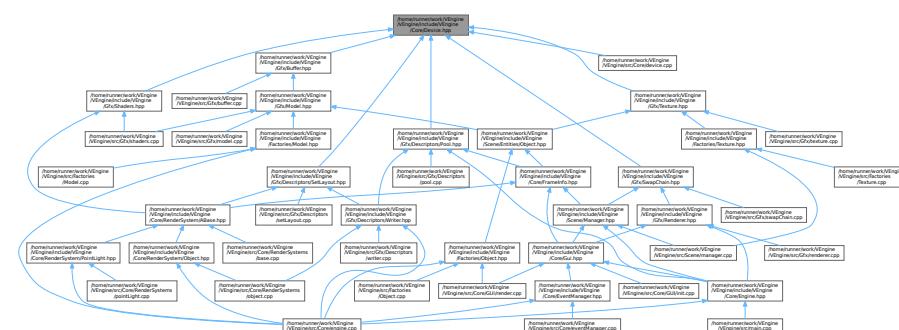
## 8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp File Reference

This file contains the Device class.

```
#include <vector>
#include <stdexcept>
#include "VEngine/Core/Window.hpp"
Include dependency graph for Device.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

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

*Class for device.*

## Namespaces

- namespace `ven`

### 8.9.1 Detailed Description

This file contains the `Device` class.

Definition in file [Device.hpp](#).

## 8.10 Device.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  /// @file Device.hpp
00003  /// @brief This file contains the Device class
00004  /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <vector>
00010 #include <stdexcept>
00011
00012 #include "VEngine/Core/Window.hpp"
00013
00014 namespace ven {
00015
00016     struct SwapChainSupportDetails {
00017         VkSurfaceCapabilitiesKHR capabilities;
00018         std::vector<VkSurfaceFormatKHR> formats;
00019         std::vector<VkPresentModeKHR> presentModes;
00020     };
00021
00022     struct QueueFamilyIndices {
00023         uint32_t graphicsFamily{};
00024         uint32_t presentFamily{};
00025         bool graphicsFamilyHasValue = false;
00026         bool presentFamilyHasValue = false;
00027         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
00028             presentFamilyHasValue; }
00029     };
00030
00031     /// @class Device
00032     /// @brief Class for device
00033     /// @namespace ven
00034     ///
00035     class Device {
00036
00037     public:
00038
00039         #ifdef NDEBUG
00040             const bool enableValidationLayers = false;
00041         #else
00042             const bool enableValidationLayers = true;
00043         #endif
00044
00045         explicit Device(Window &window);
00046         ~Device();
00047
00048         Device(const Device&) = delete;

```

```

00049     Device& operator=(const Device&) = delete;
00050
00051     [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
00052     [[nodiscard]] VkDevice device() const { return m_device; }
00053     [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
00054     [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
00055     [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }
00056
00057     [[nodiscard]] SwapChainSupportDetails querySwapChainSupport() const { return
00058         SwapChainSupport(m_physicalDevice); }
00059     [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags
00060         properties) const;
00061     [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
00062         findQueueFamilies(m_physicalDevice); }
00063     [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00064     [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00065     [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
00066         VkImageTiling tiling, VkFormatFeatureFlags features) const;
00067     [[nodiscard]] VkPhysicalDeviceProperties getProperties() const { return m_properties; }
00068     [[nodiscard]] VkInstance getInstance() const { return m_instance; }
00069
00070     // Buffer Helper Functions
00071     void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
00072         properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00073     [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00074     void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00075     void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00076     void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
00077         uint32_t layerCount) const;
00078     void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
00079         properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00080     void transitionImageLayout(VkImage image, VkFormat format, VkImageLayout oldLayout,
00081         VkImageLayout newLayout, uint32_t mipLevels = 1, uint32_t layerCount = 1) const;
00082
00083     private:
00084
00085     void createInstance();
00086     void setupDebugMessenger();
00087     void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00088     void pickPhysicalDevice();
00089     void createLogicalDevice();
00090     void createCommandPool();
00091
00092     // helper functions
00093     bool isDeviceSuitable(VkPhysicalDevice device) const;
00094     [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00095     [[nodiscard]] bool checkValidationLayerSupport() const;
00096     QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00097     static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
00098         &createInfo);
00099     void hasGlfwRequiredInstanceExtensions() const;
00100     bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00101     SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00102
00103     Window &m_window;
00104     VkDebugUtilsMessengerEXT m_debugMessenger;
00105     VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00106     VkCommandPool m_commandPool;
00107     VkDevice m_device;
00108     VkInstance m_instance;
00109     VkSurfaceKHR m_surface;
00110     VkQueue m_graphicsQueue;
00111     VkQueue m_presentQueue;
00112     VkPhysicalDeviceProperties m_properties;
00113
00114     const std::vector<const char *> m_validationLayers = {"VK_LAYER_KHRONOS_validation"};
00115     const std::vector<const char *> m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00116
00117 }; // class Device
00118 } // namespace ven

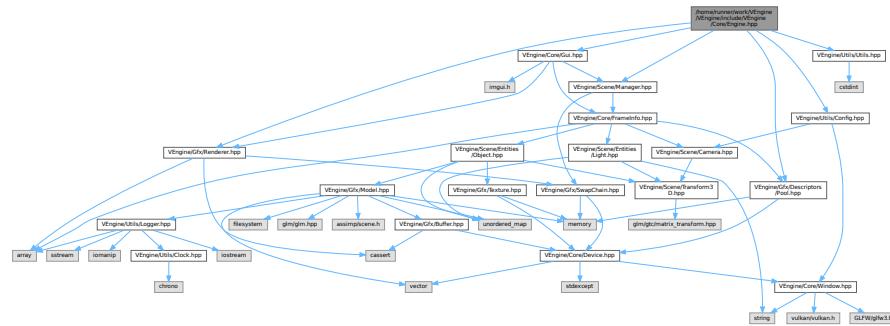
```

## 8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Engine.hpp](#) File Reference

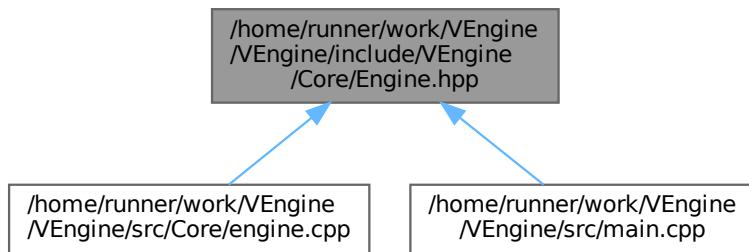
This file contains the Engine class.

```
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Gfx/Renderer.hpp"
```

```
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Scene/Manager.hpp"
#include "VEngine/Utils/Utils.hpp"
#include "VEngine/Utils/Config.hpp"
Include dependency graph for Engine.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::Engine`

*Class for engine.*

## Namespaces

- namespace `ven`

### 8.11.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

## 8.12 Engine.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Engine.hpp
00003 /// @brief This file contains the Engine class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/Gui.hpp"
00010 #include "VEngine/Gfx/Renderer.hpp"
00011 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00012 #include "VEngine/Scene/Manager.hpp"
00013 #include "VEngine/Utils/Utils.hpp"
00014 #include "VEngine/Utils/Config.hpp"
00015
00016 namespace ven {
00017
00018 /**
00019 /// @class Engine
00020 /// @brief Class for engine
00021 /// @namespace ven
00022 /**
00023 class Engine {
00024
00025     public:
00026
00027     explicit Engine(const Config& config);
00028     ~Engine() { Gui::cleanup(); }
00029
00030     Engine(const Engine&) = delete;
00031     Engine operator=(const Engine&) = delete;
00032
00033     void mainLoop();
00034
00035     private:
00036
00037     void loadObjects();
00038
00039     ENGINE_STATE m_state{EXIT};
00040
00041     Window m_window;
00042     Camera m_camera;
00043     Gui m_gui;
00044     Device m_device{m_window};
00045     SceneManager m_sceneManager{m_device};
00046     Renderer m_renderer{m_window, m_device};
00047     std::unique_ptr<DescriptorPool> m_globalPool;
00048     std::vector<std::unique_ptr<DescriptorPool>> m_framePools;
00049
00050 }; // class Engine
00051
00052 } // namespace ven

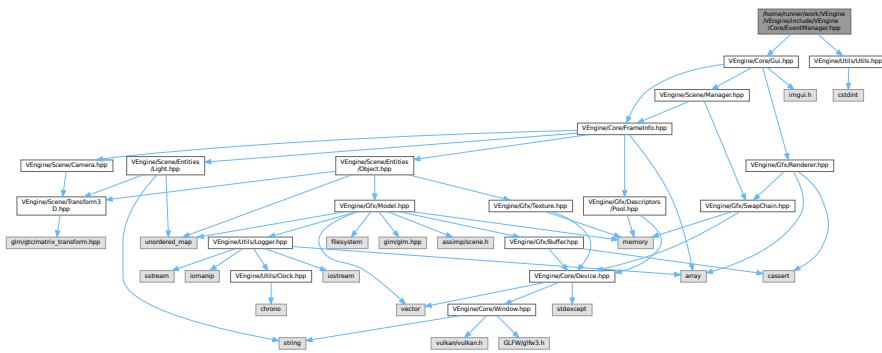
```

## 8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp File Reference

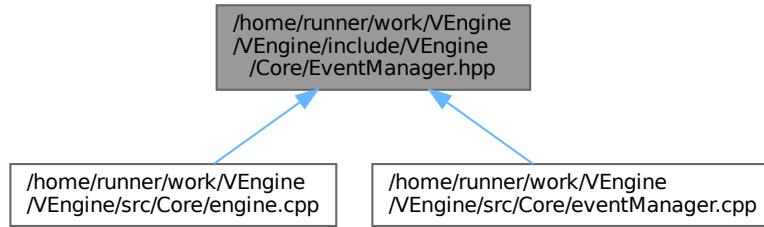
This file contains the EventManager class.

```
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Utils/Utils.hpp"
```

Include dependency graph for EventManager.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct `ven::KeyAction`
  - struct `ven::KeyMappings`
  - class `ven::EventManager`

*Class for event manager.*

## Namespaces

- namespace `ven`

## Variables

- static constexpr float ven::EPSILON = std::numeric\_limits<float>::epsilon()
  - static constexpr KeyMappings ven::DEFAULT\_KEY\_MAPPINGS {}

### **8.13.1 Detailed Description**

This file contains the EventManager class.

Definition in file [EventManager.hpp](#).

## 8.14 EventManager.hpp

[Go to the documentation of this file.](#)

```

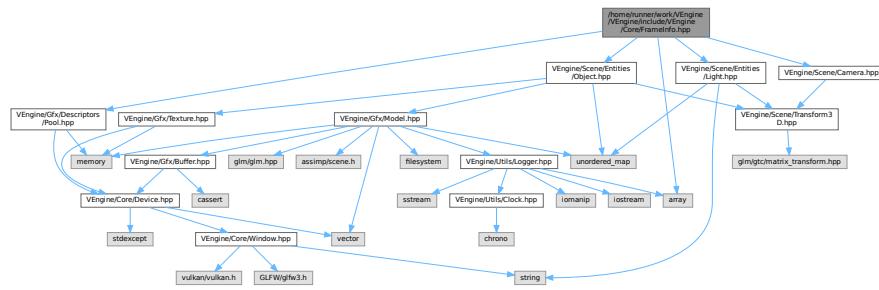
00001 /**
00002 /// @file EventManager.hpp
00003 /// @brief This file contains the EventManager class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/Gui.hpp"
00010 #include "VEngine/Utils/Utils.hpp"
00011
00012 namespace ven {
00013
00014     struct KeyAction {
00015         uint16_t key;
00016         glm::vec3 dir;
00017         glm::vec3 value;
00018     };
00019
00020     struct KeyMappings {
00021         uint16_t moveLeft = GLFW_KEY_A;
00022         uint16_t moveRight = GLFW_KEY_D;
00023         uint16_t moveForward = GLFW_KEY_W;
00024         uint16_t moveBackward = GLFW_KEY_S;
00025         uint16_t moveUp = GLFW_KEY_SPACE;
00026         uint16_t moveDown = GLFW_KEY_LEFT_SHIFT;
00027         uint16_t lookLeft = GLFW_KEY_LEFT;
00028         uint16_t lookRight = GLFW_KEY_RIGHT;
00029         uint16_t lookUp = GLFW_KEY_UP;
00030         uint16_t lookDown = GLFW_KEY_DOWN;
00031         uint16_t toggleGui = GLFW_KEY_0;
00032     };
00033
00034     static constexpr float EPSILON = std::numeric_limits<float>::epsilon();
00035     static constexpr KeyMappings DEFAULT_KEY_MAPPINGS{};
00036
00037 /**
00038 /// @class EventManager
00039 /// @brief Class for event manager
00040 /// @namespace ven
00041 /**
00042 class EventManager {
00043
00044     public:
00045
00046         EventManager() = default;
00047         ~EventManager() = default;
00048
00049         EventManager(const EventManager&) = delete;
00050         EventManager& operator=(const EventManager&) = delete;
00051
00052         void handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera, Gui& gui,
00053         float dt) const;
00054
00055     private:
00056
00057         static void moveCamera(GLFWwindow* window, Camera& camera, float dt);
00058         static void updateEngineState(ENGINE_STATE *engineState, const ENGINE_STATE newState) {
00059             *engineState = newState; }
00060         static bool isKeyJustPressed(GLFWwindow* window, long unsigned int key, std::array<bool,
00061             GLFW_KEY_LAST>& keyStates);
00062
00063         template<typename Iterator>
00064         static void processKeyActions(GLFWwindow* window, Iterator begin, Iterator end);
00065
00066         mutable std::array<bool, GLFW_KEY_LAST> m_keyState{};
00067     }; // class EventManager
00068
00069 } // namespace ven

```

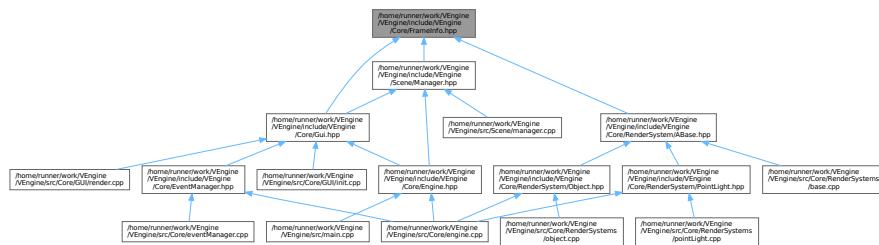
## 8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp File Reference

This file contains the FrameInfo class.

```
#include <array>
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Scene/Camera.hpp"
#include "VEngine/Scene/Entities/Object.hpp"
#include "VEngine/Scene/Entities/Light.hpp"
Include dependency graph for FrameInfo.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct `ven::PointLightData`
- struct `ven::ObjectBufferData`
- struct `ven::GlobalUbo`
- struct `ven::FrameInfo`

## Namespaces

- namespace `ven`

## Variables

- static constexpr float `ven::DEFAULT_AMBIENT_LIGHT_INTENSITY` = .2F
- static constexpr `glm::vec4` `ven::DEFAULT_AMBIENT_LIGHT_COLOR` = {`glm::vec3(1.F)`, `DEFAULT_AMBIENT_LIGHT_INTENS`

### 8.15.1 Detailed Description

This file contains the `FrameInfo` class.

Definition in file [FrameInfo.hpp](#).

## 8.16 FrameInfo.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file FrameInfo.hpp
00003 /// @brief This file contains the FrameInfo class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <array>
00010
00011 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00012 #include "VEngine/Scene/Camera.hpp"
00013 #include "VEngine/Scene/Entities/Object.hpp"
00014 #include "VEngine/Scene/Entities/Light.hpp"
00015
00016 namespace ven {
00017
00018 static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F;
00019 static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F),
00020                         DEFAULT_AMBIENT_LIGHT_INTENSITY};
00021
00022     struct PointLightData
00023     {
00024         glm::vec4 position{};
00025         glm::vec4 color{};
00026         float shininess{32.F};
00027         float padding[3]; // Pad to 32 bytes
00028     };
00029
00030     struct ObjectBufferData {
00031         glm::mat4 modelMatrix{1.F};
00032         glm::mat4 normalMatrix{1.F};
00033     };
00034
00035     struct GlobalUbo
00036     {
00037         glm::mat4 projection{1.F};
00038         glm::mat4 view{1.F};
00039         glm::mat4 inverseView{1.F};
00040         glm::vec4 ambientLightColor{DEFAULT_AMBIENT_LIGHT_COLOR};
00041         std::array<PointLightData, MAX_LIGHTS> pointLights;
00042         uint8_t numLights;
00043     };
00044
00045     struct FrameInfo
00046     {
00047         unsigned long frameIndex;
00048         float frameTime;
00049         VkCommandBuffer commandBuffer;
00050         Camera &camera;
00051         VkDescriptorSet globalDescriptorSet;
00052         DescriptorPool &frameDescriptorPool;
00053         Object::Map &objects;
00054         Light::Map &lights;
00055     };
00056 } // namespace ven

```

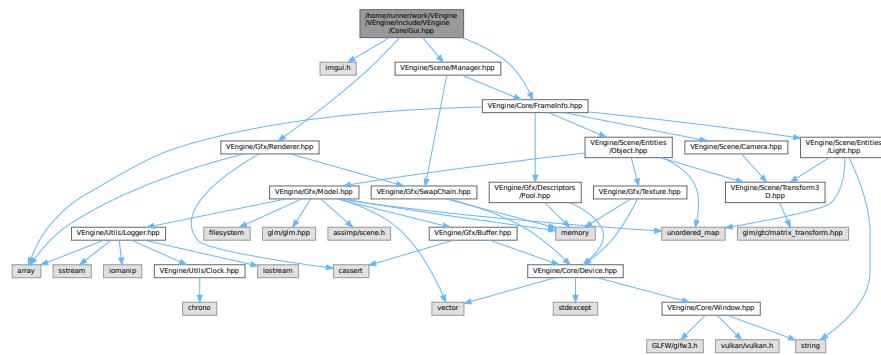
## 8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← Gui.hpp File Reference

This file contains the ImGuiWindowManager class.

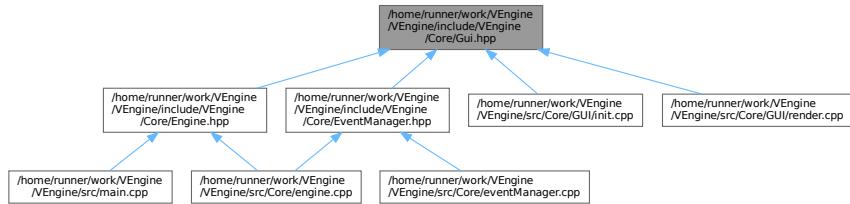
```
#include <imgui.h>
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Scene/Manager.hpp"
```

```
#include "VEngine/Gfx/Renderer.hpp"
```

Include dependency graph for Gui.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::Gui`  
*Class for Gui.*
- struct `ven::Gui::ClockData`
- struct `ven::Gui::funcs`

## Namespaces

- namespace `ven`

## Enumerations

- enum `ven::GUI_STATE` : `uint8_t` { `ven::SHOW_EDITOR` = 0 , `ven::SHOW_PLAYER` = 1 , `ven::HIDDEN` = 2 }

## Variables

- static constexpr `uint16_t` `ven::DESCRIPTOR_COUNT` = 1000

### 8.17.1 Detailed Description

This file contains the `ImGuiWindowManager` class.

Definition in file [Gui.hpp](#).

## 8.18 Gui.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Gui.hpp
00003  * @brief This file contains the ImGuiWindowManager class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <imgui.h>
00010
00011 #include "VEngine/Core/FrameInfo.hpp"
00012 #include "VEngine/Scene/Manager.hpp"
00013 #include "VEngine/Gfx/Renderer.hpp"
00014
00015 namespace ven {
00016
00017     static constexpr uint16_t DESCRIPTOR_COUNT = 1000;
00018
00019     enum GUI_STATE : uint8_t {
00020         SHOW_EDITOR = 0,
00021         SHOW_PLAYER = 1,
00022         HIDDEN = 2
00023     };
00024
00025 /**
00026  * @class Gui
00027  * @brief Class for Gui
00028  * @namespace ven
00029 /**
00030 class Gui {
00031
00032     struct ClockData {
00033         float deltaTimeMS{0.0F};
00034         float fps{0.0F};
00035     };
00036
00037     public:
00038
00039         Gui() = default;
00040         ~Gui() = default;
00041
00042         Gui(const Gui&) = delete;
00043         Gui& operator=(const Gui&) = delete;
00044
00045         void init(GLFWwindow* window, VkInstance instance, const Device* device, VkRenderPass
renderPass);
00046
00047         void render(Renderer *renderer, SceneManager& sceneManager, Camera& camera,
VkPhysicalDevice physicalDevice, GlobalUbo& ubo, const ClockData& clockData);
00048         static void cleanup();
00049
00050         void setState(GUI_STATE state) { m_state = state; }
00051         [[nodiscard]] GUI_STATE getState() const { return m_state; }
00052         [[nodiscard]] std::vector<unsigned int> *getObjectsToRemove() { return &m_objectsToRemove;
}
00053         [[nodiscard]] std::vector<unsigned int> *getLightsToRemove() { return &m_lightsToRemove; }
00054
00055     private:
00056
00057         static void initStyle();
00058         static void renderFrameWindow(const ClockData& clockData);
00059         static void cameraSection(Camera& camera);
00060         static void inputsSection(const ImGuiIO& io);
00061         static void rendererSection(Renderer *renderer, GlobalUbo& ubo);
00062         static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
00063         void objectsSection(SceneManager& sceneManager);
00064         void lightsSection(SceneManager& sceneManager);
00065
00066         struct funcs { static bool IsLegacyNativeDupe(const ImGuiKey key) { return key >= 0 && key
< 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist

```

```

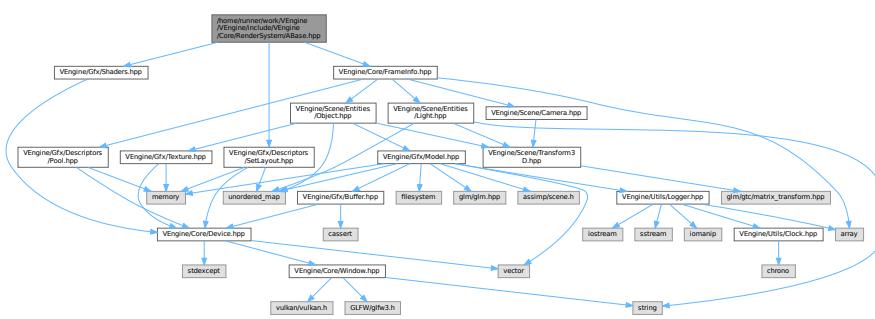
00067
00068     ImGuiIO* m_io{nullptr};
00069     GUI_STATE m_state{HIDDEN};
00070     float m_intensity{1.0f};
00071     float m_shininess{DEFAULT_SHININESS};
00072
00073     std::vector<unsigned int> m_objectsToRemove;
00074     std::vector<unsigned int> m_lightsToRemove;
00075
00076 }; // class Gui
00077
00078 } // namespace ven

```

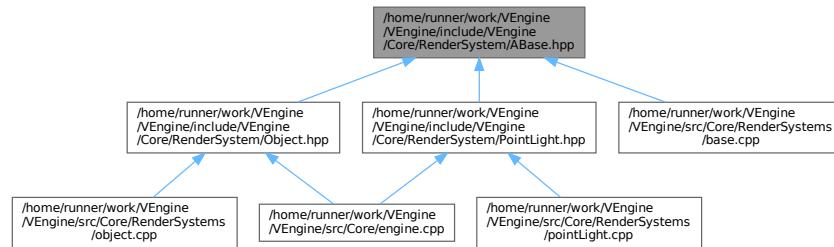
## 8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ABase.hpp File Reference

This file contains the ARenderSystemBase class.

```
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
#include "VEngine/Gfx/Shaders.hpp"
Include dependency graph for ABase.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [ven::ARenderSystemBase](#)

*Abstract class for render system base.*

## Namespaces

- namespace [ven](#)

### 8.19.1 Detailed Description

This file contains the `ARenderSystemBase` class.

Definition in file [ABase.hpp](#).

## 8.20 ABase.hpp

[Go to the documentation of this file.](#)

```

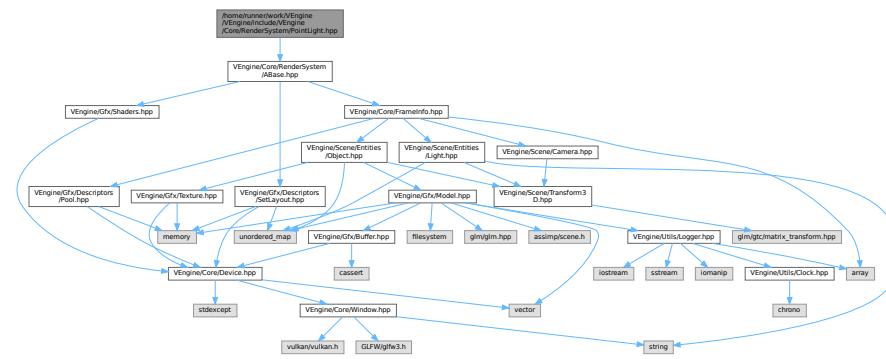
00001 /**
00002 // @file ABase.hpp
00003 // @brief This file contains the ARenderSystemBase class
00004 // @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/FrameInfo.hpp"
00010 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00011 #include "VEngine/Gfx/Shaders.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 // @class ARenderSystemBase
00017 // @brief Abstract class for render system base
00018 // @namespace ven
00019 /**
00020 class ARenderSystemBase {
00021
00022     public:
00023
00024         explicit ARenderSystemBase(Device& device) : m_device{device} {}
00025         virtual ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(),
00026             m_pipelineLayout, nullptr); }
00027         virtual void render(const FrameInfo &frameInfo) const = 0;
00028
00029     protected:
00030
00031         void createPipelineLayout(VkDescriptorSetLayout globalsetLayout, uint32_t
00032             pushConstantSize);
00032         void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
00033             std::string &shadersFragPath, bool isLight);
00034
00034         [[nodiscard]] Device& getDevice() const { return m_device; }
00035         [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
00036         [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }
00037
00038         std::unique_ptr<DescriptorsetLayout> renderSystemLayout;
00039
00040     private:
00041
00042         Device &m_device;
00043         VkPipelineLayout m_pipelineLayout{nullptr};
00044         std::unique_ptr<Shaders> m_shaders;
00045
00046
00047     }; // class ARenderSystemBase
00048
00049 } // namespace ven

```

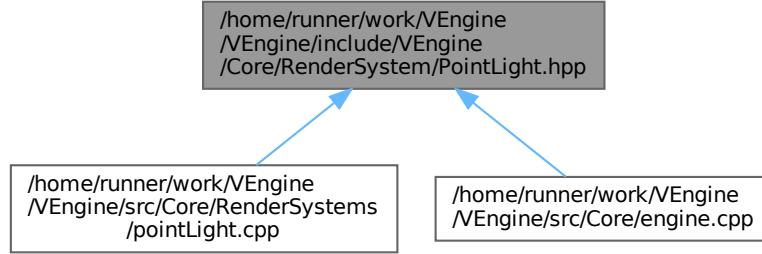
## 8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/PointLight.hpp File Reference

This file contains the PointLightRenderSystem class.

```
#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for PointLight.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- struct [ven::LightPushConstantData](#)
- class [ven::PointLightRenderSystem](#)

*Class for point light system.*

### Namespaces

- namespace [ven](#)

#### 8.21.1 Detailed Description

This file contains the PointLightRenderSystem class.

Definition in file [PointLight.hpp](#).

## 8.22 PointLight.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file PointLight.hpp
00003 /// @brief This file contains the PointLightRenderSystem class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/RenderSystem/ABase.hpp"
00010
00011 namespace ven {
00012
00013     struct LightPushConstantData {
00014         glm::vec4 position{};
00015         glm::vec4 color{};
00016         float radius;
00017     };
00018
00019 /**
00020 /// @class PointLightRenderSystem
00021 /// @brief Class for point light system
00022 /// @namespace ven
00023 /**
00024 class PointLightRenderSystem final : public ARenderSystemBase {
00025
00026     public:
00027
00028     explicit PointLightRenderSystem(Device& device, const VkRenderPass renderPass, const
00029                                     VkDescriptorSetLayout globalsetLayout) : ARenderSystemBase(device) {
00030         createPipelineLayout(globalsetLayout, sizeof(LightPushConstantData));
00031         createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_point_light.spv",
00032                      std::string(SHADERS_BIN_PATH) + "fragment_point_light.spv", true);
00033     }
00034
00035     PointLightRenderSystem(const PointLightRenderSystem&) = delete;
00036     PointLightRenderSystem& operator=(const PointLightRenderSystem&) = delete;
00037
00038     void render(const FrameInfo &frameInfo) const override;
00039 }; // class PointLightRenderSystem
00040 } // namespace ven

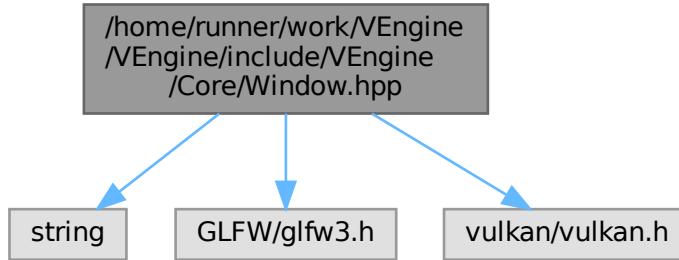
```

## 8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← Window.hpp File Reference

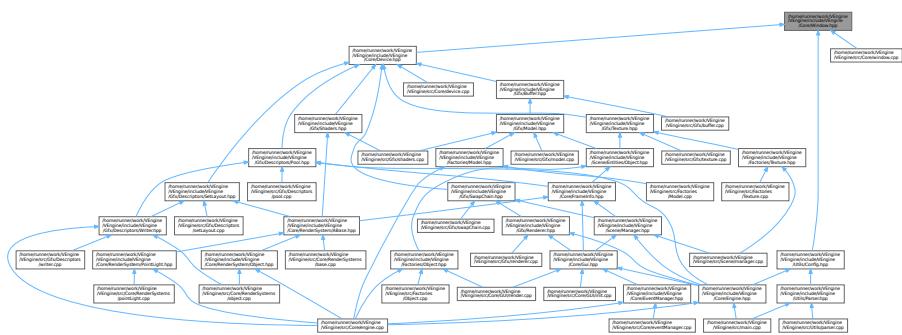
This file contains the Window class.

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

Include dependency graph for Window.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Window](#)

*Class for window.*

## Namespaces

- namespace [ven](#)

## Macros

- `#define GLFW_INCLUDE_VULKAN`

## Variables

- static constexpr std::string\_view [ven::DEFAULT\\_TITLE](#) = "VEngine"
- static constexpr uint32\_t [ven::DEFAULT\\_WIDTH](#) = 1920
- static constexpr uint32\_t [ven::DEFAULT\\_HEIGHT](#) = 1080

### 8.23.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

### 8.23.2 Macro Definition Documentation

#### 8.23.2.1 GLFW\_INCLUDE\_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

## 8.24 Window.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 /**
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     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00018     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00019     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00020
00021 /**
00022 /// @class Window
00023 /// @brief Class for window
00024 /// @namespace ven
00025 /**
00026 class Window {
00027
00028     public:
00029
00030         explicit Window(const uint32_t width = DEFAULT_WIDTH, const uint32_t height =
00031             DEFAULT_HEIGHT) : m_window(createWindow(width, height, DEFAULT_TITLE.data())), m_width(width),
00032             m_height(height) { setWindowIcon("assets/icons/icon64x64.png"); }
00033         ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr; }
00034
00035         Window(const Window&) = delete;
00036         Window& operator=(const Window&) = delete;
00037
00038         [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
00039             &title);
00040         void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00041
00042         [[nodiscard]] GLFWwindow* getGLFWwindow() const { return m_window; }
00043         [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; }
00044         [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00045         void resetWindowResizedFlag() { m_framebufferResized = false; }
00046
00047         void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);
00048         void setWindowIcon(const std::string& path);
00049
00050         static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00051
```

```

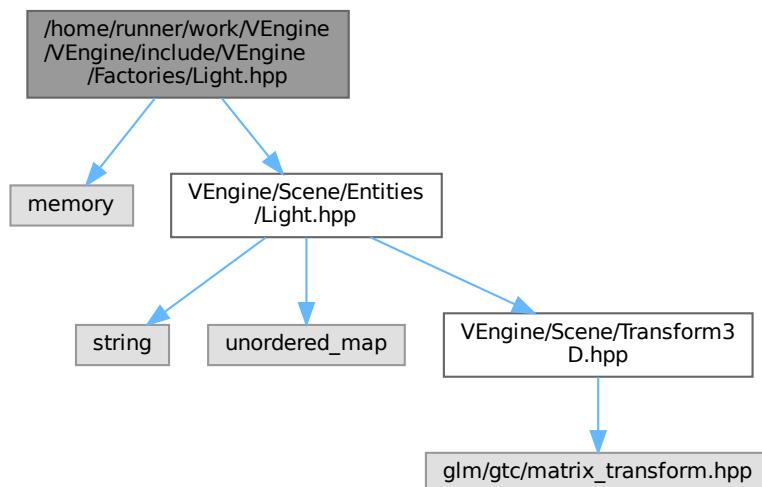
00052     GLFWwindow* m_window{nullptr};
00053     uint32_t m_width{0};
00054     uint32_t m_height{0};
00055
00056     bool m_framebufferResized = false;
00057
00058 }; // class Window
00059
00060 } // namespace ven

```

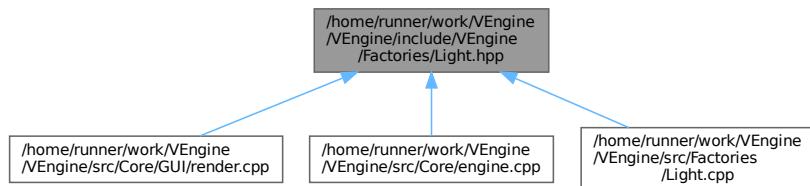
## 8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/<sup>↳</sup> Light.hpp File Reference

This file contains the Light Factory class.

```
#include <memory>
#include "VEngine/Scene/Entities/Light.hpp"
Include dependency graph for Light.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::LightFactory](#)

*Class for light factory.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr [Transform3D ven::DEFAULT\\_TRANSFORM](#) = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F, 0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}}

### 8.25.1 Detailed Description

This file contains the Light Factory class.

Definition in file [Light.hpp](#).

## 8.26 Light.hpp

[Go to the documentation of this file.](#)

```

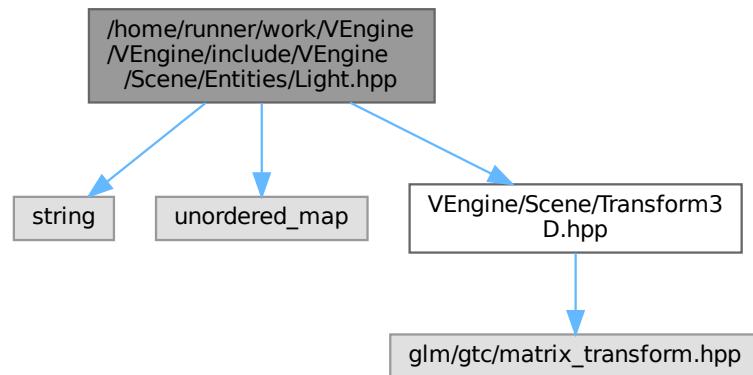
00001 /**
00002 /// @file Light.hpp
00003 /// @brief This file contains the Light Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Scene/Entities/Light.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr Transform3D DEFAULT_TRANSFORM = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F,
0.1F, 0.F}, .rotation = {0.F, 0.F, 0.F}};
00016
00017 /**
00018 /// @class LightFactory
00019 /// @brief Class for light factory
00020 /// @namespace ven
00021 /**
00022 class LightFactory {
00023
00024     public:
00025
00026         LightFactory() = delete;
00027         ~LightFactory() = default;
00028
00029         LightFactory(const LightFactory&) = delete;
00030         LightFactory& operator=(const LightFactory&) = delete;
00031         LightFactory(LightFactory&&) = delete;
00032         LightFactory& operator=(LightFactory&&) = delete;
00033
00034         static std::unique_ptr<Light> create(const Transform3D &transform = DEFAULT_TRANSFORM,
00035             glm::vec4 color = DEFAULT_LIGHT_COLOR);
00036         static std::unique_ptr<Light> duplicate(const Light &cpyLight) { return
00037             create(cpyLight.transform, cpyLight.color); }
00038
00039     private:
00040
00041         static unsigned int m_currentLightId;
00042
00043 }; // class LightFactory
00044 } // namespace ven

```

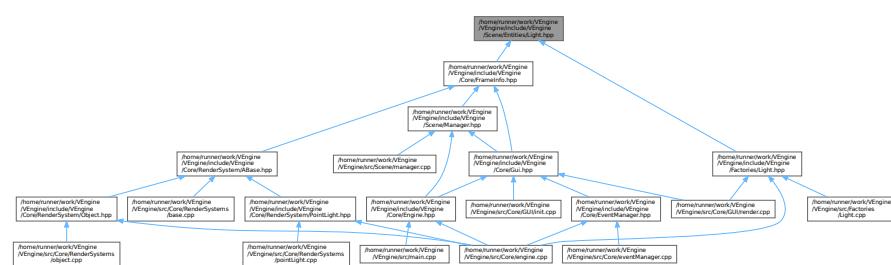
## 8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp File Reference

This file contains the Light class.

```
#include <string>
#include <unordered_map>
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Light.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::Light`  
*Class for light.*

## Namespaces

- namespace `ven`

## Variables

- static constexpr float `ven::DEFAULT_LIGHT_INTENSITY` = .2F
- static constexpr float `ven::DEFAULT_LIGHT_RADIUS` = 0.1F
- static constexpr float `ven::DEFAULT_SHININESS` = 32.F
- static constexpr glm::vec4 `ven::DEFAULT_LIGHT_COLOR` = {glm::vec3(1.F), `DEFAULT_LIGHT_INTENSITY`}
- static constexpr uint8\_t `ven::MAX_LIGHTS` = 10

### 8.27.1 Detailed Description

This file contains the Light class.

Definition in file [Light.hpp](#).

## 8.28 Light.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Light.hpp
00003  * @brief This file contains the Light class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <string>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Scene/Transform3D.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
00017     static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
00018     static constexpr float DEFAULT_SHININESS = 32.F;
00019     static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY};
00020
00021     static constexpr uint8_t MAX_LIGHTS = 10;
00022
00023 /**
00024  * @class Light
00025  * @brief Class for light
00026  * @namespace ven
00027 /**
00028 class Light {
00029
00030     public:
00031
00032         using Map = std::unordered_map<unsigned int, Light>;
00033
00034         explicit Light(const unsigned int objId) : m_lightId{objId} {}
00035
00036         ~Light() = default;
00037
00038         Light(const Light&) = delete;
00039         Light& operator=(const Light&) = delete;
00040         Light(Light&&) = default;
00041         Light& operator=(Light&&) = default;
00042
00043         [[nodiscard]] unsigned int getId() const { return m_lightId; }
00044         [[nodiscard]] std::string getName() const { return m_name; }
00045         [[nodiscard]] float getShininess() const { return m_shininess; }
00046
00047         void setName(const std::string &name) { m_name = name; }
00048         void setShininess(const float shininess) { m_shininess = shininess; }
00049
00050         glm::vec4 color{DEFAULT_LIGHT_COLOR};
00051         Transform3D transform{};
00052
00053     private:
00054
00055         unsigned int m_lightId;
00056         std::string m_name{"point light"};
00057         float m_shininess{DEFAULT_SHININESS};
00058
00059     }; // class Light
00060 } // namespace ven

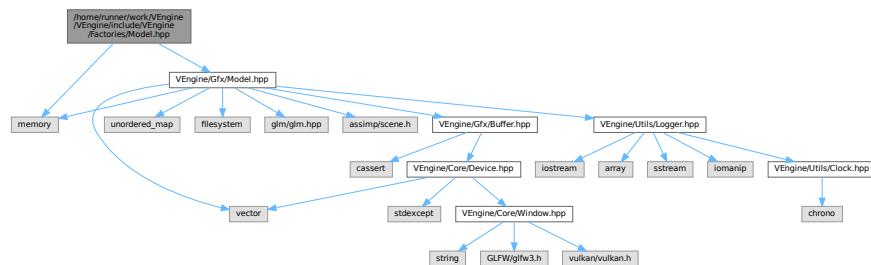
```

## 8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp File Reference

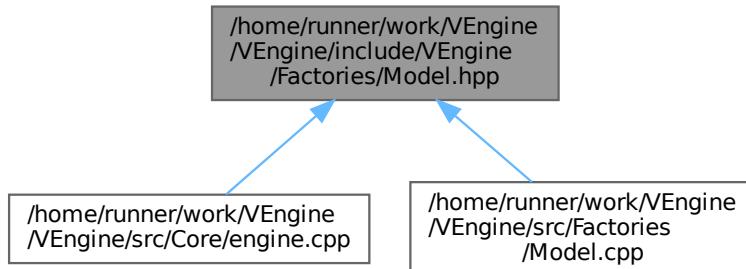
This file contains the Model Factory class.

```
#include <memory>
#include "VEngine/Gfx/Model.hpp"
```

Include dependency graph for Model.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- class [ven::ModelFactory](#)

*Class for Model factory.*

### Namespaces

- namespace [ven](#)

#### 8.29.1 Detailed Description

This file contains the Model Factory class.

Definition in file [Model.hpp](#).

## 8.30 Model.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Gfx/Model.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 /// @class ModelFactory
00017 /// @brief Class for Model factory
00018 /// @namespace ven
00019 /**
00020 class ModelFactory {
00021
00022     public:
00023
00024         ModelFactory() = delete;
00025         ~ModelFactory() = default;
00026
00027         ModelFactory(const ModelFactory&) = delete;
00028         ModelFactory& operator=(const ModelFactory&) = delete;
00029         ModelFactory(ModelFactory&&) = delete;
00030         ModelFactory& operator=(ModelFactory&&) = delete;
00031
00032         static std::unique_ptr<Model> create(Device& device, const std::string& filepath);
00033         static std::unordered_map<std::string, std::shared_ptr<Model>> loadAll(Device& device,
00034         const std::string& folderPath);
00035     }; // class ModelFactory
00036
00037 } // namespace ven

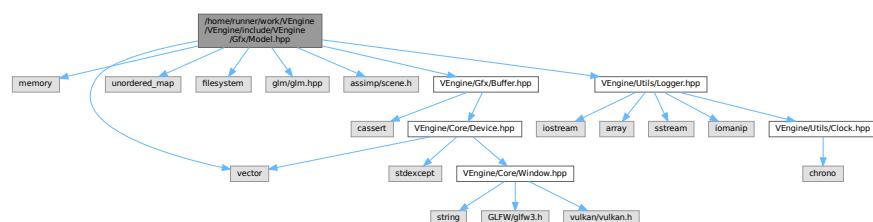
```

## 8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Model.hpp File Reference

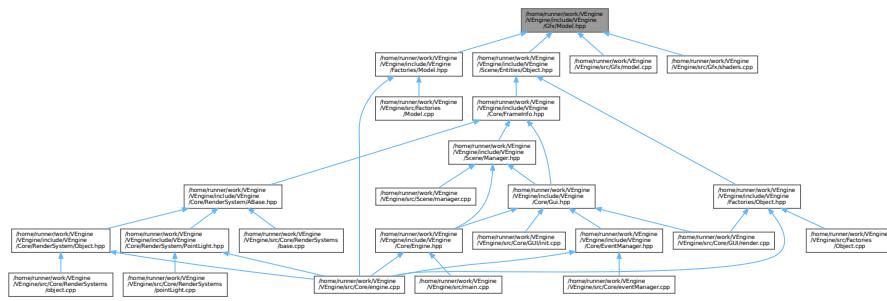
This file contains the Model class.

```
#include <memory>
#include <vector>
#include <unordered_map>
#include <filesystem>
#include <glm/glm.hpp>
#include <assimp/scene.h>
#include "VEngine/Gfx/Buffer.hpp"
#include "VEngine/Utils/Logger.hpp"
```

Include dependency graph for Model.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::Model`  
*Class for model.*
  - struct `ven::Model::Vertex`
  - struct `ven::Model::Builder`

## Namespaces

- namespace `ven`

### **8.31.1 Detailed Description**

This file contains the Model class.

Definition in file [Model.hpp](#).

## 8.32 Model.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <vector>
00011 #include <unordered_map>
00012 #include <filesystem>
00013
00014 #include <glm/glm.hpp>
00015
00016 #include <assimp/scene.h>
00017
00018 #include "VEngine/Gfx/Buffer.hpp"
00019 #include "VEngine/Utils/Logger.hpp"
00020
00021
00022 namespace ven {
00023
00024 /**
00025     /// @class Model
```

```

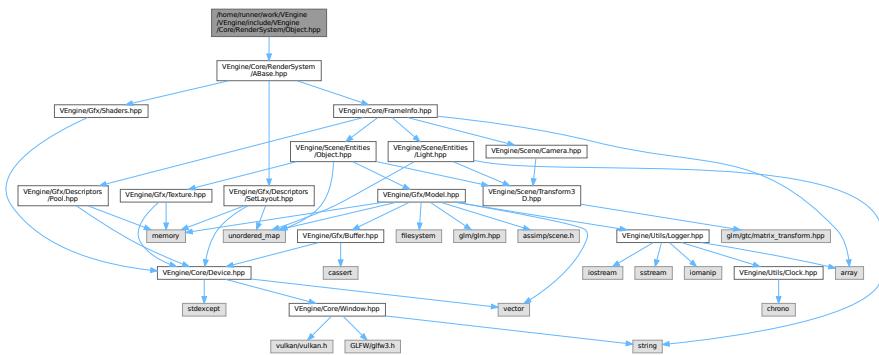
00026  /// @brief Class for model
00027  /// @namespace ven
00028  ///
00029  class Model {
00030      public:
00032      struct Vertex {
00034          glm::vec3 position{};
00035          glm::vec3 color{};
00036          glm::vec3 normal{};
00037          glm::vec2 uv{};
00038
00039          static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00040          static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00041
00042          bool operator==(const Vertex& other) const { return position == other.position &&
00043              color == other.color && normal == other.normal && uv == other.uv; }
00044      };
00045
00046      struct Builder {
00047          std::vector<Vertex> vertices;
00048          std::vector<uint32_t> indices;
00049
00050          void loadModel(const std::string &filename);
00051          void processNode(const aiNode* node, const aiScene* scene);
00052          void processMesh(const aiMesh* mesh, const aiScene* scene);
00053      };
00054
00055      Model(Device &device, const Builder &builder);
00056      ~Model() = default;
00057
00058      Model(const Model&) = delete;
00059      void operator=(const Model&) = delete;
00060
00061      void bind(VkCommandBuffer commandBuffer) const;
00062      void draw(VkCommandBuffer commandBuffer) const;
00063
00064  private:
00065
00066      void createVertexBuffer(const std::vector<Vertex>& vertices);
00067      void createIndexBuffer(const std::vector<uint32_t>& indices);
00068
00069      Device& m_device;
00070      std::unique_ptr<Buffer> m_vertexBuffer;
00071      uint32_t m_vertexCount;
00072
00073      bool m_hasIndexBuffer{false};
00074      std::unique_ptr<Buffer> m_indexBuffer;
00075      uint32_t m_indexCount;
00076
00077  }; // class Model
00078
00079 } // namespace ven

```

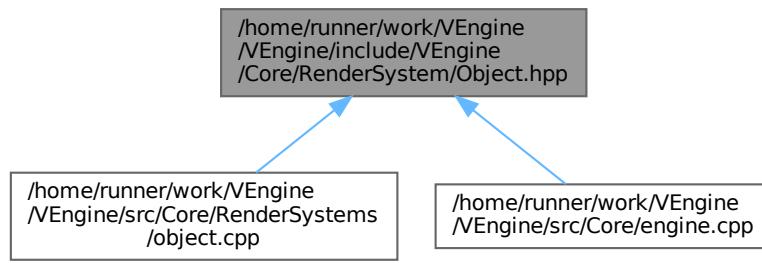
## 8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← RenderSystem/Object.hpp File Reference

This file contains the ObjectRenderSystem class.

```
#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for Object.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct `ven::ObjectPushConstantData`
  - class `ven::ObjectRenderSystem`
- Class for object render system.*

## Namespaces

- namespace `ven`

### 8.33.1 Detailed Description

This file contains the `ObjectRenderSystem` class.

Definition in file [Object.hpp](#).

## 8.34 Object.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Object.hpp
00003 /// @brief This file contains the ObjectRenderSystem class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/RenderSystem/ABase.hpp"
00010
00011 namespace ven {
00012
00013     struct ObjectPushConstantData {
00014         glm::mat4 modelMatrix{};
00015         glm::mat4 normalMatrix{};
00016     };
00017
00018 /**
00019     /// @class ObjectRenderSystem
00020     /// @brief Class for object render system
00021     /// @namespace ven
00022 /**
00023     class ObjectRenderSystem final : public ARenderSystemBase {
00024
00025     public:
00026
00027         explicit ObjectRenderSystem(Device& device, const VkRenderPass renderPass, const
00028             VkDescriptorSetLayout globalsetLayout) : ARenderSystemBase(device) {
00029             createPipelineLayout(globalssetLayout, sizeof(ObjectPushConstantData));
00030             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_shader.spv",
00031                         std::string(SHADERS_BIN_PATH) + "fragment_shader.spv", false);
00032         }
00033
00034         ObjectRenderSystem(const ObjectRenderSystem&) = delete;
00035         ObjectRenderSystem& operator=(const ObjectRenderSystem&) = delete;
00036
00037         void render(const FrameInfo &frameInfo) const override;
00038     }; // class ObjectRenderSystem
00039 } // namespace ven

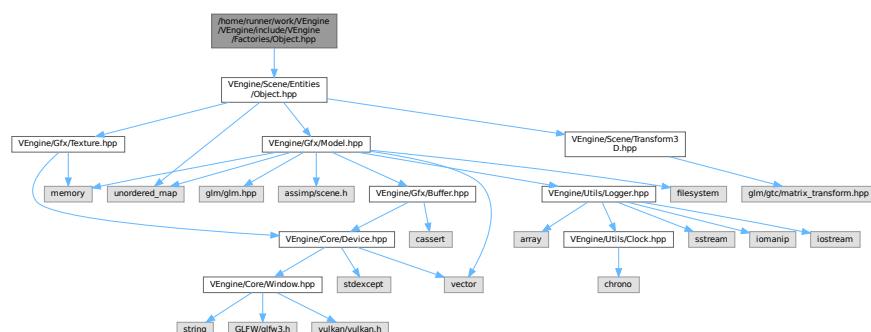
```

## 8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/← Object.hpp File Reference

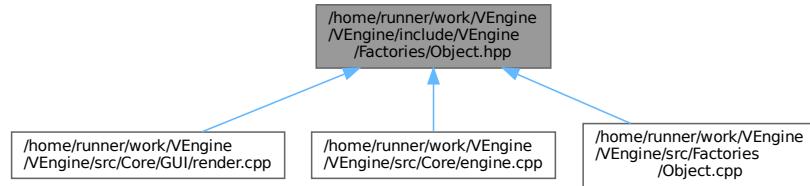
This file contains the Object Factory class.

```
#include "VEngine/Scene/Entities/Object.hpp"
```

Include dependency graph for Object.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::ObjectFactory](#)

*Class for object factory.*

## Namespaces

- namespace [ven](#)

### 8.35.1 Detailed Description

This file contains the Object Factory class.

Definition in file [Object.hpp](#).

## 8.36 Object.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Object.hpp
00003  * @brief This file contains the Object Factory class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Scene/Entities/Object.hpp"
00010
00011 namespace ven {
00012
00013 /**
00014  * @class ObjectFactory
00015  * @brief Class for object factory
00016  * @namespace ven
00017 /**
00018 class ObjectFactory {
00019
00020     public:
00021
00022         ObjectFactory() = delete;
00023         ~ObjectFactory() = default;
00024
00025         ObjectFactory(const ObjectFactory&) = delete;
00026         ObjectFactory& operator=(const ObjectFactory&) = delete;
00027         ObjectFactory(ObjectFactory&&) = delete;
00028         ObjectFactory& operator=(ObjectFactory&&) = delete;
00029
  
```

```

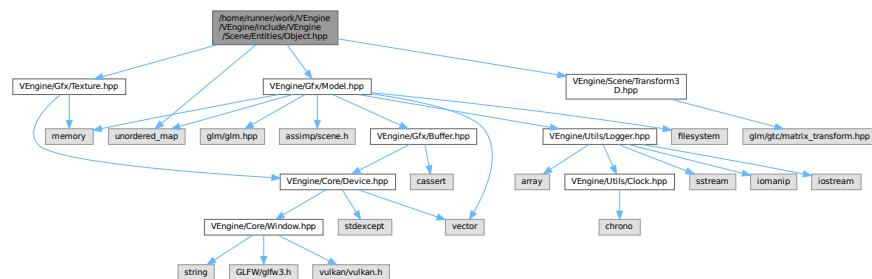
00030             static std::unique_ptr<Object> create(const std::shared_ptr<Texture>& texture, const
00031                 std::shared_ptr<Model>& model, const std::string &name, const Transform3D &transform);
00031             static std::unique_ptr<Object> duplicate(const Object& objSrc) { return
00032                 create(objSrc.getDiffuseMap(), objSrc.getModel(), objSrc.getName(), objSrc.transform); }
00032
00033     private:
00034
00035         static unsigned int m_currentObjId;
00036
00037     }; // class ObjectFactory
00038
00039 } // namespace ven

```

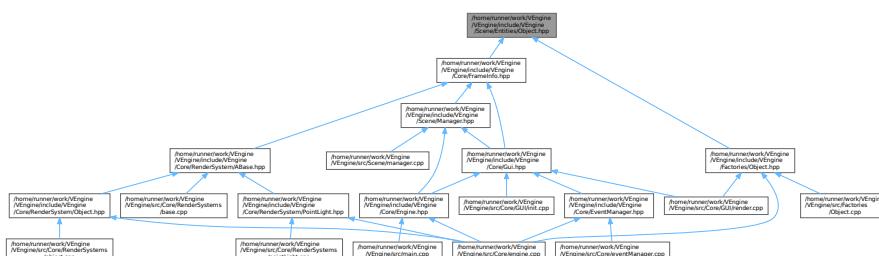
## 8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/ Entities/Object.hpp File Reference

This file contains the Object class.

```
#include <unordered_map>
#include "VEngine/Gfx/Texture.hpp"
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Object.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class **ven::Object**

*Class for object.*

## Namespaces

- namespace `ven`

## Variables

- static constexpr uint16\_t `ven::MAX_OBJECTS` = 1000

### 8.37.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

## 8.38 Object.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Object.hpp
00003 /// @brief This file contains the Object class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <unordered_map>
00010
00011 #include "VEngine/Gfx/Texture.hpp"
00012 #include "VEngine/Gfx/Model.hpp"
00013 #include "VEngine/Scene/Transform3D.hpp"
00014
00015 namespace ven {
00016
00017     static constexpr uint16_t MAX_OBJECTS = 1000;
00018
00019 /**
00020 /// @class Object
00021 /// @brief Class for object
00022 /// @namespace ven
00023 /**
00024 class Object {
00025
00026     public:
00027
00028         using Map = std::unordered_map<unsigned int, Object>;
00029
00030         explicit Object(const unsigned int objId) : m_objId{objId} {}
00031
00032         ~Object() = default;
00033
00034         Object(const Object &) = delete;
00035         Object &operator=(const Object &) = delete;
00036         Object(Object &&) = default;
00037         Object &operator=(Object &&) = default;
00038
00039         [[nodiscard]] unsigned int getId() const { return m_objId; }
00040         [[nodiscard]] std::string getName() const { return m_name; }
00041         [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }
00042         [[nodiscard]] std::shared_ptr<Texture> getDiffuseMap() const { return m_diffuseMap; }
00043         [[nodiscard]] VkDescriptorBufferInfo getBufferInfo(const int frameIndex) const { return
00044             m_bufferInfo.at(frameIndex); }
00045         void setModel(const std::shared_ptr<Model> &model) { m_model = model; }
00046         void setDiffuseMap(const std::shared_ptr<Texture> &diffuseMap) { m_diffuseMap =
00047             diffuseMap; }
00048         void setName(const std::string &name) { m_name = name; }
00049         void setBufferInfo(const int frameIndex, const VkDescriptorBufferInfo& info) {
00050             m_bufferInfo[frameIndex] = info; }
00051
00052     private:

```

```

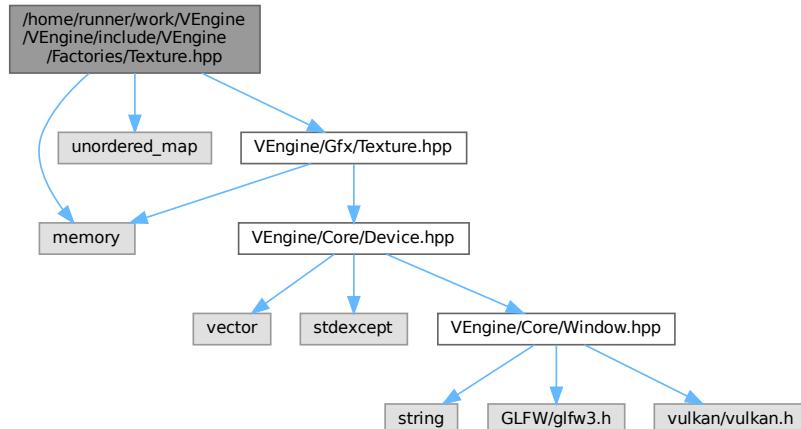
00052
00053     unsigned int m_objid;
00054     std::string m_name;
00055     std::shared_ptr<Model> m_model = nullptr;
00056     std::shared_ptr<Texture> m_diffuseMap = nullptr;
00057     std::unordered_map<int, VkDescriptorBufferInfo> m_bufferInfo;
00058
00059 }; // class Object
00060
00061 } // namespace ven

```

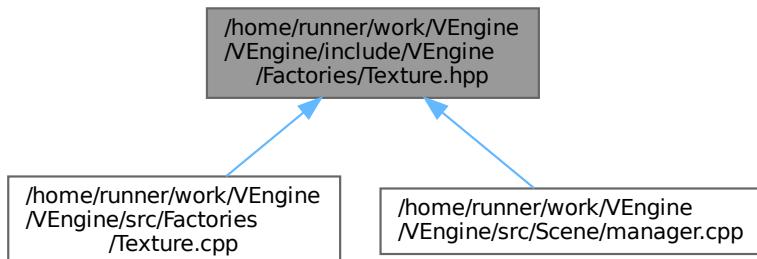
## 8.39 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/← Texture.hpp File Reference

This file contains the Texture Factory class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Gfx/Texture.hpp"
Include dependency graph for Texture.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::TextureFactory](#)

*Class for Texture factory.*

## Namespaces

- namespace [ven](#)

### 8.39.1 Detailed Description

This file contains the Texture Factory class.

Definition in file [Texture.hpp](#).

## 8.40 Texture.hpp

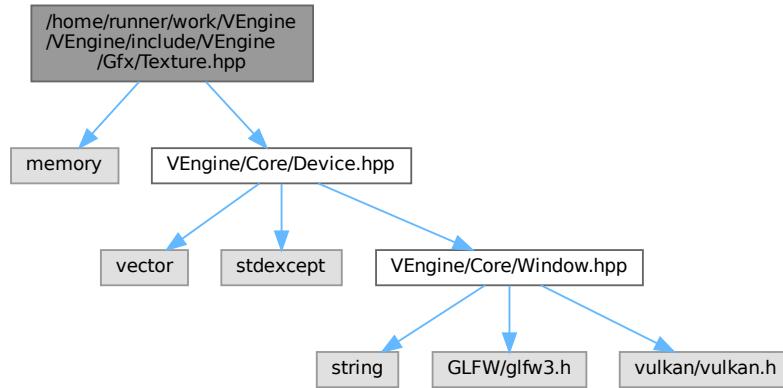
[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Texture.hpp
00003 /// @brief This file contains the Texture Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Gfx/Texture.hpp"
00013
00014 namespace ven {
00015
00016 /**
00017 /// @class TextureFactory
00018 /// @brief Class for Texture factory
00019 /// @namespace ven
00020 /**
00021 class TextureFactory {
00022
00023     public:
00024
00025         TextureFactory() = delete;
00026         ~TextureFactory() = default;
00027
00028         TextureFactory(const TextureFactory&) = delete;
00029         TextureFactory& operator=(const TextureFactory&) = delete;
00030         TextureFactory(TextureFactory&&) = delete;
00031         TextureFactory& operator=(TextureFactory&&) = delete;
00032
00033         static std::unique_ptr<Texture> create(Device& device, const std::string& filepath) {
00034             return std::make_unique<Texture>(device, filepath);
00035             static std::unordered_map<std::string, std::shared_ptr<Texture>> loadAll(Device& device,
00036             const std::string& folderPath);
00037
00038     }; // class TextureFactory
00039 } // namespace ven
```

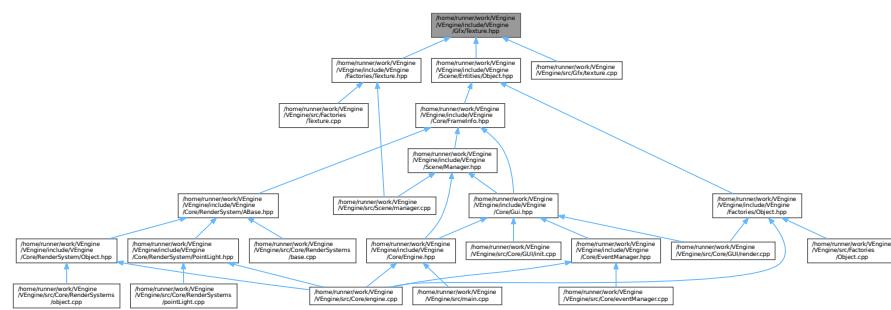
## 8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp File Reference

This file contains the Texture class.

```
#include <memory>
#include "VEngine/Core/Device.hpp"
Include dependency graph for Texture.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class `ven::Texture`

*Class for texture.*

### Namespaces

- namespace `ven`

### 8.41.1 Detailed Description

This file contains the Texture class.

Definition in file [Texture.hpp](#).

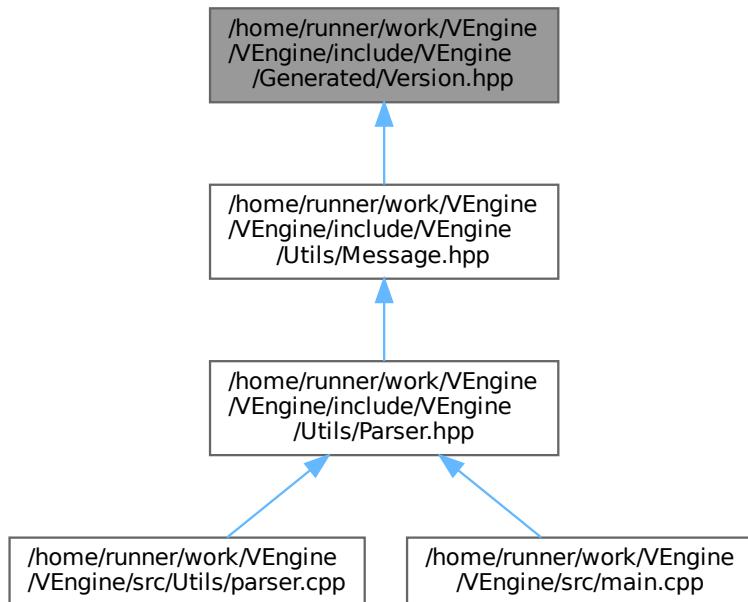
## 8.42 Texture.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002  * @file Texture.hpp
00003  * @brief This file contains the Texture class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016  * @class Texture
00017  * @brief Class for texture
00018  * @namespace ven
00019 /**
00020 class Texture {
00021
00022     public:
00023
00024         Texture(Device &device, const std::string &textureFilepath);
00025         Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00026                 VkSampleCountFlagBits sampleCount);
00027         ~Texture();
00028
00029         Texture(const Texture &) = delete;
00030         Texture &operator=(const Texture &) = delete;
00031
00032         void updateDescriptor();
00033         void transitionLayout(VkCommandBuffer commandBuffer, VkImageLayout oldLayout,
00034                               VkImageLayout newLayout) const;
00035
00036         [[nodiscard]] VkImageView imageView() const { return m_textureImageView; }
00037         [[nodiscard]] VkSampler sampler() const { return m_textureSampler; }
00038         [[nodiscard]] VkImage getImage() const { return m_textureImage; }
00039         [[nodiscard]] VkImageView getImageView() const { return m_textureImageView; }
00040         [[nodiscard]] VkDescriptorImageInfo getImageInfo() const { return m_descriptor; }
00041         [[nodiscard]] VkImageLayout getImageLayout() const { return m_textureLayout; }
00042         [[nodiscard]] VkExtent3D getExtent() const { return m_extent; }
00043         [[nodiscard]] VkFormat getFormat() const { return m_format; }
00044
00045     private:
00046
00047         void createTextureImage(const std::string &filepath);
00048         void createTextureImageView(VkImageViewType viewType);
00049         void createTextureSampler();
00050
00051         VkDescriptorImageInfo m_descriptor{};
00052         Device &m_device;
00053         VkImage m_textureImage = nullptr;
00054         VkDeviceMemory m_textureImageMemory = nullptr;
00055         VkImageView m_textureImageView = nullptr;
00056         VkSampler m_textureSampler = nullptr;
00057         VkFormat m_format;
00058         VkImageLayout m_textureLayout{};
00059         uint32_t m_mipLevels{1};
00060         uint32_t m_layerCount{1};
00061         VkExtent3D m_extent{};
00062
00063 } // namespace ven
```

## 8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Generated/Version.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define PROJECT_VERSION_MAJOR 0`
- `#define PROJECT_VERSION_MINOR 0`
- `#define PROJECT_VERSION_PATCH 1`
- `#define PROJECT_VERSION "0.0.1"`

### 8.43.1 Macro Definition Documentation

#### 8.43.1.1 PROJECT\_VERSION

```
#define PROJECT_VERSION "0.0.1"
```

Definition at line 10 of file [Version.hpp](#).

#### 8.43.1.2 PROJECT\_VERSION\_MAJOR

```
#define PROJECT_VERSION_MAJOR 0
```

Definition at line 7 of file [Version.hpp](#).

### 8.43.1.3 PROJECT\_VERSION\_MINOR

```
#define PROJECT_VERSION_MINOR 0
```

Definition at line 8 of file Version.hpp.

### 8.43.1.4 PROJECT\_VERSION\_PATCH

```
#define PROJECT_VERSION_PATCH 1
```

Definition at line 9 of file Version.hpp.

## 8.44 Version.hpp

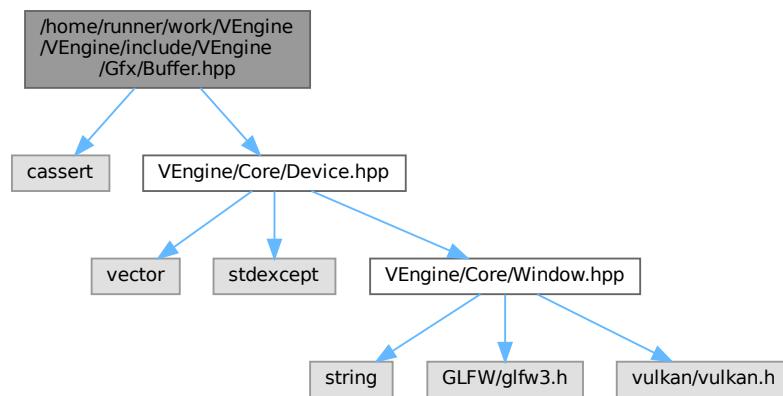
[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 // =====
00004 // DO NOT EDIT THIS FILE MANUALLY. IT IS GENERATED BY CMAKE DURING THE BUILD PROCESS.
00005 // =====
00006
00007 #define PROJECT_VERSION_MAJOR 0
00008 #define PROJECT_VERSION_MINOR 0
00009 #define PROJECT_VERSION_PATCH 1
00010 #define PROJECT_VERSION "0.0.1"
```

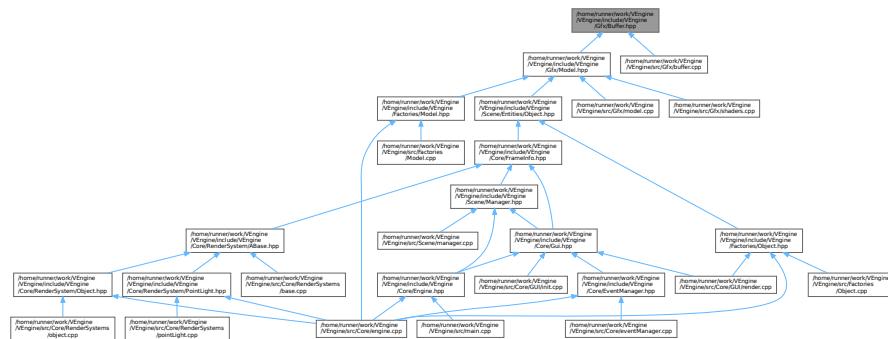
## 8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Buffer.hpp File Reference

This file contains the Buffer class.

```
#include <cassert>
#include "VEngine/Core/Device.hpp"
Include dependency graph for Buffer.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Buffer](#)

*Class for buffer.*

## Namespaces

- namespace [ven](#)

### 8.45.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

## 8.46 Buffer.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Buffer.hpp
00003 /// @brief This file contains the Buffer class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <cassert>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 /// @class Buffer
00017 /// @brief Class for buffer
00018 /// @namespace ven
00019 /**
00020 class Buffer {
00021
00022     public:
00023
00024         Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
00025             VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
00026             minOffsetAlignment = 1);
  
```

```

00025         ~Buffer();
00026
00027         Buffer(const Buffer&) = delete;
00028         Buffer& operator=(const Buffer&) = delete;
00029
00030         /**
00031         /// @brief Map a memory range of this buffer. If successful, mapped points to the
00032         /// specified buffer range.
00033         /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the
00034         /// complete buffer range.
00035         /// @param offset (Optional) Byte offset from beginning
00036         /// @return VkResult of the buffer mapping call
00037         ///
00038         VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00039
00040         /**
00041         /// @brief Unmap a mapped memory range
00042         ///
00043         /// @note Does not return a result as vkUnmapMemory can't fail
00044         ///
00045         void unmap();
00046
00047         /**
00048         /// @brief Copies the specified data to the mapped buffer. Default value writes whole
00049         /// buffer range
00050         /// @param data Pointer to the data to copy
00051         /// @param size (Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the
00052         /// complete buffer range.
00053         /// @param offset (Optional) Byte offset from beginning of mapped region
00054         void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
00055         offset = 0) const;
00056
00057         /**
00058         /// @brief Flush a memory range of the buffer to make it visible to the device
00059         /// @note Only required for non-coherent memory
00060         ///
00061         /// @param size (Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush
00062         /// the complete buffer range.
00063         /// @param offset (Optional) Byte offset from beginning
00064         /// @return VkResult of the flush call
00065         ///
00066         VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0) const;
00067
00068         /**
00069         /// @brief Create a buffer info descriptor
00070         ///
00071         /// @param size (Optional) Size of the memory range of the descriptor
00072         /// @param offset (Optional) Byte offset from beginning
00073         ///
00074         /// @return VkDescriptorBufferInfo of specified offset and range
00075         ///
00076         [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size =
00077         VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset,
00078         size, }; }
00079
00080         /**
00081         /// @brief Invalidate a memory range of the buffer to make it visible to the host
00082         /// @note Only required for non-coherent memory
00083         /// @param size (Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to
00084         /// invalidate the complete buffer range.
00085         /// @param offset (Optional) Byte offset from beginning
00086         /// @return VkResult of the invalidate call
00087         ///
00088         [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset =
00089         0) const;
00090
00091         /**
00092         /// Copies "instanceSize" bytes of data to the mapped buffer at an offset of index *
00093         alignmentSize
00094         /// @param data Pointer to the data to copy
00095         /// @param index Used in offset calculation
00096         ///
00097         /**
00098         void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
00099         m_instanceSize, index * m_alignmentSize); }

```

```

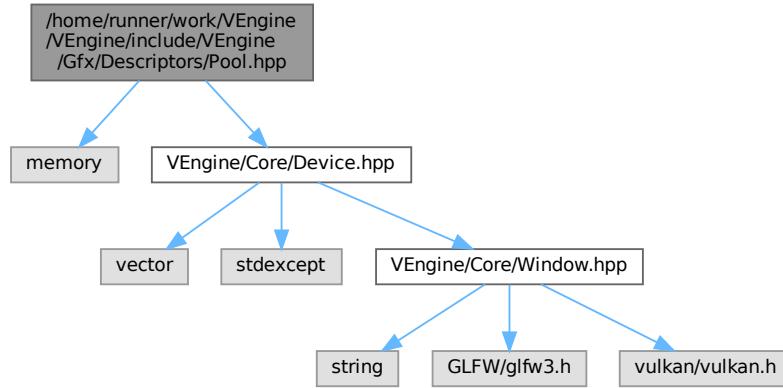
00100      /**
00101      /// Flush the memory range at index * alignmentSize of the buffer to make it visible to
00102      /// the device
00103      /// @param index Used in offset calculation
00104      ///
00105      VkResult flushIndex(const VkDeviceSize index) const { assert(m_alignmentSize %
00106      m_device.getProperties().limits.nonCoherentAtomSize == 0 && "Cannot use LveBuffer::flushIndex if
00107      alignmentSize isn't a multiple of Device Limits nonCoherentAtomSize"); return flush(m_alignmentSize,
00108      index * m_alignmentSize); }
00109      /**
00110      /**
00111      /// Create a buffer info descriptor
00112      /**
00113      /// @param index Specifies the region given by index * alignmentSize
00114      /**
00115      /// @return VkDescriptorBufferInfo for instance at index
00116      /**
00117      [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index)
00118      const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }
00119      /**
00120      /**
00121      /// Invalidate a memory range of the buffer to make it visible to the host
00122      /**
00123      /// @note Only required for non-coherent memory
00124      /**
00125      /// @param index Specifies the region to invalidate: index * alignmentSize
00126      /**
00127      [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return
00128      invalidate(m_alignmentSize, index * m_alignmentSize); }
00129      /**
00130      [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
00131      /**
00132      [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
00133      /**
00134      [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
00135      /**
00136      [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }
00137      /**
00138      [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_alignmentSize; }
00139      /**
00140      [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00141      /**
00142      [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
00143      m_memoryPropertyFlags; }
00144      /**
00145      [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }
00146      /**
00147      private:
00148      /**
00149      /**
00150      /**
00151      /**
00152      /**
00153      /**
00154      /**
00155      /**
00156      /**
00157      /**
00158      /**
00159      /**
00160      /**
00161      /**
00162      /**
00163  } // namespace ven

```

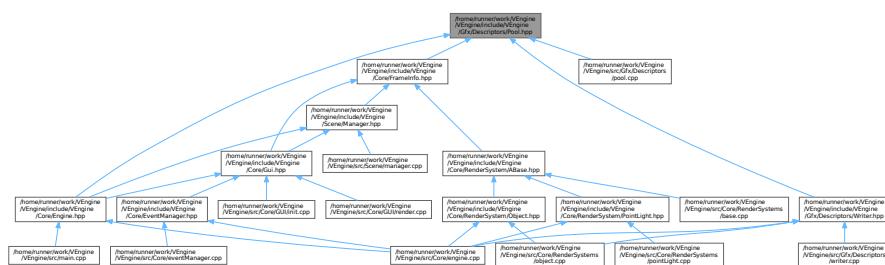
## 8.47 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp File Reference

This file contains the DescriptorPool class.

```
#include <memory>
#include "VEngine/Core/Device.hpp"
Include dependency graph for Pool.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::DescriptorPool`  
*Class for descriptor pool.*
- class `ven::DescriptorPool::Builder`

## Namespaces

- namespace `ven`

## Variables

- static constexpr uint32\_t `ven::DEFAULT_MAX_SETS` = 1000

### 8.47.1 Detailed Description

This file contains the `DescriptorPool` class.

Definition in file [Pool.hpp](#).

## 8.48 Pool.hpp

[Go to the documentation of this file.](#)

```

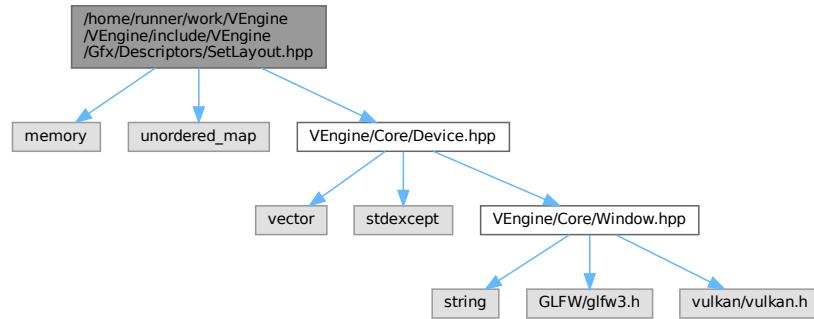
00001 /**
00002 /// @file Pool.hpp
00003 /// @brief This file contains the DescriptorPool class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr uint32_t DEFAULT_MAX_SETS = 1000;
00016
00017 /**
00018     /// @class DescriptorPool
00019     /// @brief Class for descriptor pool
00020     /// @namespace ven
00021 /**
00022     class DescriptorPool {
00023
00024         public:
00025
00026             class Builder {
00027
00028                 public:
00029
00030                     explicit Builder(Device &device) : m_device{device} {}
00031
00032                     [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
00033                         std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00034
00035                     Builder &addPoolSize(const VkDescriptorType descriptorType, const uint32_t count)
00036                     { m_poolSizes.push_back({descriptorType, count}); return *this; }
00037                     Builder &setPoolFlags(const VkDescriptorPoolCreateFlags flags) { m_poolFlags =
00038                         flags; return *this; }
00039                     Builder &setMaxSets(const uint32_t count) { m_maxSets = count; return *this; }
00040
00041             private:
00042                 Device &m_device;
00043                 std::vector<VkDescriptorPoolSize> m_poolSizes;
00044                 uint32_t m_maxSets{DEFAULT_MAX_SETS};
00045                 VkDescriptorPoolCreateFlags m_poolFlags{0};
00046
00047         }; // class Builder
00048
00049         DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
00050             const std::vector<VkDescriptorPoolSize> &poolSizes);
00051         ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
00052     }
00053
00054         DescriptorPool(const DescriptorPool &) = delete;
00055         DescriptorPool &operator=(const DescriptorPool &) = delete;
00056
00057         bool allocateDescriptor(VkDescriptorSetLayout descriptorsetLayout, VkDescriptorSet
00058             &descriptor) const;
00059         void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
00060             vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
00061             descriptors.data()); }
00062         void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00063
00064         [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }
00065
00066     private:
00067         Device &m_device;
00068         VkDescriptorPool m_descriptorPool;
00069         friend class DescriptorWriter;
00070     }; // class DescriptorPool
00071 } // namespace ven

```

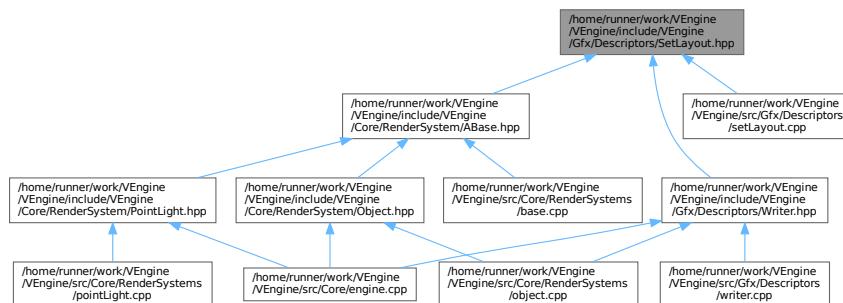
## 8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp File Reference

This file contains the DescriptorsetLayout class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Core/Device.hpp"
Include dependency graph for SetLayout.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [ven::DescriptorsetLayout](#)  
*Class for descriptor set layout.*
- class [ven::DescriptorsetLayout::Builder](#)

### Namespaces

- namespace [ven](#)

### 8.49.1 Detailed Description

This file contains the DescriptorSetLayout class.

Definition in file [SetLayout.hpp](#).

## 8.50 SetLayout.hpp

[Go to the documentation of this file.](#)

```

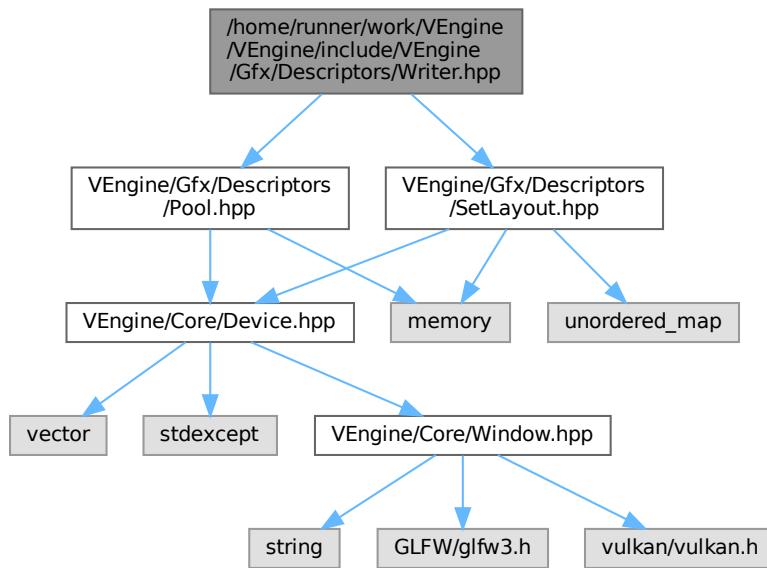
00001 /**
00002 /// @file SetLayout.hpp
00003 /// @brief This file contains the DescriptorsetLayout class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Core/Device.hpp"
00013
00014 namespace ven {
00015
00016 /**
00017 /// @class DescriptorsetLayout
00018 /// @brief Class for descriptor set layout
00019 /// @namespace ven
00020 /**
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,
00032                         VkShaderStageFlags stageFlags, uint32_t count = 1);
00033                     std::unique_ptr<DescriptorsetLayout> build() const { return
00034                         std::make_unique<DescriptorsetLayout>(m_device, m_bindings); }
00035
00036             private:
00037
00038                 Device &m_device;
00039                 std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00040
00041             }; // class Builder
00042
00043             DescriptorsetLayout(Device &device, const std::unordered_map<uint32_t,
00044                 VkDescriptorsetLayoutBinding>& bindings);
00045             ~DescriptorsetLayout() { vkDestroyDescriptorsetLayout(m_device.device(),
00046                 m_descriptorsLayout, nullptr); }
00047
00048             VkDescriptorsetLayout getDescriptorsetLayout() const { return m_descriptorsLayout; }
00049
00050             private:
00051
00052                 Device &m_device;
00053                 VkDescriptorsetLayout m_descriptorsLayout;
00054                 std::unordered_map<uint32_t, VkDescriptorsetLayoutBinding> m_bindings;
00055
00056             friend class DescriptorWriter;
00057
00058     }; // class DescriptorsetLayout
00059 } // namespace ven

```

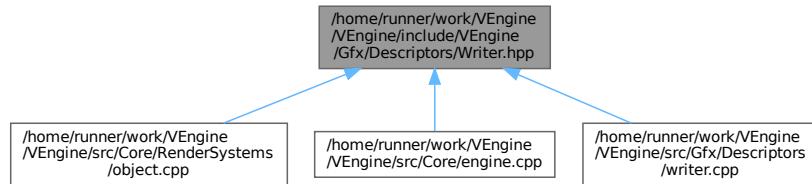
## 8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp File Reference

This file contains the DescriptorsWriter class.

```
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
Include dependency graph for Writer.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class `ven::DescriptorWriter`  
*Class for descriptor writer.*

### Namespaces

- namespace `ven`

### 8.51.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file [Writer.hpp](#).

## 8.52 Writer.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Writer.hpp
00003 /// @brief This file contains the DescriptorsWriter class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00010 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00011
00012 namespace ven {
00013
00014 /**
00015 /// @class DescriptorWriter
00016 /// @brief Class for descriptor writer
00017 /// @namespace ven
00018 /**
00019 class DescriptorWriter {
00020
00021     public:
00022
00023         DescriptorWriter(DescriptorsetLayout &setLayout, DescriptorPool &pool) :
00024             m_setLayout{setLayout}, m_pool{pool} {}
00025         ~DescriptorWriter() = default;
00026
00027         DescriptorWriter(const DescriptorWriter &) = delete;
00028         DescriptorWriter &operator=(const DescriptorWriter &) = delete;
00029
00030         DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00031         DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00032
00033         bool build(VkDescriptorSet &set);
00034         void overwrite(const VkDescriptorSet &set);
00035
00036     private:
00037
00038         DescriptorsetLayout &m_setLayout;
00039         DescriptorPool &m_pool;
00040         std::vector<VkWriteDescriptorSet> m_writes;
00041     }; // class DescriptorWriter
00042
00043 } // namespace ven

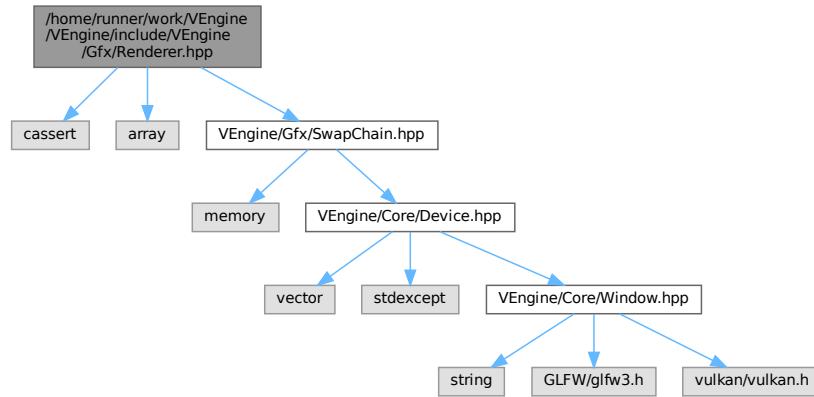
```

## 8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/↔ Renderer.hpp File Reference

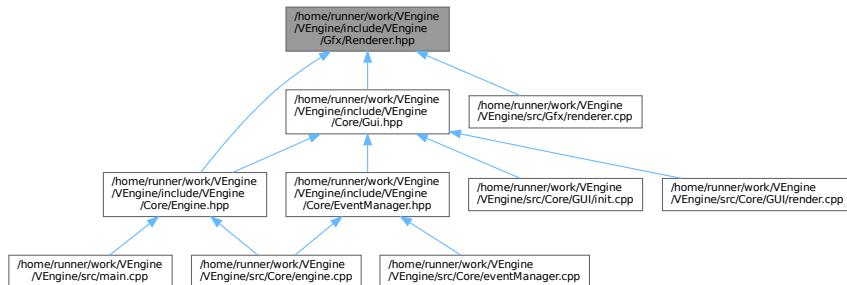
This file contains the Renderer class.

```
#include <cassert>
#include <array>
```

```
#include "VEngine/Gfx/SwapChain.hpp"
Include dependency graph for Renderer.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::Renderer`

*Class for renderer.*

## Namespaces

- namespace `ven`

## Variables

- static constexpr `VkClearColorValue` `ven::DEFAULT_CLEAR_COLOR` = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr `VkClearDepthStencilValue` `ven::DEFAULT_CLEAR_DEPTH` = {1.0F, 0}

### 8.53.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

## 8.54 Renderer.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Renderer.hpp
00003  * @brief This file contains the Renderer class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <cassert>
00010 #include <array>
00011
00012 #include "VEngine/Gfx/SwapChain.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}};
00017     static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0};
00018
00019 /**
00020  * @class Renderer
00021  * @brief Class for renderer
00022  * @namespace ven
00023 /**
00024  class Renderer {
00025
00026     public:
00027
00028         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
00029             recreateSwapChain(); createCommandBuffers(); }
00030         ~Renderer() { freeCommandBuffers(); }
00031
00032         Renderer(const Renderer &) = delete;
00033         Renderer& operator=(const Renderer &) = delete;
00034
00035         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
00036             m_swapChain->getRenderPass(); }
00037         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00038         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00039         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress())
00040             && "cannot get command m_buffer when frame not in progress"; return
00041             m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00042
00043         [[nodiscard]] unsigned long getFrameIndex() const { assert(isFrameInProgress() && "cannot
00044             get frame index when frame not in progress"); return m_currentFrameIndex; }
00045         [[nodiscard]] std::array<float, 4> getClearColor() const { return
00046             m_clearValues[0].color.float32[0],
00047             m_clearValues[0].color.float32[1],
00048             m_clearValues[0].color.float32[2],
00049             m_clearValues[0].color.float32[3]
00050         ; }
00051
00052         [[nodiscard]] Window& getWindow() const { return m_window; }
00053
00054         void setClearColor(const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR, const
00055             VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
00056                 clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00057         VkCommandBuffer beginFrame();
00058         void endFrame();
00059         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00060         void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00061
00062     private:
00063
00064         void createCommandBuffers();
00065         void freeCommandBuffers();
00066         void recreateSwapChain();
00067
00068         Window &m_window;
00069         Device &m_device;

```

```

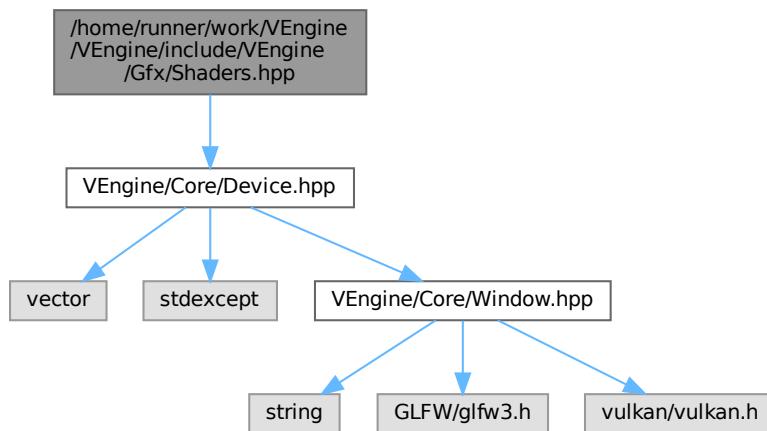
00063     std::unique_ptr<SwapChain> m_swapChain;
00064     std::vector<VkCommandBuffer> m_commandBuffers;
00065     std::array<VkClearValue, 2> m_clearValues{DEFAULT_CLEAR_COLOR, 1.0F, 0.F};
00066
00067     uint32_t m_currentImageIndex{0};
00068     unsigned long m_currentFrameIndex{0};
00069     bool m_isFrameStarted{false};
00070
00071 }; // class Renderer
00072
00073 } // namespace ven

```

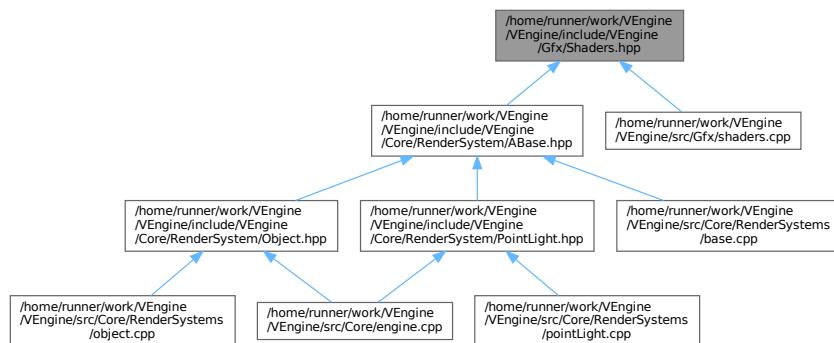
## 8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp File Reference

This file contains the Shader class.

```
#include "VEngine/Core/Device.hpp"
Include dependency graph for Shaders.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [ven::PipelineConfigInfo](#)
- class [ven::Shaders](#)

*Class for shaders.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr std::string\_view [ven::SHADERS\\_BIN\\_PATH](#) = "build/shaders/"

### 8.55.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

## 8.56 Shaders.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Shaders.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/Device.hpp"
00010
00011 namespace ven {
00012
00013     static constexpr std::string_view SHADERS_BIN_PATH = "build/shaders/";
00014
00015     struct PipelineConfigInfo {
00016         PipelineConfigInfo() = default;
00017         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00018         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00019
00020         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00021         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00022         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00023         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00024         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00025         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00026         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00027         VKPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00028         std::vector<VkDynamicState> dynamicStateEnables;
00029         VKPipelineDynamicStateCreateInfo dynamicStateInfo{};
00030         VKPipelineLayout pipelineLayout = nullptr;
00031         VkRenderPass renderPass = nullptr;
00032         uint32_t subpass = 0;
00033     };
00034
00035 /**
00036 /// @class Shaders
00037 /// @brief Class for shaders
00038 /// @namespace ven
00039 /**
00040 class Shaders {
00041     public:
```

```

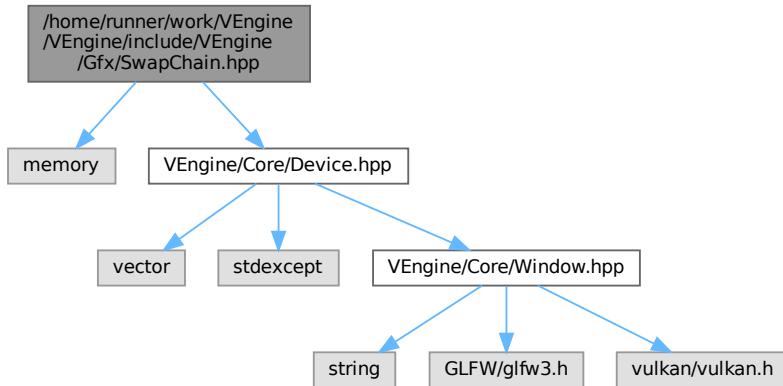
00043
00044     Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
00045     const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
00046     fragFilepath, configInfo); }
00047     ~Shaders();
00048     Shaders(const Shaders&) = delete;
00049     Shaders& operator=(const Shaders&) = delete;
00050     static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00051     void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
00052         VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00053     private:
00054     static std::vector<char> readFile(const std::string &filename);
00055     void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
00056     fragFilepath, const PipelineConfigInfo& configInfo);
00057     void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
00058     const;
00059     Device& m_device;
00060     VkPipeline m_graphicsPipeline{nullptr};
00061     VkShaderModule m_vertShaderModule{nullptr};
00062     VkShaderModule m_fragShaderModule{nullptr};
00063     }; // class Shaders
00064 } // namespace ven

```

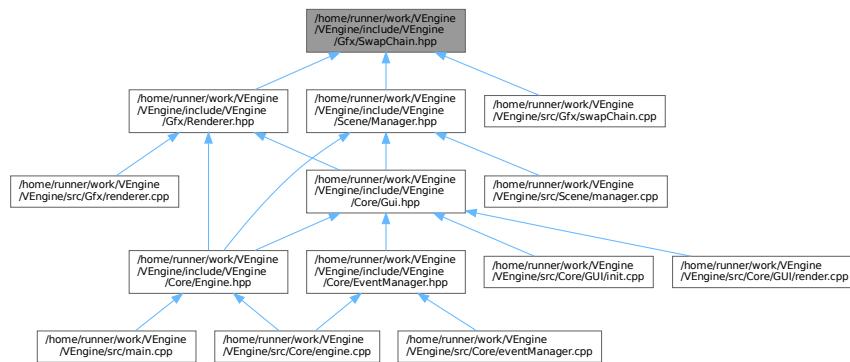
## 8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp File Reference

This file contains the Shader class.

```
#include <memory>
#include "VEngine/Core/Device.hpp"
Include dependency graph for SwapChain.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::SwapChain](#)  
*Class for swap chain.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr int [ven::MAX\\_FRAMES\\_IN\\_FLIGHT](#) = 2

### 8.57.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

## 8.58 SwapChain.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file SwapChain.hpp
00003  * @brief This file contains the Shader class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00016

```

```

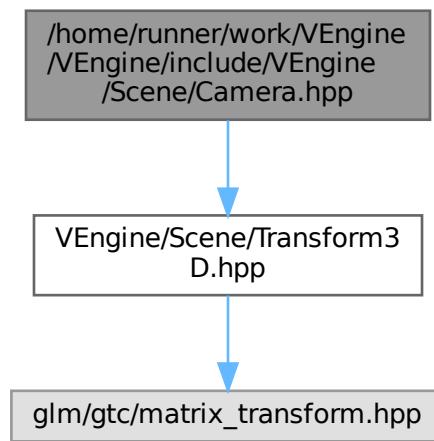
00017  /**
00018  * @class SwapChain
00019  * @brief Class for swap chain
00020  * @namespace ven
00021  */
00022  class SwapChain {
00023
00024     public:
00025
00026         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : m_device{deviceRef},
00027             m_windowExtent{windowExtentRef} { init(); }
00028         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
00029             previous) : m_device{deviceRef}, m_windowExtent{windowExtentRef}, m_oldSwapChain{std::move(previous)}
00030             { init(); m_oldSwapChain = nullptr; }
00031             ~SwapChain();
00032
00033         SwapChain(const SwapChain &) = delete;
00034         SwapChain& operator=(const SwapChain &) = delete;
00035
00036         [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return
00037             m_swapChainFrameBuffers[index]; }
00038         [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }
00039         [[nodiscard]] VkImageView getImageView(const int index) const { return
00040             m_swapChainImageViews[static_cast<unsigned long>(index)]; }
00041         [[nodiscard]] size_t getCount() const { return m_swapChainImages.size(); }
00042         [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_swapChainImageFormat; }
00043         [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00044         [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
00045         [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00046
00047         [[nodiscard]] float extentAspectRatio() const { return
00048             static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
00049         [[nodiscard]] VkFormat findDepthFormat() const;
00050
00051         VkResult acquireNextImage(uint32_t *imageIndex) const;
00052         VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00053
00054         [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return
00055             m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat ==
00056             swapChain.m_swapChainDepthFormat; }
00057
00058     private:
00059
00060         void init();
00061         void createSwapChain();
00062         void createImageViews();
00063         void createDepthResources();
00064         void createRenderPass();
00065         void createFrameBuffers();
00066         void createSyncObjects();
00067
00068         static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
00069             &availableFormats);
00070         static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
00071             &availablePresentModes);
00072         [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
00073             const;
00074
00075         VkFormat m_swapChainImageFormat{};
00076         VkFormat m_swapChainDepthFormat{};
00077         VkExtent2D m_swapChainExtent{};
00078
00079         std::vector<VkFramebuffer> m_swapChainFrameBuffers;
00080         VkRenderPass m_renderPass{};
00081
00082         std::vector<VkImage> m_depthImages;
00083         std::vector<VkDeviceMemory> m_depthImageMemory;
00084         std::vector<VkImageView> m_depthImageViews;
00085         std::vector<VkImage> m_swapChainImages;
00086         std::vector<VkImageView> m_swapChainImageViews;
00087
00088         Device &m_device;
00089         VkExtent2D m_windowExtent;
00090
00091         VkSwapchainKHR m_swapChain{};
00092         std::shared_ptr<SwapChain> m_oldSwapChain;
00093
00094         std::vector<VkSemaphore> m_imageAvailableSemaphores;
00095         std::vector<VkSemaphore> m_renderFinishedSemaphores;
00096         std::vector<VkFence> m_inFlightFences;
00097         std::vector<VkFence> m_imagesInFlight;
00098         size_t m_currentFrame{0};
00099
00100     }; // class SwapChain
00101 } // namespace ven

```

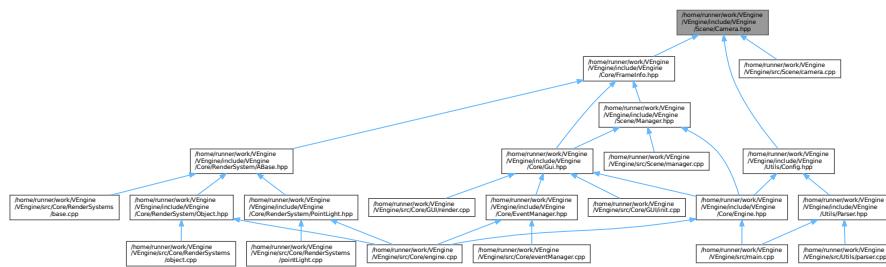
## 8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/← Camera.hpp File Reference

This file contains the Camera class.

```
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Camera.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [ven::Camera](#)

*Class for camera.*

### Namespaces

- namespace [ven](#)

## Variables

- static constexpr glm::vec3 **ven::DEFAULT\_POSITION** {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 **ven::DEFAULT\_ROTATION** {0.F, 0.F, 0.F}
- static constexpr float **ven::DEFAULT\_FOV** = glm::radians(50.0F)
- static constexpr float **ven::DEFAULT\_NEAR** = 0.1F
- static constexpr float **ven::DEFAULT\_FAR** = 100.F
- static constexpr float **ven::DEFAULT\_MOVE\_SPEED** = 3.F
- static constexpr float **ven::DEFAULT\_LOOK\_SPEED** = 1.5F

### 8.59.1 Detailed Description

This file contains the Camera class.

Definition in file [Camera.hpp](#).

## 8.60 Camera.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Camera.hpp
00003 /// @brief This file contains the Camera class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Scene/Transform3D.hpp"
00010
00011 namespace ven {
00012
00013     static constexpr glm::vec3 DEFAULT_POSITION{0.F, 0.F, -2.5F};
00014     static constexpr glm::vec3 DEFAULT_ROTATION{0.F, 0.F, 0.F};
00015
00016     static constexpr float DEFAULT_FOV = glm::radians(50.0F);
00017     static constexpr float DEFAULT_NEAR = 0.1F;
00018     static constexpr float DEFAULT_FAR = 100.F;
00019
00020     static constexpr float DEFAULT_MOVE_SPEED = 3.F;
00021     static constexpr float DEFAULT_LOOK_SPEED = 1.5F;
00022
00023 /**
00024 /// @class Camera
00025 /// @brief Class for camera
00026 /// @namespace ven
00027 /**
00028 class Camera {
00029
00030     public:
00031
00032         Camera(const float fov, const float near, const float far, const float moveSpeed, const
00033             float lookSpeed) : m_fov(fov), m_near(near), m_far(far), m_moveSpeed(moveSpeed),
00034             m_lookSpeed(lookSpeed) { }
00035         ~Camera() = default;
00036
00037         Camera(const Camera&) = delete;
00038         Camera& operator=(const Camera&) = delete;
00039
00040         void setOrthographicProjection(float left, float right, float top, float bottom, float
00041             near, float far);
00042         void setPerspectiveProjection(float aspect);
00043         void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up = {0.F, -1.F,
00044             0.F});
00045         void setViewTarget(const glm::vec3 position, const glm::vec3 target, const glm::vec3 up =
00046             {0.F, -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00047         void setViewXYZ(glm::vec3 position, glm::vec3 rotation);
00048         void setFov(const float fov) { m_fov = fov; }
00049         void setNear(const float near) { m_near = near; }
00050         void setFar(const float far) { m_far = far; }
00051         void setMoveSpeed(const float moveSpeed) { m_moveSpeed = moveSpeed; }
00052         void setLookSpeed(const float lookSpeed) { m_lookSpeed = lookSpeed; }

```

```

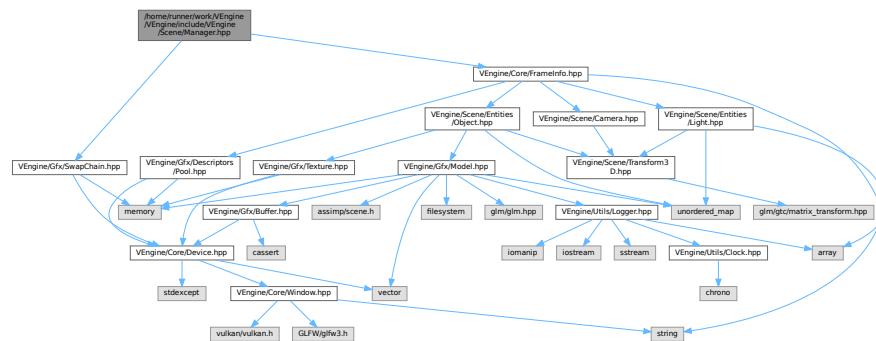
00048
00049     [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00050     [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00051     [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00052     float getFov() const { return m_fov; }
00053     float getNear() const { return m_near; }
00054     float getFar() const { return m_far; }
00055     float getMoveSpeed() const { return m_moveSpeed; }
00056     float getLookSpeed() const { return m_lookSpeed; }
00057
00058     Transform3D transform{DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION};
00059
00060     private:
00061
00062     float m_fov;
00063     float m_near;
00064     float m_far;
00065     float m_moveSpeed;
00066     float m_lookSpeed;
00067     glm::mat4 m_projectionMatrix{1.F};
00068     glm::mat4 m_viewMatrix{1.F};
00069     glm::mat4 m_inverseViewMatrix{1.F};
00070
00071 }; // class Camera
00072
00073 } // namespace ven

```

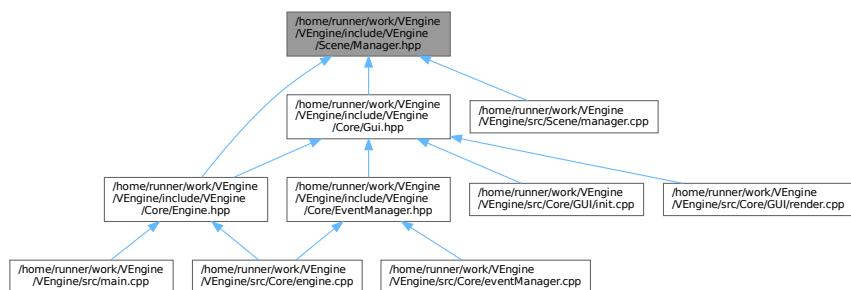
## 8.61 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp File Reference

This file contains the SceneManager class.

```
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Gfx/SwapChain.hpp"
Include dependency graph for Manager.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::SceneManager](#)

*Class for object manager.*

## Namespaces

- namespace [ven](#)

### 8.61.1 Detailed Description

This file contains the SceneManager class.

Definition in file [Manager.hpp](#).

## 8.62 Manager.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002  /// @file Manager.hpp
00003  /// @brief This file contains the SceneManager class
00004  /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/FrameInfo.hpp"
00010 #include "VEngine/Gfx/SwapChain.hpp"
00011
00012 namespace ven {
00013
00014 /**
00015  /// @class SceneManager
00016  /// @brief Class for object manager
00017  /// @namespace ven
00018 /**
00019 class SceneManager {
00020
00021     public:
00022
00023     explicit SceneManager(Device &device);
00024
00025     SceneManager(const SceneManager &) = delete;
00026     SceneManager &operator=(const SceneManager &) = delete;
00027     SceneManager(SceneManager &&) = delete;
00028     SceneManager &operator=(SceneManager &&) = delete;
00029
00030     void addObject(const std::unique_ptr<Object> &object) { m_objects.insert({object->getId(),
00031         std::move(*object)}); }
00032     void addLight(const std::unique_ptr<Light> &light) { m_lights.insert({light->getId(),
00033         std::move(*light)}); }
00034
00035     void destroyObject(const unsigned int objectId) { m_objects.erase(objectId); }
00036     void destroyLight(const unsigned int lightId) { m_lights.erase(lightId); }
00037     void destroyEntity(std::vector<unsigned int> *objectIds, std::vector<unsigned int>
00038         *lightsIds);
00039
00040     void updateBuffer(GlobalUbo &ubo, unsigned long frameIndex, float frameTime);
00041
00042     [[nodiscard]] VkDescriptorBufferInfo getBufferInfoForObject(const int frameIndex, const
00043         unsigned int objectId) const { return m_uboBuffers.at(static_cast<long unsigned
00044         int>(frameIndex))->descriptorInfoForIndex(objectId); }
00045     Object::Map& getObjects() { return m_objects; }
00046     Light::Map& getLights() { return m_lights; }
00047     std::vector<std::unique_ptr<Buffer>> &getUboBuffers() { return m_uboBuffers; }
00048     std::shared_ptr<Texture> getTextureDefault() { return m_textureDefault; }
00049     [[nodiscard]] bool getDestroyState() const { return m_destroyState; }
00050
00051     void setDestroyState(const bool state) { m_destroyState = state; }
```

```

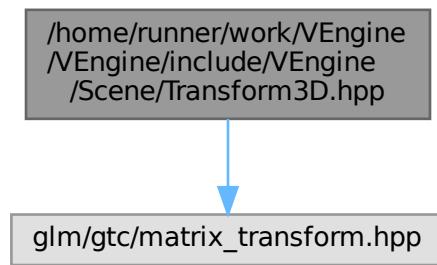
00048     private:
00049
00050         unsigned int m_currentObjId{0};
00051         unsigned int m_currentLightId{0};
00052         std::shared_ptr<Texture> m_textureDefault;
00053         Object::Map m_objects;
00054         Light::Map m_lights;
00055         std::vector<std::unique_ptr<Buffer>> m_uboBuffers{MAX_FRAMES_IN_FLIGHT};
00056         bool m_destroyState{false};
00057
00058     }; // class SceneManager
00059
00060 } // namespace ven

```

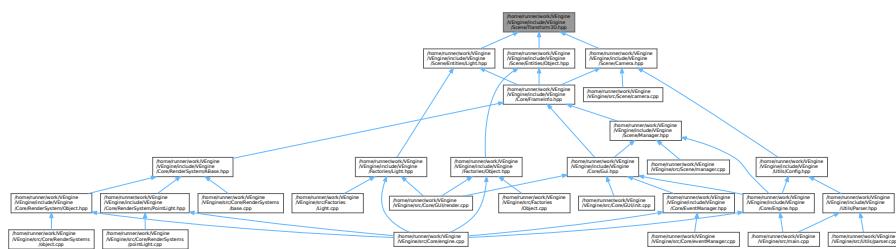
## 8.63 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp File Reference

This file contains the Transform3D class.

#include <glm/gtc/matrix\_transform.hpp>  
Include dependency graph for Transform3D.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- class [ven::Transform3D](#)

*Class for 3D transformation.*

## Namespaces

- namespace `ven`

### 8.63.1 Detailed Description

This file contains the `Transform3D` class.

Definition in file [Transform3D.hpp](#).

## 8.64 Transform3D.hpp

[Go to the documentation of this file.](#)

```

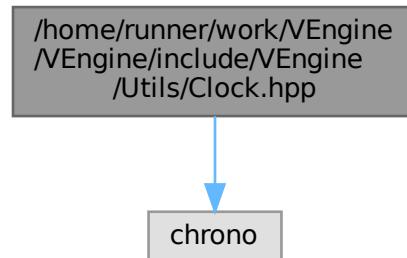
00001 /**
00002 /// @file Transform3D.hpp
00003 /// @brief This file contains the Transform3D class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <glm/gtc/matrix_transform.hpp>
00010
00011 namespace ven {
00012
00013 /**
00014 /// @class Transform3D
00015 /// @brief Class for 3D transformation
00016 /// @namespace ven
00017 /**
00018 class Transform3D {
00019
00020     public:
00021
00022     [[nodiscard]] glm::mat4 transformMatrix() const {
00023         auto rotationMatrix = glm::mat4(1.0F);
00024
00025         rotationMatrix = rotate(rotationMatrix, rotation.x, glm::vec3(1.0F, 0.0F, 0.0F));
00026         rotationMatrix = rotate(rotationMatrix, rotation.y, glm::vec3(0.0F, 1.0F, 0.0F));
00027         rotationMatrix = rotate(rotationMatrix, rotation.z, glm::vec3(0.0F, 0.0F, 1.0F));
00028
00029         const glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0F), scale);
00030         const glm::mat4 translationMatrix = translate(glm::mat4(1.0F), translation);
00031
00032         return translationMatrix * rotationMatrix * scaleMatrix;
00033     }
00034     [[nodiscard]] glm::mat3 normalMatrix() const { return
00035         transpose(inverse(glm::mat3(transformMatrix())));
00036     }
00037     glm::vec3 translation{};
00038     glm::vec3 scale{};
00039     glm::vec3 rotation{};
00040 }; // class Transform3D
00041
00042 } // namespace ven

```

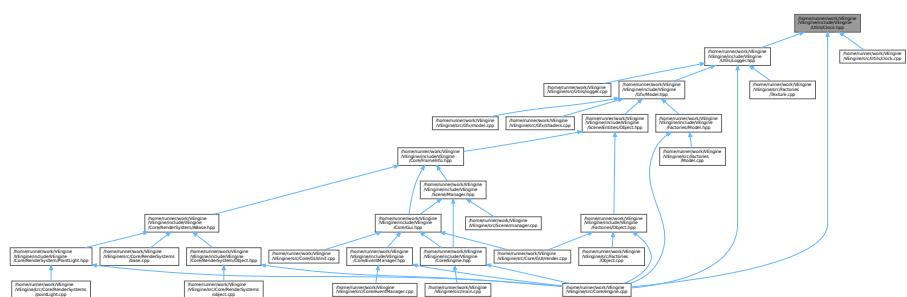
## 8.65 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/← Clock.hpp File Reference

This file contains the `Clock` class.

```
#include <chrono>
Include dependency graph for Clock.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Clock](#)  
*Class for clock.*

## Namespaces

- namespace [ven](#)

## Typedefs

- using [ven::TimePoint](#) = std::chrono::time\_point<std::chrono::high\_resolution\_clock>

### 8.65.1 Detailed Description

This file contains the Clock class.

Definition in file [Clock.hpp](#).

## 8.66 Clock.hpp

[Go to the documentation of this file.](#)

```

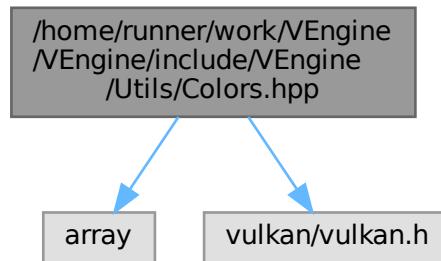
00001 /**
00002 /// @file Clock.hpp
00003 /// @brief This file contains the Clock class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <chrono>
00010
00011 namespace ven {
00012
00013     using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;
00014
00015 /**
00016 /// @class Clock
00017 /// @brief Class for clock
00018 /// @namespace ven
00019 /**
00020 class Clock {
00021
00022     public:
00023
00024         Clock() { start(); }
00025         ~Clock() = default;
00026
00027         Clock(const Clock&) = delete;
00028         Clock& operator=(const Clock&) = delete;
00029
00030         void start() { m_startTime = std::chrono::high_resolution_clock::now(); }
00031         void stop();
00032         void resume();
00033         void update();
00034         static std::chrono::high_resolution_clock::time_point now() { return
00035             std::chrono::high_resolution_clock::now(); }
00036
00037         [[nodiscard]] float getDeltaTime() const { return m_deltaTime.count(); }
00038         [[nodiscard]] float getDeltaTimeMS() const { return m_deltaTime.count() * 1000.F; }
00039         [[nodiscard]] float getFPS() const { return 1.F / m_deltaTime.count(); }
00040
00041     private:
00042         TimePoint mStartTime;
00043         TimePoint mStopTime;
00044         std::chrono::duration<float> mDeltaTime{0.F};
00045
00046         bool m_isStopped{false};
00047
00048     }; // class Clock
00049
00050 } // namespace ven

```

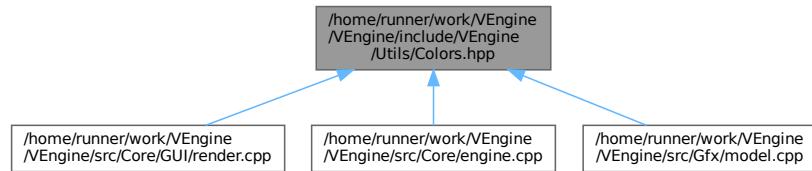
## 8.67 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp File Reference

```
#include <array>
#include <vulkan/vulkan.h>
```

Include dependency graph for Colors.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Colors](#)

*Class for colors.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr float [ven::COLOR\\_MAX](#) = 255.0F

## 8.68 Colors.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Colors.hpp
00003 /// @brief
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <array>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 namespace ven {
00014
00015     static constexpr float COLOR_MAX = 255.0F;
00016
00017 /**
00018     /// @class Colors
00019     /// @brief Class for colors
00020     /// @namespace ven
00021 /**
00022     class Colors {
00023
00024     public:
00025
00026         static constexpr glm::vec3 WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX;
00027         static constexpr glm::vec4 WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F };
00028         static constexpr VkClearColorValue WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } };

00029         static constexpr glm::vec3 BLACK_3 = glm::vec3(0.0F);
00030         static constexpr glm::vec4 BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F };
00031         static constexpr VkClearColorValue BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } };

00032         static constexpr glm::vec3 RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
00033         static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F };
00034         static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } };

00035         static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00036         static constexpr glm::vec4 GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00037         static constexpr VkClearColorValue GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };

00038         static constexpr glm::vec3 BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
00039         static constexpr glm::vec4 BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F };
00040         static constexpr VkClearColorValue BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } };

00041         static constexpr glm::vec3 YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
00042         static constexpr glm::vec4 YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F };
00043         static constexpr VkClearColorValue YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } };

00044         static constexpr glm::vec3 CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00045         static constexpr glm::vec4 CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00046         static constexpr VkClearColorValue CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };

00047         static constexpr glm::vec3 MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00048         static constexpr glm::vec4 MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00049         static constexpr VkClearColorValue MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };

00050         static constexpr glm::vec3 SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
00051         static constexpr glm::vec4 SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F };
00052         static constexpr VkClearColorValue SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } };

00053         static constexpr glm::vec3 GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
00054         static constexpr glm::vec4 GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F };
00055         static constexpr VkClearColorValue GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } };

00056         static constexpr glm::vec3 MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
00057         static constexpr glm::vec4 MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F };
00058         static constexpr VkClearColorValue MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } };

00059         static constexpr glm::vec3 OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
00060         static constexpr glm::vec4 OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F };
00061         static constexpr VkClearColorValue OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } };

00062         static constexpr glm::vec3 LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00063         static constexpr glm::vec4 LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00064         static constexpr VkClearColorValue LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };

00065         static constexpr glm::vec3 AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00066         static constexpr glm::vec4 AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00067         static constexpr VkClearColorValue AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };

00068         static constexpr glm::vec3 TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;

```

```

00083     static constexpr glm::vec4 TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F };
00084     static constexpr VkClearColorValue TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } };
00085
00086     static constexpr glm::vec3 NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
00087     static constexpr glm::vec4 NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F };
00088     static constexpr VkClearColorValue NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } };
00089
00090     static constexpr glm::vec3 FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00091     static constexpr glm::vec4 FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00092     static constexpr VkClearColorValue FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };
00093
00094     static constexpr glm::vec3 NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
00095     static constexpr glm::vec4 NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F };
00096     static constexpr VkClearColorValue NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } };
00097
00098     static constexpr glm::vec3 SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
00099     static constexpr glm::vec4 SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F };
00100     static constexpr VkClearColorValue SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } };
00101
00102     static constexpr glm::vec3 SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
00103     static constexpr glm::vec4 SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F };
00104     static constexpr VkClearColorValue SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } };
00105
00106
00107     static constexpr std::array<std::pair<const char *, glm::vec3>, 20> COLOR_PRESETS_3 = {{
00108         {"White", WHITE_3},
00109         {"Black", BLACK_3},
00110         {"Red", RED_3},
00111         {"Green", GREEN_3},
00112         {"Blue", BLUE_3},
00113         {"Yellow", YELLOW_3},
00114         {"Cyan", CYAN_3},
00115         {"Magenta", MAGENTA_3},
00116         {"Silver", SILVER_3},
00117         {"Gray", GRAY_3},
00118         {"Maroon", MAROON_3},
00119         {"Olive", OLIVE_3},
00120         {"Lime", LIME_3},
00121         {"Aqua", AQUA_3},
00122         {"Teal", TEAL_3},
00123         {"Navy", NAVY_3},
00124         {"Fuchsia", FUCHSIA_3},
00125         {"Night Blue", NIGHT_BLUE_3},
00126         {"Sky Blue", SKY_BLUE_3},
00127         {"Sunset", SUNSET_3}
00128     }};
00129
00130     static constexpr std::array<std::pair<const char *, glm::vec4>, 20> COLOR_PRESETS_4 = {{
00131         {"White", WHITE_4},
00132         {"Black", BLACK_4},
00133         {"Red", RED_4},
00134         {"Green", GREEN_4},
00135         {"Blue", BLUE_4},
00136         {"Yellow", YELLOW_4},
00137         {"Cyan", CYAN_4},
00138         {"Magenta", MAGENTA_4},
00139         {"Silver", SILVER_4},
00140         {"Gray", GRAY_4},
00141         {"Maroon", MAROON_4},
00142         {"Olive", OLIVE_4},
00143         {"Lime", LIME_4},
00144         {"Aqua", AQUA_4},
00145         {"Teal", TEAL_4},
00146         {"Navy", NAVY_4},
00147         {"Fuchsia", FUCHSIA_4},
00148         {"Night Blue", NIGHT_BLUE_4},
00149         {"Sky Blue", SKY_BLUE_4},
00150         {"Sunset", SUNSET_4}
00151     }};
00152
00153     static constexpr std::array<std::pair<const char *, VkClearColorValue>, 20>
COLOR_PRESETS_VK = {{
00154         {"White", WHITE_V},
00155         {"Black", BLACK_V},
00156         {"Red", RED_V},
00157         {"Green", GREEN_V},
00158         {"Blue", BLUE_V},
00159         {"Yellow", YELLOW_V},
00160         {"Cyan", CYAN_V},
00161         {"Magenta", MAGENTA_V},
00162         {"Silver", SILVER_V},
00163         {"Gray", GRAY_V},
00164         {"Maroon", MAROON_V},
00165         {"Olive", OLIVE_V},
00166         {"Lime", LIME_V},
00167         {"Aqua", AQUA_V},
00168         {"Teal", TEAL_V},

```

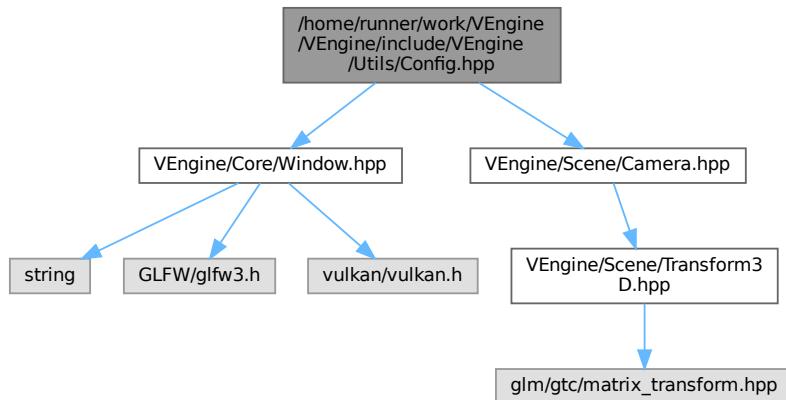
```

00169         {"Navy", NAVY_V},
00170         {"Fuchsia", FUCHSIA_V},
00171         {"Night Blue", NIGHT_BLUE_V},
00172         {"Sky Blue", SKY_BLUE_V},
00173         {"Sunset", SUNSET_V}
00174     });
00175
00176 } // class Colors
00177
00178 } // namespace ven

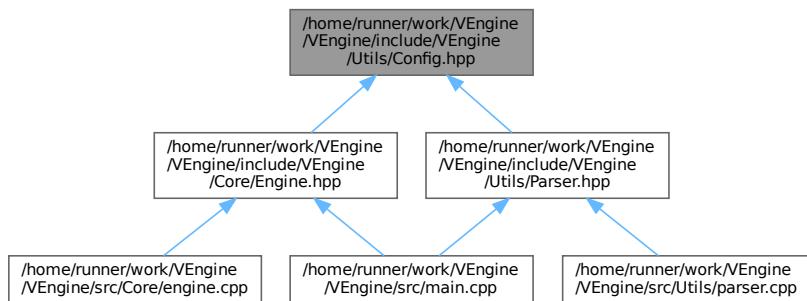
```

## 8.69 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Config.hpp File Reference

```
#include "VEngine/Core/Window.hpp"
#include "VEngine/Scene/Camera.hpp"
Include dependency graph for Config.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- struct `ven::WindowConf`
- struct `ven::CameraConf`
- struct `ven::Config`

## Namespaces

- namespace [ven](#)

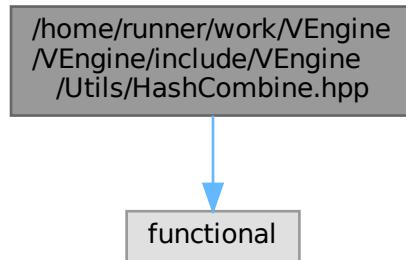
## 8.70 Config.hpp

[Go to the documentation of this file.](#)

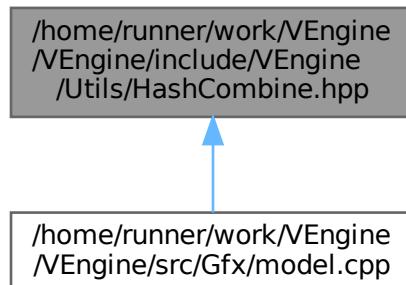
```
00001 /**
00002 /// @file Config.hpp
00003 /// @brief
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #ifndef _WIN32
00010     #undef near
00011     #undef far
00012 #endif
00013
00014 #include "VEngine/Core/Window.hpp"
00015 #include "VEngine/Scene/Camera.hpp"
00016
00017 namespace ven {
00018
00019     struct WindowConf
00020     {
00021         uint16_t width = DEFAULT_WIDTH;
00022         uint16_t height = DEFAULT_HEIGHT;
00023         bool fullscreen = false; // TODO: Implement fullscreen
00024     };
00025
00026     struct CameraConf
00027     {
00028         float fov = DEFAULT_FOV;
00029         float move_speed = DEFAULT_MOVE_SPEED;
00030         float look_speed = DEFAULT_LOOK_SPEED;
00031         float near = DEFAULT_NEAR;
00032         float far = DEFAULT_FAR;
00033     };
00034
00035     struct Config
00036     {
00037         WindowConf window;
00038         CameraConf camera;
00039         bool vsync = false; // TODO: Implement vsync
00040     };
00041
00042 } // namespace ven
```

## 8.71 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp File Reference

```
#include <functional>
Include dependency graph for HashCombine.hpp:
```



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `ven`

### Functions

- template<typename T , typename... Rest>  
void `ven::hashCombine` (std::size\_t &seed, const T &v, const Rest &... rest)

## 8.72 HashCombine.hpp

[Go to the documentation of this file.](#)

```

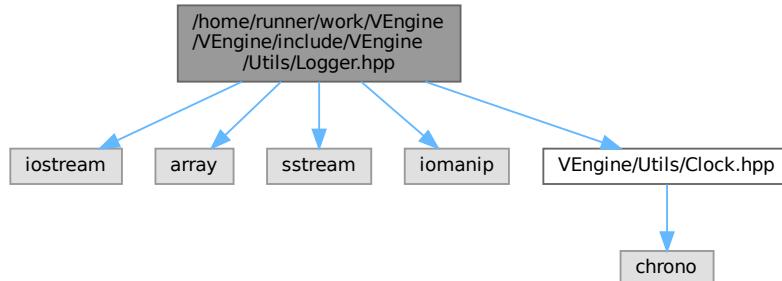
00001 /**
00002 /// @file HashCombine.hpp
00003 /// @brief
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018 } // namespace ven

```

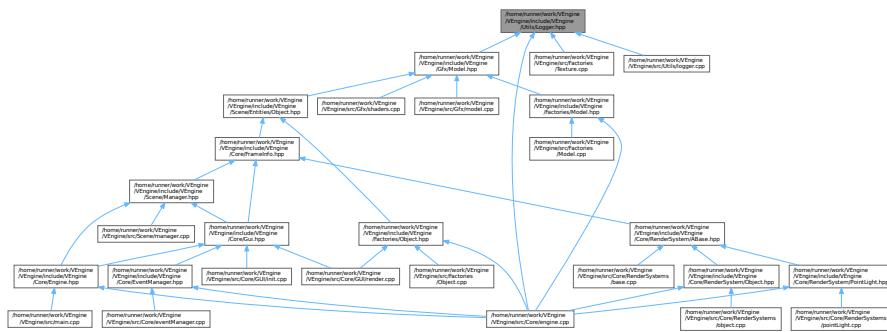
## 8.73 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp File Reference

This file contains the `Logger` class.

```
#include <iostream>
#include <array>
#include <sstream>
#include <iomanip>
#include "VEngine/Utils/Clock.hpp"
Include dependency graph for Logger.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Logger](#)

## Namespaces

- namespace [ven](#)

## Enumerations

- enum class [ven::LogLevel](#) : uint8\_t { [ven::INFO](#) , [ven::WARNING](#) }

### 8.73.1 Detailed Description

This file contains the Logger class.

Definition in file [Logger.hpp](#).

## 8.74 Logger.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Logger.hpp
00003 /// @brief This file contains the Logger class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <iostream>
00010 #include <array>
00011 #include <sstream>
00012 #include <iomanip>
00013
00014 #ifdef _WIN32
00015     #include <windows.h>
00016 #endif
00017
00018 #include "VEngine/Utils/Clock.hpp"
00019
00020
00021 namespace ven {

```

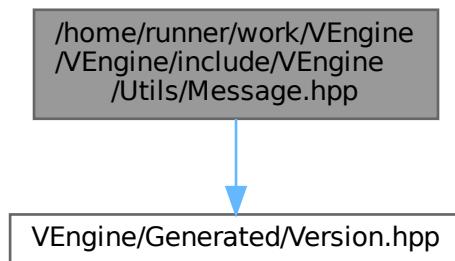
```

00022
00023     enum class LogLevel : uint8_t {
00024         INFO,
00025         WARNING
00026     };
00027
00028     class Logger {
00029         public:
00030             static Logger& getInstance() { static Logger instance; return instance; }
00031
00032             template <typename Func>
00033             static void logExecutionTime(const std::string& message, Func&& func)
00034             {
00035                 Clock clock;
00036                 func();
00037                 clock.update();
00038                 const float duration = clock.getDeltaTimeMS();
00039
00040                 std::cout << getColorForDuration(duration) << formatLogMessage(LogLevel::INFO, message +
00041 " took " + std::to_string(duration) + " ms") << LOG_LEVEL_COLOR.at(3);
00042             }
00043
00044             static void logWarning(const std::string& message) { std::cout << LOG_LEVEL_COLOR.at(2) <<
00045                 formatLogMessage(LogLevel::WARNING, message) << LOG_LEVEL_COLOR.at(3); }
00046
00047         private:
00048             static constexpr std::array<const char*, 2> LOG_LEVEL_STRING = {"INFO", "WARNING"};
00049             static constexpr std::array<const char*, 4> LOG_LEVEL_COLOR = {"\033[31m", "\033[32m",
00050 "\033[33m", "\033[0m\n"};
00051
00052             Logger();
00053
00054             static const char* getColorForDuration(const float duration) { return duration < 20.0F ?
00055                 LOG_LEVEL_COLOR.at(1) : (duration < 90.0F ? LOG_LEVEL_COLOR.at(2) : LOG_LEVEL_COLOR.at(0)); }
00056
00057             [[nodiscard]] static std::string formatLogMessage(LogLevel level, const std::string&
00058             message);
00059
00060     }; // class Logger
00061
00062 } // namespace ven

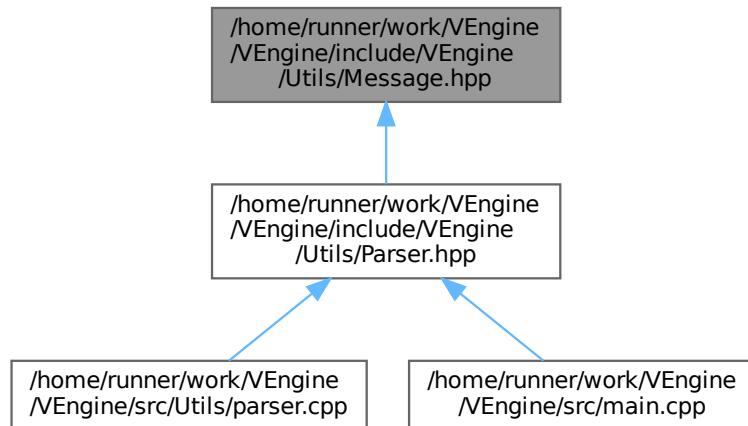
```

## 8.75 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Message.hpp File Reference

#include "VEngine/Generated/Version.hpp"  
Include dependency graph for Message.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `ven`

## Variables

- `constexpr auto VERSION_MESSAGE`
- `constexpr auto HELP_MESSAGE`

## 8.76 Message.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  /// @file Message.hpp
00003  /// @brief
00004  /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Generated/Version.hpp"
00010
00011 namespace ven {
00012
00013     constexpr auto VERSION_MESSAGE = "VEngine Version " PROJECT_VERSION "\n"
00014         "Built on " __DATE__
00015         " at " __TIME__
00016         "\nAuthor: Elliot Masina (bobis33)\n"
00017         "License: MIT\n"
00018         "Repository: https://github.com/bobis33/VEngine\n"
00019         "Documentation: https://bobis33.github.io/VEngine/\n";
00020
00021     constexpr auto HELP_MESSAGE = "Usage: VEngine [options]\n"
00022         "Options:\n"
00023         "  --help, -h           Show this help message and exit\n"
00024         "  --version, -v        Show version information and exit\n"
00025         "  --fullscreen, -f      Enable fullscreen mode\n"
00026         "  --vsync, -V          Enable vertical sync\n"
00027         "  --width <value>       Set the width of the window (e.g., 800)\n"
  
```

```

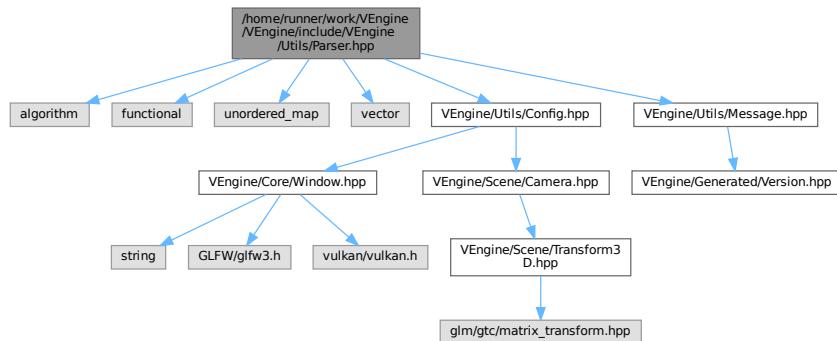
00028      " --height <value>      Set the height of the window (e.g., 600)\n"
00029      " --fov <value>        Set the field of view (1.0 to 300.0)\n"
00030      " --mspeed <value>      Set the move speed (0.1 to 100.0)\n"
00031      " --lspeed <value>      Set the look speed (0.1 to 100.0)\n"
00032      " --near <value>        Set the near plane (0.1 to 100.0)\n"
00033      " --far <value>         Set the far plane (0.1 to 100.0)\n";
00034
00035 } // namespace ven

```

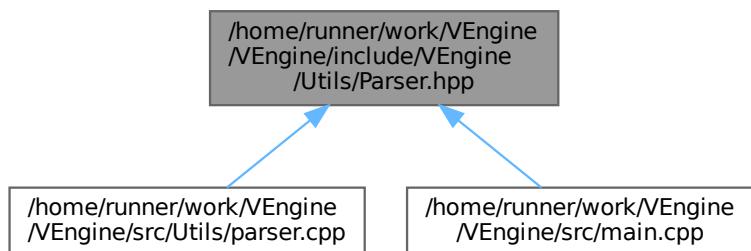
## 8.77 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Parser.hpp File Reference

This file contains the Parser class.

```
#include <algorithm>
#include <functional>
#include <unordered_map>
#include <vector>
#include "VEngine/Utils/Config.hpp"
#include "VEngine/Utils/Message.hpp"
Include dependency graph for Parser.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::ParserException`  
*Custom exception class for parsing errors.*
- class `ven::Parser`  
*Class for Parser.*

## Namespaces

- namespace `ven`

## Functions

- bool `ven::isNumeric` (const std::string\_view str)

## Variables

- static const std::unordered\_map< std::string\_view, std::function< void(Config &conf, std::string\_view arg)> > `ven::FUNCTION_MAP_OPT_LONG`
- static const std::unordered\_map< std::string\_view, std::function< void(Config &conf)> > `ven::FUNCTION_MAP_OPT_SHORT`

### 8.77.1 Detailed Description

This file contains the Parser class.

Definition in file [Parser.hpp](#).

## 8.78 Parser.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002  /// @file Parser.hpp
00003  /// @brief This file contains the Parser class
00004  /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <algorithm>
00010 #include <functional>
00011 #include <unordered_map>
00012 #include <vector>
00013
00014 #include "VEngine/Utils/Config.hpp"
00015 #include "VEngine/Utils/Message.hpp"
00016
00017 namespace ven {
00018 /**
00019  /// @class ParserException
00020  /// @brief Custom exception class for parsing errors.
00021  /// @namespace ven
00022 /**
00023  class ParserException final : public std::exception {
00024
00025      public:
00026
00027          explicit ParserException(std::string msg) : m_msg{std::move(msg)} {};
00028          ~ParserException() override = default;
```

```

00031     ParserException(const ParserException&) = delete;
00032     ParserException& operator=(const ParserException&) = delete;
00033     ParserException(ParserException&&) = delete;
00034     ParserException& operator=(ParserException&&) = delete;
00035
00036     [[nodiscard]] const char *what() const noexcept override { return m_msg.c_str(); };
00037
00038     private:
00039
00040         std::string m_msg{0};
00041
00042     }; // class ParserException
00043
00044     inline bool isNumeric(const std::string_view str) { return !str.empty() &&
00045         std::ranges::all_of(str, ::isdigit); }
00046
00047     static const std::unordered_map<std::string_view, std::function<void(Config& conf,
00048         std::string_view arg)>> FUNCTION_MAP_OPT_LONG = {
00049         { "help", [](Config& conf, std::string_view arg) { std::cout << HELP_MESSAGE; throw
00050             ParserException(""); } },
00051         { "version", [](Config& conf, std::string_view arg) { std::cout << VERSION_MESSAGE; throw
00052             ParserException(""); } },
00053         { "width", [](Config& conf, const std::string_view arg)
00054         {
00055             if (!isNumeric(arg)) {
00056                 throw std::invalid_argument("Invalid value for width: " + std::string(arg));
00057             }
00058             conf.window.width = std::stoi(std::string(arg));
00059         } },
00060         { "height", [](Config& conf, const std::string_view arg)
00061         {
00062             if (!isNumeric(arg)) {
00063                 throw std::invalid_argument("Invalid value for height: " + std::string(arg));
00064             }
00065             conf.window.height = std::stoi(std::string(arg));
00066         } },
00067         { "fullscreen", [](Config& conf, std::string_view arg) { conf.window.fullscreen = true; } },
00068         { "vsync", [](Config& conf, std::string_view arg) { conf.vsync = true; } },
00069         { "fov", [](Config& conf, const std::string_view arg)
00070         {
00071             if (!isNumeric(arg)) {
00072                 throw std::invalid_argument("Invalid value for fov: " + std::string(arg));
00073             }
00074             const float value = std::stof(std::string(arg));
00075             if (value < 1.0F || value > 300.0F) {
00076                 throw std::out_of_range("Field of view must be between 1 and 300");
00077             }
00078             conf.camera.fov = value;
00079         } },
00080         { "mspeed", [](Config& conf, const std::string_view arg)
00081         {
00082             if (!isNumeric(arg)) {
00083                 throw std::invalid_argument("Invalid value for move speed: " + std::string(arg));
00084             }
00085             const float value = std::stof(std::string(arg));
00086             if (value < 0.1F || value > 100.0F) {
00087                 throw std::out_of_range("Move speed must be between 0.1 and 100.0");
00088             }
00089             conf.camera.move_speed = value;
00090         } },
00091         { "lspeed", [](Config& conf, const std::string_view arg)
00092         {
00093             if (!isNumeric(arg)) {
00094                 throw std::invalid_argument("Invalid value for look speed: " + std::string(arg));
00095             }
00096             const float value = std::stof(std::string(arg));
00097             if (value < 0.1F || value > 100.0F) {
00098                 throw std::out_of_range("Look speed must be between 0.1 and 100.0");
00099             }
00100             conf.camera.look_speed = value;
00101         } },
00102         { "far", [](Config& conf, const std::string_view arg)
00103         {
00104             if (!isNumeric(arg)) {
00105                 throw std::invalid_argument("Invalid value for far: " + std::string(arg));
00106             }
00107             const float value = std::stof(std::string(arg));
00108             if (value < 0.1F || value > 100.0F) {
00109                 throw std::out_of_range("Far plane must be between 0.1 and 100.0");
00110             }
00111             conf.camera.far = value;
00112         } },
00113         { "near", [](Config& conf, const std::string_view arg)
00114         {
00115             if (!isNumeric(arg)) {
00116                 throw std::invalid_argument("Invalid value for near: " + std::string(arg));
00117             }
00118         } }
00119     };

```

```

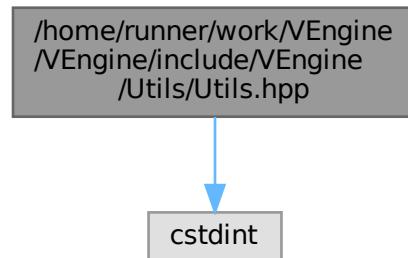
00114         const float value = std::stof(std::string(arg));
00115         if (value < 0.1F || value > 100.0F) {
00116             throw std::out_of_range("Near plane must be between 0.1 and 100.0");
00117         }
00118         conf.camera.near = value;
00119     } }
00120 };
00121
00122 static const std::unordered_map<std::string_view, std::function<void(Config& conf)>>
FUNCTION_MAP_OPT_SHORT = {
00123     { "h", [](Config& conf) { std::cout << HELP_MESSAGE; throw ParserException(""); } },
00124     { "v", [](Config& conf) { std::cout << VERSION_MESSAGE; throw ParserException(""); } },
00125     { "f", [](Config& conf) { conf.window.fullscreen = true; } },
00126     { "V", [](Config& conf) { conf.vsync = true; } }
00127 };
00128
00129 /**
00130 /// @class Parser
00131 /// @brief Class for Parser
00132 /// @namespace ven
00133 /**
00134 class Parser {
00135
00136     public:
00137
00138     Parser(int argc, char* argv[], char* envp[]);
00139     ~Parser() = default;
00140
00141     Parser(const Parser&) = delete;
00142     Parser& operator=(const Parser&) = delete;
00143     Parser(Parser&&) = delete;
00144     Parser& operator=(Parser&&) = delete;
00145
00146     void printArguments() const;
00147
00148     [[nodiscard]] Config getConfig() const { return m_config; }
00149
00150     private:
00151
00152     Config m_config;
00153     bool m_state = true;
00154
00155     void parseArgs(const std::vector<std::string_view>& argv);
00156     void parseEnv(const std::unordered_map<std::string, std::string>& envp);
00157
00158     void handleLongOption(const std::string_view& arg, const std::vector<std::string_view>&
00159     argv, size_t& index);
00160     void handleShortOptions(const std::string_view& arg, const std::vector<std::string_view>&
00161     argv, size_t& index);
00162
00163     [[nodiscard]] static bool isValidOption(const std::string_view& option) { return
00164     FUNCTION_MAP_OPT_LONG.contains(option) || FUNCTION_MAP_OPT_SHORT.contains(option); }
00165 }; // class Parser
00166 // namespace ven

```

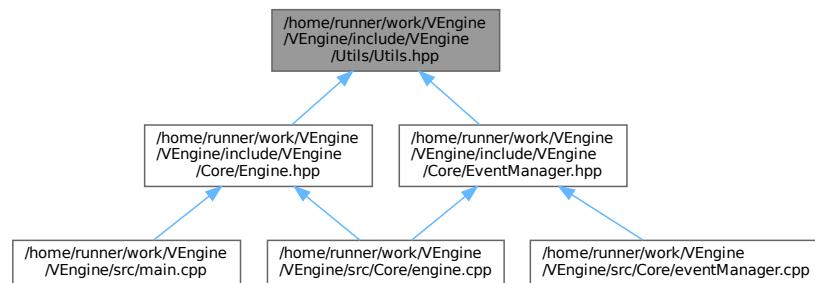
## 8.79 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp File Reference

This file contains utils for VEngine.

```
#include <cstdint>
Include dependency graph for Utils.hpp:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `ven`

## Enumerations

- enum `ven::ENGINE_STATE` : `uint8_t` { `ven::EDITOR` = 0 , `ven::PLAYER` = 1 , `ven::PAUSED` = 2 , `ven::EXIT` = 3 }

### 8.79.1 Detailed Description

This file contains utils for VEngine.

Definition in file [Utils.hpp](#).

## 8.80 Utils.hpp

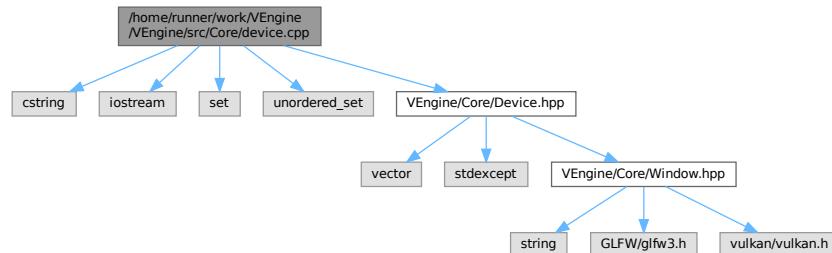
[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Utils.hpp
00003 /// @brief This file contains utils for VEngine
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <cstdint>
00010
00011 namespace ven {
00012
00013     enum ENGINE_STATE : uint8_t {
00014         EDITOR = 0,
00015         PLAYER = 1,
00016         PAUSED = 2,
00017         EXIT = 3
00018     };
00019 }
00020 } // namespace ven
```

## 8.81 /home/runner/work/VEngine/VEngine/README.md File Reference

## 8.82 /home/runner/work/VEngine/VEngine/src/Core/device.cpp File Reference

```
#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>
#include "VEngine/Core/Device.hpp"
Include dependency graph for device.cpp:
```



### Functions

- static VKAPI\_ATTR VkBool32 VKAPI\_CALL [debugCallback](#) (const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtilsMessengerCallbackDataEXT \*pCallbackData, void \*pUserData)
- VkResult [CreateDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerCreateInfoEXT \*pCreateInfo, const VkAllocationCallbacks \*pAllocator, VkDebugUtilsMessengerEXT \*pDebugMessenger)
- void [DestroyDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks \*pAllocator)

## 8.82.1 Function Documentation

### 8.82.1.1 CreateDebugUtilsMessengerEXT()

```
VkResult CreateDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerCreateInfoEXT * pCreateInfo,
    const VkAllocationCallbacks * pAllocator,
    VkDebugUtilsMessengerEXT * pDebugMessenger)
```

Definition at line 16 of file [device.cpp](#).

Referenced by [ven::Device::setupDebugMessenger\(\)](#).

Here is the caller graph for this function:



### 8.82.1.2 debugCallback()

```
static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (
    const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity,
    const VkDebugUtilsMessageTypeFlagsEXT messageType,
    const VkDebugUtilsMessengerCallbackDataEXT * pCallbackData,
    void * pUserData) [static]
```

Definition at line 8 of file [device.cpp](#).

Referenced by [ven::Device::populateDebugMessengerCreateInfo\(\)](#).

Here is the caller graph for this function:



### 8.82.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerEXT debugMessenger,
    const VkAllocationCallbacks * pAllocator)
```

Definition at line 25 of file [device.cpp](#).

Referenced by [ven::Device::~Device\(\)](#).

Here is the caller graph for this function:



## 8.83 device.cpp

[Go to the documentation of this file.](#)

```
00001 #include <cstring>
00002 #include <iostream>
00003 #include <set>
00004 #include <unordered_set>
00005
00006 #include "VEngine/Core/Device.hpp"
00007
00008 static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT
    messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const
    VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
00009 {
00010     (void) pUserData; (void) messageSeverity; (void) messageType;
00011
00012     std::cerr << "validation layer: " << pCallbackData->pMessage << '\n';
00013     return VK_FALSE;
00014 }
00015
00016 VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const
    VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator,
    VkDebugUtilsMessengerEXT *pDebugMessenger)
00017 {
00018     if (const auto func =
        reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
        "vkCreateDebugUtilsMessengerEXT")); func != nullptr) {
00019         return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00020     }
00021
00022     return VK_ERROR_EXTENSION_NOT_PRESENT;
00023 }
00024
00025 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
    debugMessenger, const VkAllocationCallbacks *pAllocator)
00026 {
00027     if (const auto func =
        reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
        "vkDestroyDebugUtilsMessengerEXT")); func != nullptr) {
00028         func(instance, debugMessenger, pAllocator);
00029     }
00030 }
00031
00032 ven::Device::Device(Window &window) : m_window{window}
00033 {
00034     createInstance();
00035     setupDebugMessenger();
00036     createSurface();
00037     pickPhysicalDevice();
```

```

00038     createLogicalDevice();
00039     createCommandPool();
00040 }
00041
00042 ven::Device::~Device()
00043 {
00044     vkDestroyCommandPool(m_device, m_commandPool, nullptr);
00045     vkDestroyDevice(m_device, nullptr);
00046
00047     if (enableValidationLayers) {
00048         DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
00049     }
00050
00051     vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
00052     vkDestroyInstance(m_instance, nullptr);
00053 }
00054
00055 void ven::Device::createInstance()
00056 {
00057     if (enableValidationLayers && !checkValidationLayerSupport()) {
00058         throw std::runtime_error("validation layers requested, but not available!");
00059     }
00060
00061     constexpr VkApplicationInfo appInfo = {
00062         .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
00063         .pNext = nullptr,
00064         .pApplicationName = "VEngine App",
00065         .applicationVersion = VK_MAKE_VERSION(1, 0, 0),
00066         .pEngineName = "VEngine",
00067         .engineVersion = VK_MAKE_VERSION(1, 0, 0),
00068         .apiVersion = VK_API_VERSION_1_0
00069     };
00070
00071     VkInstanceCreateInfo createInfo = {};
00072     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00073     createInfo.pApplicationInfo = &appInfo;
00074
00075     const std::vector<const char *> extensions = getRequiredExtensions();
00076     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00077     createInfo.ppEnabledExtensionNames = extensions.data();
00078
00079     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00080     if (enableValidationLayers) {
00081         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00082         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00083
00084         populateDebugMessengerCreateInfo(debugCreateInfo);
00085         createInfo.pNext = &debugCreateInfo;
00086     } else {
00087         createInfo.enabledLayerCount = 0;
00088         createInfo.pNext = nullptr;
00089     }
00090
00091     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
00092         throw std::runtime_error("failed to create instance!");
00093     }
00094
00095     hasGlfwRequiredInstanceExtensions();
00096 }
00097
00098 void ven::Device::pickPhysicalDevice()
00099 {
00100     uint32_t deviceCount = 0;
00101     vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
00102     if (deviceCount == 0) {
00103         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00104     }
00105     std::cout << "Device count: " << deviceCount << '\n';
00106     std::vector<VkPhysicalDevice> devices(deviceCount);
00107     vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());
00108
00109     for (const auto &device : devices) {
00110         if (isDeviceSuitable(device)) {
00111             m_physicalDevice = device;
00112             break;
00113         }
00114     }
00115
00116     if (m_physicalDevice == VK_NULL_HANDLE) {
00117         throw std::runtime_error("failed to find a suitable GPU!");
00118     }
00119
00120     vkGetPhysicalDeviceProperties(m_physicalDevice, &m_properties);
00121     std::cout << "physical device: " << m_properties.deviceName << '\n';
00122 }
00123
00124 void ven::Device::createLogicalDevice()

```

```

00125 {
00126     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
00127         findQueueFamilies(m_physicalDevice);
00128     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00129     const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, presentFamily};
00130     float queuePriority = 1.0f;
00131
00132     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00133         VkDeviceQueueCreateInfo queueCreateInfo = {};
00134         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00135         queueCreateInfo.queueFamilyIndex = queueFamily;
00136         queueCreateInfo.queueCount = 1;
00137         queueCreateInfo.pQueuePriorities = &queuePriority;
00138         queueCreateInfo.push_back(queueCreateInfo);
00139     }
00140
00141     VkPhysicalDeviceFeatures deviceFeatures = {};
00142     deviceFeatures.samplerAnisotropy = VK_TRUE;
00143
00144     VkDeviceCreateInfo createInfo = {};
00145     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00146
00147     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00148     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00149
00150     createInfo.pEnabledFeatures = &deviceFeatures;
00151     createInfo.enabledExtensionCount = static_cast<uint32_t>(m_deviceExtensions.size());
00152     createInfo.ppEnabledExtensionNames = m_deviceExtensions.data();
00153
00154     // might not really be necessary anymore because device specific validation layers
00155     // have been deprecated
00156     if (enableValidationLayers) {
00157         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00158         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00159     } else {
00160         createInfo.enabledLayerCount = 0;
00161     }
00162
00163     if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
00164         throw std::runtime_error("failed to create logical device!");
00165     }
00166
00167     vkGetDeviceQueue(m_device, graphicsFamily, 0, &m_graphicsQueue);
00168     vkGetDeviceQueue(m_device, presentFamily, 0, &m_presentQueue);
00169 }
00170
00171 void ven::Device::createCommandPool()
00172 {
00173     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
00174         findPhysicalQueueFamilies();
00175
00176     const VkCommandPoolCreateInfo poolInfo = {
00177         .sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO,
00178         .pNext = nullptr,
00179         .flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
00180                 VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT,
00181         .queueFamilyIndex = graphicsFamily
00182     };
00183
00184     if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
00185         throw std::runtime_error("failed to create command pool!");
00186     }
00187
00188     bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00189     {
00190         const QueueFamilyIndices indices = findQueueFamilies(device);
00191         const bool extensionsSupported = checkDeviceExtensionSupport(device);
00192         bool swapChainAdequate = false;
00193
00194         if (extensionsSupported) {
00195             auto [capabilities, formats, presentModes] = querySwapChainSupport(device);
00196             swapChainAdequate = !formats.empty() && !presentModes.empty();
00197         }
00198
00199         VkPhysicalDeviceFeatures supportedFeatures;
00200         vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00201
00202         return indices.isComplete() && extensionsSupported && swapChainAdequate &&
00203             (supportedFeatures.samplerAnisotropy != 0U);
00204     }
00205
00206     void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &CreateInfo)
00207     {
00208         CreateInfo = {};
00209         CreateInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;

```

```

00208     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
00209         VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00210     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
00211         VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
00212         VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00213     createInfo.pfnUserCallback = debugCallback;
00214     createInfo.pUserData = nullptr; // Optional
00215 }
00216
00217 void ven::Device::setupDebugMessenger()
00218 {
00219     if (!enableValidationLayers) { return; }
00220     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00221     populateDebugMessengerCreateInfo(createInfo);
00222     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
00223         VK_SUCCESS) {
00224         throw std::runtime_error("failed to set up debug messenger!");
00225     }
00226
00227 bool ven::Device::checkValidationLayerSupport() const
00228 {
00229     uint32_t layerCount = 0;
00230     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00231
00232     std::vector<VkLayerProperties> availableLayers(layerCount);
00233     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00234
00235     for (const char *validationLayer : m_validationLayers) {
00236         bool layerFound = false;
00237
00238         for (const auto &[layerName, specVersion, implementationVersion, description] :
00239             availableLayers) {
00240             if (strcmp(layerName, validationLayer) == 0) {
00241                 layerFound = true;
00242                 break;
00243             }
00244             if (!layerFound) {
00245                 return false;
00246             }
00247         }
00248
00249     return true;
00250 }
00251
00252 std::vector<const char *> ven::Device::getRequiredExtensions() const
00253 {
00254     uint32_t glfwExtensionCount = 0;
00255     const char **glfwExtensions = nullptr;
00256     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00257
00258     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00259
00260     if (enableValidationLayers) {
00261         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00262     }
00263
00264     return extensions;
00265 }
00266
00267 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00268 {
00269     uint32_t extensionCount = 0;
00270     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
00271     std::vector<VkExtensionProperties> extensions(extensionCount);
00272     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00273
00274     std::cout << "available extensions:\n";
00275     std::unordered_set<std::string> available;
00276     for (const auto &[extensionName, specVersion] : extensions) {
00277         std::cout << '\t' << extensionName << '\n';
00278         available.insert(extensionName);
00279     }
00280
00281     std::cout << "required extensions:\n";
00282     for (const std::vector<const char *> &requiredExtensions = getRequiredExtensions(); const auto
00283         &required : requiredExtensions) {
00284         std::cout << "\t" << required << '\n';
00285         if (!available.contains(required)) {
00286             throw std::runtime_error("Missing required glfw extension");
00287         }
00288     }
00289
00290 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00291 {

```

```
00292     uint32_t extensionCount = 0;
00293     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00294
00295     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00296     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
00297         availableExtensions.data());
00298
00299     std::set<std::string> requiredExtensions(m_deviceExtensions.begin(), m_deviceExtensions.end());
00300     for (const auto &[extensionName, specVersion] : availableExtensions) {
00301         requiredExtensions.erase(extensionName);
00302     }
00303
00304     return requiredExtensions.empty();
00305
00306     ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00307 {
00308     QueueFamilyIndices indices;
00309     uint32_t queueFamilyCount = 0;
00310     uint32_t index = 0;
00311     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00312     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00313     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00314
00315     for (const auto &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
00316         queueFamilies) {
00317         if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00318             indices.graphicsFamily = index;
00319             indices.graphicsFamilyHasValue = true;
00320         }
00321         VkBool32 presentSupport = 0U;
00322         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00323         if (queueCount > 0 && (presentSupport != 0U)) {
00324             indices.presentFamily = index;
00325             indices.presentFamilyHasValue = true;
00326         }
00327         if (indices.isComplete()) {
00328             break;
00329         }
00330         index++;
00331     }
00332     return indices;
00333 }
00334
00335     ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00336 {
00337     uint32_t formatCount = 0;
00338     uint32_t presentModeCount = 0;
00339     SwapChainSupportDetails details;
00340     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
00341
00342     vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
00343     if (formatCount != 0) {
00344         details.formats.resize(formatCount);
00345         vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
00346     }
00347     vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
00348     if (presentModeCount != 0) {
00349         details.presentModes.resize(presentModeCount);
00350         vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount,
00351             details.presentModes.data());
00352     }
00353 }
00354
00355     VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
00356         tiling, const VkFormatFeatureFlags features) const
00357 {
00358     for (const VkFormat format : candidates) {
00359         VkFormatProperties props;
00360         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00361         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00362             return format;
00363         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
00364             features) {
00365             return format;
00366         }
00367     }
00368     throw std::runtime_error("failed to find supported format!");
00369
00370     uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
00371         properties) const
00372 {
00373     VkPhysicalDeviceMemoryProperties memProperties;
00374     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
```

```

00373
00374     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00375         if (((typeFilter & (1 << i)) != 0U) &&
00376             (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00377             return i;
00378         }
00379     }
00380
00381     throw std::runtime_error("failed to find suitable m_memory type!");
00382 }
00383
00384 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
00385 { VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00386     VkBufferCreateInfo bufferInfo{};
00387     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00388     bufferInfo.size = size;
00389     bufferInfo.usage = usage;
00390     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00391
00392     if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00393         throw std::runtime_error("failed to create vertex m_buffer!");
00394     }
00395
00396     VkMemoryRequirements memRequirements;
00397     vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);
00398
00399     const VkMemoryAllocateInfo allocInfo{
00400         .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
00401         .pNext = nullptr,
00402         .allocationSize = memRequirements.size,
00403         .memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties)
00404     };
00405
00406     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00407         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00408     }
00409
00410     vkBindBufferMemory(m_device, buffer, bufferMemory, 0);
00411 }
00412
00413 VkCommandBuffer ven::Device::beginSingleTimeCommands() const
00414 {
00415     VkCommandBufferAllocateInfo allocInfo{};
00416     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00417     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00418     allocInfo.commandPool = m_commandPool;
00419     allocInfo.commandBufferCount = 1;
00420
00421     VkCommandBuffer commandBuffer = nullptr;
00422     vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);
00423
00424     VkCommandBufferBeginInfo beginInfo{};
00425     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00426     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00427
00428     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00429     return commandBuffer;
00430 }
00431
00432 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00433 {
00434     vkEndCommandBuffer(commandBuffer);
00435
00436     VkSubmitInfo submitInfo{};
00437     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00438     submitInfo.commandBufferCount = 1;
00439     submitInfo.pCommandBuffers = &commandBuffer;
00440
00441     vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
00442     vkQueueWaitIdle(m_graphicsQueue);
00443
00444     vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
00445 }
00446
00447 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
00448 size) const
00449 {
00450     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00451
00452     VkBufferCopy copyRegion{};
00453     copyRegion.srcOffset = 0; // Optional
00454     copyRegion.dstOffset = 0; // Optional
00455     copyRegion.size = size;
00456     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00457     endSingleTimeCommands(commandBuffer);

```

```
00458 }
00459
00460 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
00461   const uint32_t height, const uint32_t layerCount) const
00462 {
00463     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00464     const VkBufferImageCopy region{
00465         .bufferOffset = 0,
00466         .bufferRowLength = 0,
00467         .bufferImageHeight = 0,
00468         .imageSubresource = {
00469             .aspectMask = VK_IMAGE_ASPECT_COLOR_BIT,
00470             .mipLevel = 0,
00471             .baseArrayLayer = 0,
00472             .layerCount = layerCount
00473         },
00474         .imageOffset = {0, 0, 0},
00475         .imageExtent = {width, height, 1}
00476     };
00477     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
00478     &region);
00479     endSingleTimeCommands(commandBuffer);
00480 }
00481 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
00482   properties, VkImage &image, VkDeviceMemory &imageMemory) const
00483 {
00484     if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00485         throw std::runtime_error("failed to create image!");
00486     }
00487     VkMemoryRequirements memRequirements;
00488     vkGetImageMemoryRequirements(m_device, image, &memRequirements);
00489
00490     VkMemoryAllocateInfo allocInfo{};
00491     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00492     allocInfo.allocationSize = memRequirements.size;
00493     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00494
00495     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00496         throw std::runtime_error("failed to allocate image memory!");
00497     }
00498
00499     if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
00500         throw std::runtime_error("failed to bind image memory!");
00501     }
00502 }
00503
00504 void ven::Device::transitionImageLayout(const VkImage image, const VkFormat format, const
00505   VkImageLayout oldLayout, const VkImageLayout newLayout, const uint32_t mipLevels, const uint32_t
00506   layerCount) const {
00507     // uses an image memory barrier transition image layouts and transfer queue
00508     // family ownership when VK_SHARING_MODE_EXCLUSIVE is used. There is an
00509     // equivalent buffer memory barrier to do this for buffers
00510     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00511
00512     VkImageMemoryBarrier barrier{};
00513     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00514     barrier.oldLayout = oldLayout;
00515     barrier.newLayout = newLayout;
00516
00517     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00518     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00519
00520     barrier.image = image;
00521     barrier.subresourceRange.baseMipLevel = 0;
00522     barrier.subresourceRange.levelCount = mipLevels;
00523     barrier.subresourceRange.baseArrayLayer = 0;
00524     barrier.subresourceRange.layerCount = layerCount;
00525
00526     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00527         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00528         if (format == VK_FORMAT_D32_SFLOAT_S8_UINT || format == VK_FORMAT_D24_UNORM_S8_UINT) {
00529             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00530         }
00531     } else {
00532         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00533     }
00534
00535     VkPipelineStageFlags sourceStage = 0;
00536     VkPipelineStageFlags destinationStage = 0;
00537
00538     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00539         barrier.srcAccessMask = 0;
00540         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00541 }
```

```

00540     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00541     destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00542 } else if (
00543     oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00544     barrier.srcAccessMask = 0;
00545     barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00546
00547     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00548     destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00549 } else if (
00550     oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL &&
00551     newLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00552     barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00553     barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00554
00555     sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00556     destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00557 } else if (
00558     oldLayout == VK_IMAGE_LAYOUT_UNDEFINED &&
00559     newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00560     barrier.srcAccessMask = 0;
00561     barrier.dstAccessMask =
00562         VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00563
00564     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00565     destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00566 } else {
00567     throw std::invalid_argument("unsupported layout transition!");
00568 }
00569 vkCmdPipelineBarrier(
00570     commandBuffer,
00571     sourceStage,
00572     destinationStage,
00573     0,
00574     0,
00575     nullptr,
00576     0,
00577     nullptr,
00578     1,
00579     &barrier);
00580
00581 endSingleTimeCommands(commandBuffer);
00582 }

```

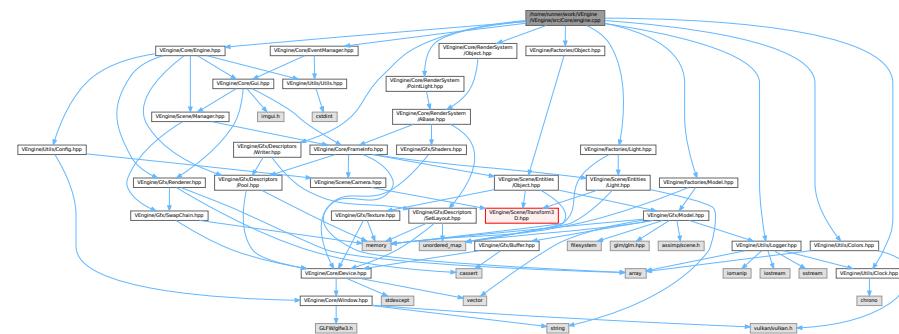
## 8.84 /home/runner/work/VEngine/VEngine/src/Core/engine.cpp File Reference

```

#include "VEngine/Core/Engine.hpp"
#include "VEngine/Core/EventManager.hpp"
#include "VEngine/Core/RenderSystem/Object.hpp"
#include "VEngine/Core/RenderSystem/PointLight.hpp"
#include "VEngine/Gfx/Descriptors/Writer.hpp"
#include "VEngine/Factories/Light.hpp"
#include "VEngine/Factories/Object.hpp"
#include "VEngine/Factories/Model.hpp"
#include "VEngine/Utils/Clock.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Utils/Logger.hpp"

```

Include dependency graph for engine.cpp:



## 8.85 engine.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Core/Engine.hpp"
00002 #include "VEngine/Core/EventManager.hpp"
00003 #include "VEngine/Core/RenderSystem/Object.hpp"
00004 #include "VEngine/Core/RenderSystem/PointLight.hpp"
00005 #include "VEngine/Gfx/Descriptors/Writer.hpp"
00006 #include "VEngine/Factories/Light.hpp"
00007 #include "VEngine/Factories/Object.hpp"
00008 #include "VEngine/Factories/Model.hpp"
00009 #include "VEngine/Utils/Clock.hpp"
00010 #include "VEngine/Utils/Colors.hpp"
00011 #include "VEngine/Utils/Logger.hpp"
00012
00013 ven::Engine::Engine(const Config& config) : m_state(EDITOR), m_window(config.window.width,
    config.window.height), m_camera(config.camera.fov, config.camera.near, config.camera.far,
    config.camera.move_speed, config.camera.look_speed) {
00014     m_gui.init(m_window.getGLFWwindow(), m_device.getInstance(), &m_device,
    m_renderer.getSwapChainRenderPass());
00015     m_globalPool =
        DescriptorPool::Builder(m_device).setMaxSets(MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    MAX_FRAMES_IN_FLIGHT).build();
00016     m_framePools.resize(MAX_FRAMES_IN_FLIGHT);
00017     const auto framePoolBuilder = DescriptorPool::Builder(m_device)
00018         .setMaxSets(1000)
00019         .addPoolSize(VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1000)
00020         .addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1000)
00021         .setPoolFlags(VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT);
00022     for (auto & framePool : m_framePools) {
00023         framePool = framePoolBuilder.build();
00024     }
00025     loadObjects();
00026 }
00027
00028 void ven::Engine::loadObjects()
00029 {
00030     constexpr std::array lightColors{Colors::RED_4, Colors::GREEN_4, Colors::BLUE_4, Colors::YELLOW_4,
    Colors::CYAN_4, Colors::MAGENTA_4};
00031     const std::unordered_map<std::string, std::shared_ptr<Model>> modelCache =
        ModelFactory::loadAll(m_device, "assets/models/");
00032     const std::shared_ptr<Texture> defaultTexture = m_sceneManager.getTextureDefault();
00033
00034     Logger::logExecutionTime("Creating object quad", [&] {
00035         m_sceneManager.addObject(ObjectFactory::create(
00036             defaultTexture,
00037             modelCache.at("assets/models/quad.obj"),
00038             "quad",
00039             {
00040                 .translation = {0.F, .5F, 0.F},
00041                 .scale = {3.F, 1.F, 3.F},
00042                 .rotation = {0.F, 0.F, 0.F}
00043             }));
00044     });
00045
00046     Logger::logExecutionTime("Creating object smooth vase", [&]{
00047         m_sceneManager.addObject(ObjectFactory::create(
00048             defaultTexture,
00049             modelCache.at("assets/models/smooth_vase.obj"),

```

```

00050         "smooth vase",
00051         {
00052             .translation = { .5F, .5F, 0.F },
00053             .scale = { 3.F, 1.5F, 3.F },
00054             .rotation = { 0.F, 0.F, 0.F }
00055         }));
00056     });
00057     Logger::logExecutionTime("Creating object flat vase", [&]{
00058         m_sceneManager.addObject(ObjectFactory::create(
00059             defaultTexture,
00060             modelCache.at("assets/models/flat_vase.obj"),
00061             "flat vase",
00062             {
00063                 .translation = { -.5F, .5F, 0.F },
00064                 .scale = { 3.F, 1.5F, 3.F },
00065                 .rotation = { 0.F, 0.F, 0.F }
00066             }));
00067     });
00068     for (std::size_t i = 0; i < lightColors.size(); i++)
00069     {
00070         Logger::logExecutionTime("Creating light n" + std::to_string(i), [&] {
00071             const glm::mat4 rotateLight = rotate(
00072                 glm::mat4(1.F),
00073                 static_cast<float>(i) * glm::two_pi<float>() / 6.0F, // 6 = num of lights
00074                 { 0.F, -1.F, 0.F }
00075         );
00076         m_sceneManager.addLight(LightFactory::create({
00077             .translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F, 1.F)),
00078             .scale = { 0.1F, 0.0F, 0.0F },
00079             .rotation = { 0.F, 0.F, 0.F },
00080             lightColors.at(i)
00081         }));
00082     });
00083 }
00084 }
00085
00086 void ven::Engine::mainLoop()
00087 {
00088     Clock clock;
00089     const EventManager eventManager{};
00090     GlobalUBO ubo{};
00091     VkCommandBuffer_T *commandBuffer = nullptr;
00092     VkDescriptorBufferInfo bufferInfo{};
00093     float frameTime = 0.0F;
00094     unsigned long frameIndex = 0;
00095     const std::unique_ptr globalSetLayout(DescriptorsetLayout::Builder(m_device).addBinding(0,
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_SHADER_STAGE_ALL_GRAPHICS).build());
00096     std::vector<std::unique_ptr<Buffer>> uboBuffers(MAX_FRAMES_IN_FLIGHT);
00097     std::vector<VkDescriptorSet> globalDescriptorSets(MAX_FRAMES_IN_FLIGHT);
00098     const ObjectRenderSystem objectRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorsetLayout());
00099     const PointLightRenderSystem pointLightRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorsetLayout());
00100
00101     for (auto& uboBuffer : uboBuffers)
00102     {
00103         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUBO), 1,
VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00104         uboBuffer->map();
00105     }
00106     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00107         bufferInfo = uboBuffers[i]->descriptorInfo();
00108         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
&bufferInfo).build(globalDescriptorSets[i]);
00109     }
00110
00111     while (m_state != EXIT)
00112     {
00113         clock.update();
00114         frameTime = clock.getDeltaTime();
00115         eventManager.handleEvents(m_window.getGLFWWindow(), &m_state, m_camera, m_gui, frameTime);
00116         commandBuffer = m_renderer.beginFrame();
00117
00118         m_camera.setViewXYZ(m_camera.transform.translation, m_camera.transform.rotation);
00119         m_camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00120
00121         if (commandBuffer != nullptr) {
00122             frameIndex = m_renderer.getFrameIndex();
00123             m_framePools[frameIndex]->resetPool();
00124             FrameInfo frameInfo{
00125                 .frameIndex=frameIndex,
00126                 .frameTime=frameTime,
00127                 .commandBuffer=commandBuffer,
00128                 .camera=m_camera,
00129                 .globalDescriptorSet=globalDescriptorSets[frameIndex],
00130                 .frameDescriptorPool=&m_framePools[frameIndex],
00131                 .objects=m_sceneManager.getObjects(),

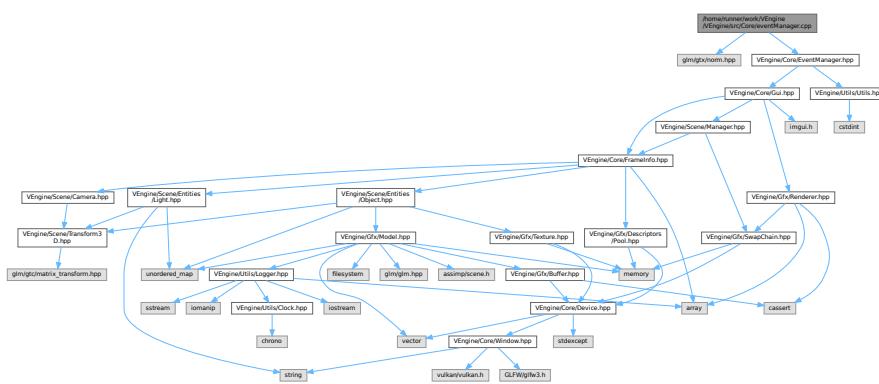
```

```

00132         .lights=m_sceneManager.getLights()
00133     };
00134     ubo.projection=m_camera.getProjection();
00135     ubo.view=m_camera.getView();
00136     ubo.inverseView=m_camera.getInverseView();
00137     m_sceneManager.updateBuffer(ubo, frameIndex, frameTime);
00138     uboBuffers.at(frameIndex)->writeToBuffer(&ubo);
00139     uboBuffers.at(frameIndex)->flush();
00140     m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00141     objectRenderSystem.render(frameInfo);
00142     pointLightRenderSystem.render(frameInfo);
00143
00144     if (m_gui.getState() != HIDDEN) {
00145         m_gui.render(
00146             &m_renderer,
00147             m_sceneManager,
00148             m_camera,
00149             m_device.getPhysicalDevice(),
00150             ubo,
00151             { .deltaTimeMS=clock.getDeltaTimeMS(), .fps=clock.getFPS() }
00152         );
00153     }
00154
00155     m_renderer.endSwapChainRenderPass(commandBuffer);
00156     m_renderer.endFrame();
00157     commandBuffer = nullptr;
00158 }
00159 if (m_sceneManager.getDestroyState()) {
00160     vkDeviceWaitIdle(m_device.device());
00161     m_sceneManager.destroyEntity(m_gui.getObjectsToRemove(), m_gui.getLightsToRemove());
00162 }
00163
00164 vkDeviceWaitIdle(m_device.device());
00165 }
```

## 8.86 /home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp File Reference

```
#include <glm/gtx/norm.hpp>
#include "VEngine/Core/EventManager.hpp"
Include dependency graph for eventManager.cpp:
```



### Macros

- #define GLM\_ENABLE\_EXPERIMENTAL

## 8.86.1 Macro Definition Documentation

### 8.86.1.1 GLM\_ENABLE\_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 1 of file `eventManager.cpp`.

## 8.87 eventManager.cpp

[Go to the documentation of this file.](#)

```
00001 #define GLM_ENABLE_EXPERIMENTAL
00002 #include <glm/gtx/norm.hpp>
00003
00004 #include "VEngine/Core/EventManager.hpp"
00005
00006 bool ven::EventManager::isKeyJustPressed(GLFWwindow* window, const long unsigned int key,
00007     std::array<bool, GLFW_KEY_LAST>& keyStates)
00008 {
00009     const bool isPressed = glfwGetKey(window, static_cast<int>(key)) == GLFW_PRESS;
00010     const bool wasPressed = keyStates.at(key);
00011
00012     keyStates.at(key) = isPressed;
00013
00014     return isPressed && !wasPressed;
00015 }
00016
00017 template<typename Iterator>
00018 void ven::EventManager::processKeyActions(GLFWwindow* window, Iterator begin, Iterator end)
00019 {
00020     for (auto it = begin; it != end; ++it) {
00021         if (glfwGetKey(window, it->key) == GLFW_PRESS) {
00022             *it->dir += it->value;
00023         }
00024     }
00025 }
00026
00027 void ven::EventManager::moveCamera(GLFWwindow* window, Camera& camera, const float dt)
00028 {
00029     glm::vec3 rotate{0};
00030     glm::vec3 moveDir{0.F};
00031     static constexpr glm::vec3 upDir{0.F, -1.F, 0.F};
00032     const float yaw = camera.transform.rotation.y;
00033     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
00034     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00035     const std::array<KeyAction, 10> moveActions = {
00036         {.key=DEFAULT_KEY_MAPPINGS.lookLeft, .dir=&rotate, .value={0.F, -1.F, 0.F}},
00037         {.key=DEFAULT_KEY_MAPPINGS.lookRight, .dir=&rotate, .value={0.F, 1.F, 0.F}},
00038         {.key=DEFAULT_KEY_MAPPINGS.lookUp, .dir=&rotate, .value={1.F, 0.F, 0.F}},
00039         {.key=DEFAULT_KEY_MAPPINGS.lookDown, .dir=&rotate, .value={-1.F, 0.F, 0.F}},
00040         {.key=DEFAULT_KEY_MAPPINGS.moveForward, .dir=&moveDir, .value=forwardDir},
00041         {.key=DEFAULT_KEY_MAPPINGS.moveBackward, .dir=&moveDir, .value=-forwardDir},
00042         {.key=DEFAULT_KEY_MAPPINGS.moveRight, .dir=&moveDir, .value=rightDir},
00043         {.key=DEFAULT_KEY_MAPPINGS.moveLeft, .dir=&moveDir, .value=-rightDir},
00044         {.key=DEFAULT_KEY_MAPPINGS.moveUp, .dir=&moveDir, .value=upDir},
00045         {.key=DEFAULT_KEY_MAPPINGS.moveDown, .dir=&moveDir, .value=-upDir}
00046     };
00047
00048     processKeyActions(window, moveActions.begin(), moveActions.end());
00049
00050     if (const float lengthRotate = length2(rotate); lengthRotate > EPSILON) {
00051         camera.transform.rotation += camera.getLookSpeed() * dt * rotate / std::sqrt(lengthRotate);
00052     }
00053     if (const float lengthMove = length2(moveDir); lengthMove > EPSILON) {
00054         camera.transform.translation += camera.getMoveSpeed() * dt * moveDir / std::sqrt(lengthMove);
00055     }
00056
00057     camera.transform.rotation.x = glm::clamp(camera.transform.rotation.x, -1.5F, 1.5F);
00058     camera.transform.rotation.y = glm::mod(camera.transform.rotation.y, glm::two_pi<float>());
00059 }
00060
00061 void ven::EventManager::handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera,
00062     Gui& gui, const float dt) const
00063 {
00064     glfwPollEvents();
00065     if (glfwWindowShouldClose(window) == GLFW_TRUE) {
00066         updateEngineState(engineState, EXIT);
00067 }
```

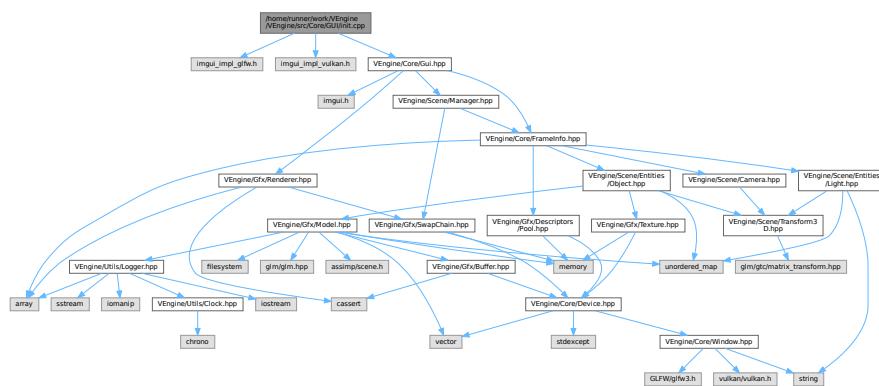
```

00065     }
00066     if (isKeyJustPressed(window, DEFAULT_KEY_MAPPINGS.toggleGui, m_keyState)) {
00067         if (gui.getState() != HIDDEN) {
00068             gui.setState(HIDDEN);
00069         } else {
00070             if (*engineState == EDITOR) {
00071                 gui.setState(SHOW_EDITOR);
00072             } else {
00073                 gui.setState(SHOW_PLAYER);
00074             }
00075         }
00076     }
00077     moveCamera(window, camera, dt);
00078 }

```

## 8.88 /home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include "VEngine/Core/Gui.hpp"
Include dependency graph for init.cpp:
```



## 8.89 init.cpp

[Go to the documentation of this file.](#)

```

00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include "VEngine/Core/Gui.hpp"
00005
00006 void ven::Gui::init(GLFWwindow* window, const VkInstance instance, const Device* device, const
00007 VkRenderPass renderPass)
00008 {
00009     VkDescriptorPool pool = nullptr;
00010     ImGui_ImplVulkan_InitInfo init_info{ };
00011     ImGui::CreateContext();
00012     m_io = &ImGui::GetIO();
00013     m_io->IniFilename = nullptr;
00014
00015     constexpr std::array<VkDescriptorPoolSize, 11> pool_sizes = {{
00016         { .type=VK_DESCRIPTOR_TYPE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00017         { .type=VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00018         { .type=VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00019         { .type=VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00020         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00021         { .type=VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00022         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00023         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00024     }};

```

```

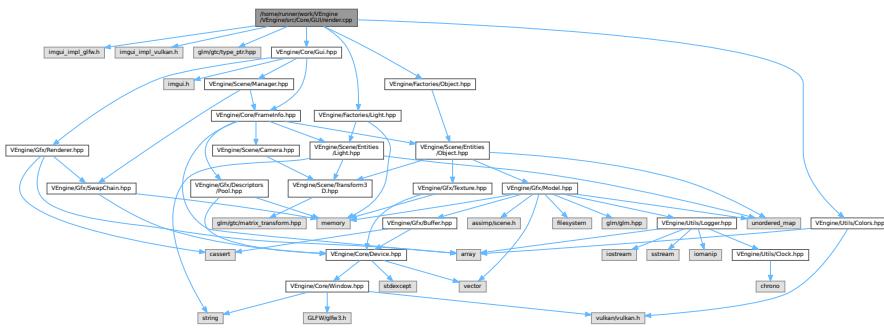
00023     { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00024     { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00025     { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
00026 };
00027 const VkDescriptorPoolCreateInfo pool_info = {
00028     VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00029     nullptr,
00030     VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00031     DESCRIPTOR_COUNT,
00032     std::size(pool_sizes),
00033     pool_sizes.data()
00034 };
00035
00036 if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00037     throw std::runtime_error("Failed to create ImGui descriptor pool");
00038 }
00039
00040 init_info.Instance = instance;
00041 init_info.PhysicalDevice = device->getPhysicalDevice();
00042 init_info.Device = device->device();
00043 init_info.Queue = device->graphicsQueue();
00044 init_info.DescriptorPool = pool;
00045 init_info.MinImageCount = 3;
00046 init_info.ImageCount = 3;
00047 init_info.MSAASamples = VK_SAMPLE_COUNT_1_BIT;
00048
00049 ImGui_ImplGlfw_InitForVulkan(window, true);
00050 ImGui_ImplVulkan_Init(&init_info, renderPass);
00051 initWithStyle();
00052 }
00053
00054 void ven::Gui::initWithStyle()
00055 {
00056     ImGuiStyle& style = ImGui::GetStyle();
00057     style.Alpha = 1.0;
00058     style.WindowRounding = 3;
00059     style.GrabRounding = 1;
00060     style.GrabMinSize = 20;
00061     style.FrameRounding = 3;
00062
00063     style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00064     style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
00065     style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
00066     style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
00067     style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
00068     style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
00069     style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
00070     style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
00071     style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
00072     style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
00073     style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
00074     style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
00075     style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
00076     style.Colors[ImGuiCol_ScrollbarGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.44F);
00077     style.Colors[ImGuiCol_ScrollbarGrabHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00078     style.Colors[ImGuiCol_ScrollbarGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00079     style.Colors[ImGuiCol_CheckMark] = ImVec4(0.00F, 1.00F, 1.00F, 0.68F);
00080     style.Colors[ImGuiCol_SliderGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.36F);
00081     style.Colors[ImGuiCol_SliderGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.76F);
00082     style.Colors[ImGuiCol_Button] = ImVec4(0.00F, 0.65F, 0.65F, 0.46F);
00083     style.Colors[ImGuiCol_ButtonHovered] = ImVec4(0.01F, 1.00F, 1.00F, 0.43F);
00084     style.Colors[ImGuiCol_ButtonActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.62F);
00085     style.Colors[ImGuiCol_Header] = ImVec4(0.00F, 1.00F, 1.00F, 0.33F);
00086     style.Colors[ImGuiCol_HeaderHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.42F);
00087     style.Colors[ImGuiCol_HeaderActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00088     style.Colors[ImGuiCol_ResizeGrip] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00089     style.Colors[ImGuiCol_ResizeGripHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00090     style.Colors[ImGuiCol_ResizeGripActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00091     style.Colors[ImGuiCol_PlotLines] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00092     style.Colors[ImGuiCol_PlotLinesHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00093     style.Colors[ImGuiCol_PlotHistogram] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00094     style.Colors[ImGuiCol_PlotHistogramHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00095     style.Colors[ImGuiCol_TextSelectedBg] = ImVec4(0.00F, 1.00F, 1.00F, 0.22F);
00096 }

```

## 8.90 /home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
```

```
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Factories/Object.hpp"
#include "VEngine/Factories/Light.hpp"
Include dependency graph for render.cpp:
```



## 8.91 render.cpp

[Go to the documentation of this file.](#)

```
00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include <glm/gtc/type_ptr.hpp>
00005
00006 #include "VEngine/Core/Gui.hpp"
00007 #include "VEngine/Utils/Colors.hpp"
00008 #include "VEngine/Factories/Object.hpp"
00009 #include "VEngine/Factories/Light.hpp"
0010
0011 void ven::Gui::cleanup()
0012 {
0013     ImGui_ImplVulkan_Shutdown();
0014     ImGui_ImplGlfw_Shutdown();
0015     ImGui::DestroyContext();
0016 }
0017
0018 void ven::Gui::render(Renderer* renderer, SceneManager& sceneManager, Camera& camera, const
    VkPhysicalDevice physicalDevice, GlobalUbo& ubo, const ClockData& clockData)
0019 {
0020     VkPhysicalDeviceProperties deviceProperties;
0021     vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
0022     ImGui_ImplVulkan_NewFrame();
0023     ImGui_ImplGlfw_NewFrame();
0024     ImGui::NewFrame();
0025     renderFrameWindow(clockData);
0026
0027     rendererSection(renderer, ubo);
0028     cameraSection(camera);
0029     lightsSection(sceneManager);
0030     objectsSection(sceneManager);
0031     inputsSection(*m_io);
0032     devicePropertiesSection(deviceProperties);
0033
0034     ImGui::End();
0035     ImGui::Render();
0036     ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
0037 }
0038
0039 void ven::Gui::renderFrameWindow(const ClockData& clockData)
0040 {
0041     ImGui::SetNextWindowPos(ImVec2(0.0F, 0.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
0042     ImGui::Begin("Application Info", nullptr, ImGuiWindowFlags_NoDecoration | ImGuiWindowFlags_NoMove
        | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoSavedSettings | ImGuiWindowFlags_NoFocusOnAppearing |
        ImGuiWindowFlags_NoNav);
0043     ImGui::Text("FPS: %.1f", clockData.fps);
0044     ImGui::Text("Frame time: %.3fms", clockData.deltaTimeMS);
0045     ImGui::End();
0046 }
```

```

00047
00048 void ven::Gui::rendererSection(Renderer *renderer, GlobalUBo& ubo)
00049 {
00050     ImGui::SetNextWindowPos(ImVec2(0.0F, 45.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
00051     ImGui::Begin("Editor tools");
00052     if (ImGui::CollapsingHeader("Renderer")) {
00053         ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());
00054
00055         if (ImGui::BeginTable("ClearColorTable", 2)) {
00056             ImGui::TableNextColumn();
00057             std::array<float, 4> clearColor = renderer->getClearColor();
00058
00059             if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
00060                 const VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1],
00061                     clearColor[2], clearColor[3]}};
00062                 renderer->setClearColorValue(clearColorValue);
00063             }
00064
00065             ImGui::TableNextColumn();
00066             static int item_current = 0;
00067
00068             if (ImGui::Combo("Color Presets## clearColor",
00069                             &item_current,
00070                             [] (void*, const int idx, const char** out_text) -> bool {
00071                                 if (idx < 0 || idx >=
00072                                     static_cast<int>(std::size(Colors::COLOR_PRESETS_VK))) { return false; }
00073                                 *out_text = Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
00074                                     long>(idx)).first;
00075                                 return true;
00076                             },
00077                             nullptr,
00078                             std::size(Colors::COLOR_PRESETS_VK))) {
00079                 renderer->setClearColorValue(Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
00080                                     long>(item_current)).second);
00081             }
00082
00083             ImGui::TableNextColumn();
00084             ImGui::ColorEdit4("Ambient Light Color", glm::value_ptr(ubo.ambientLightColor));
00085             ImGui::TableNextColumn();
00086             if (ImGui::Combo("Color Presets## ambientColor",
00087                             &item_current,
00088                             [] (void*, const int idx, const char** out_text) -> bool {
00089                                 if (idx < 0 || idx >=
00090                                     static_cast<int>(std::size(Colors::COLOR_PRESETS_4))) { return false; }
00091                                 *out_text = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
00092                                     long>(idx)).first;
00093                                 return true;
00094                             },
00095                             nullptr,
00096                             std::size(Colors::COLOR_PRESETS_4))) {
00097                 ubo.ambientLightColor = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
00098                                     long>(item_current)).second;
00099             }
00100
00101             ImGui::TableNextColumn();
00102             ImGui::SliderFloat(("Intensity##" + std::to_string(0)).c_str(), &ubo.ambientLightColor.a,
00103                 0.0F, 1.0F);
00104             ImGui::TableNextColumn();
00105             if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a =
00106                 DEFAULT_AMBIENT_LIGHT_INTENSITY; }
00107         }
00108
00109 void ven::Gui::cameraSection(Camera &camera)
00110 {
00111     if (ImGui::CollapsingHeader("Camera")) {
00112         float fov = camera.getFov();
00113         float tnear = camera.getNear();
00114         float tfar = camera.getFar();
00115         if (ImGui::BeginTable("CameraTable", 2)) {
00116             ImGui::TableNextColumn();
00117             ImGui::DragFloat3("Position", glm::value_ptr(camera.transform.translation), 0.1F);
00118             ImGui::TableNextColumn();
00119             if (ImGui::Button("Reset##position")) { camera.transform.translation = DEFAULT_POSITION; }
00120
00121             ImGui::TableNextColumn();
00122             ImGui::DragFloat3("Rotation", glm::value_ptr(camera.transform.rotation), 0.1F);
00123             ImGui::TableNextColumn();

```

```

00124         if (ImGui::Button("Reset##rotation")) { camera.transform.rotation = DEFAULT_ROTATION; }
00125
00126         ImGui::TableNextColumn();
00127         if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
00128             camera.setFov(fov); }
00129             ImGui::TableNextColumn();
00130             if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }
00131
00132             ImGui::TableNextColumn();
00133             if (ImGui::SliderFloat("Near", &tnear, 0.001F, 10.0F)) { camera.setNear(tnear); }
00134             ImGui::TableNextColumn();
00135             if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }
00136
00137             ImGui::TableNextColumn();
00138             if (ImGui::SliderFloat("Far", &tfar, 1.F, 1000.0F)) { camera.setFar(tfar); }
00139             ImGui::TableNextColumn();
00140             if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }
00141
00142             ImGui::TableNextColumn();
00143             float moveSpeed = camera.getMoveSpeed();
00144             if (ImGui::SliderFloat("Move speed", &moveSpeed, 0.1F, 10.0F)) {
00145                 camera.setMoveSpeed(moveSpeed); }
00146                 ImGui::TableNextColumn();
00147                 if (ImGui::Button("Reset##moveSpeed")) { camera.setMoveSpeed(DEFAULT_MOVE_SPEED); }
00148
00149                 ImGui::TableNextColumn();
00150                 float lookSpeed = camera.getLookSpeed();
00151                 if (ImGui::SliderFloat("Look speed", &lookSpeed, 0.1F, 10.0F)) {
00152                     camera.setLookSpeed(lookSpeed); }
00153
00154             ImGui::EndTable(); }
00155     }
00156 }
00157
00158 void ven::Gui::objectsSection(SceneManager& sceneManager)
00159 {
00160     if (ImGui::CollapsingHeader("Objects")) {
00161         bool open = false;
00162         for (Object::Map& objects = sceneManager.getObjects(); auto& [id, object] : objects) {
00163             ImGui::PushStyleColor(ImGuiCol_Text, { Colors::GRAY_4.r, Colors::GRAY_4.g,
00164             Colors::GRAY_4.b, 1.0F });
00165             open = ImGui::TreeNode(std::string(object.getName() + " [" +
00166             std::to_string(object.getId()) + "]").c_str());
00167             ImGui::PopStyleColor(1);
00168             if (open) {
00169                 if (ImGui::Button(("Delete##" + object.getName()).c_str())) {
00170                     m_objectsToRemove.push_back(id);
00171                     sceneManager.setDestroyState(true);
00172                 }
00173                 ImGui::SameLine();
00174                 if (ImGui::Button(("Duplicate##" + object.getName()).c_str())) {
00175                     ObjectFactory::duplicate(object);
00176                 }
00177                 ImGui::Text("Address: %p", static_cast<void*>(&object));
00178                 ImGui::DragFloat3(("Position##" + object.getName()).c_str(),
00179                 glm::value_ptr(object.transform.translation), 0.1F);
00180                 ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(),
00181                 glm::value_ptr(object.transform.rotation), 0.1F);
00182                 ImGui::DragFloat3(("Scale##" + object.getName()).c_str(),
00183                 glm::value_ptr(object.transform.scale), 0.1F);
00184             }
00185         }
00186     }
00187
00188     if (ImGui::CollapsingHeader("Lights")) {
00189         bool open = false;
00190         float tempIntensity = m_intensity;
00191         float tempShininess = m_shininess;
00192         Light::Map& lights = sceneManager.getLights();
00193
00194         if (ImGui::BeginTable("LightTable", 2)) {
00195             ImGui::TableNextColumn();
00196             if (ImGui::SliderFloat("Global Intensity", &tempIntensity, 0.0F, 5.F)) {
00197                 m_intensity = tempIntensity;
00198                 for (auto& [fst, snd] : lights) {
00199                     snd.color.a = m_intensity;
00200                 }
00201             }
00202         }
00203     }

```

```

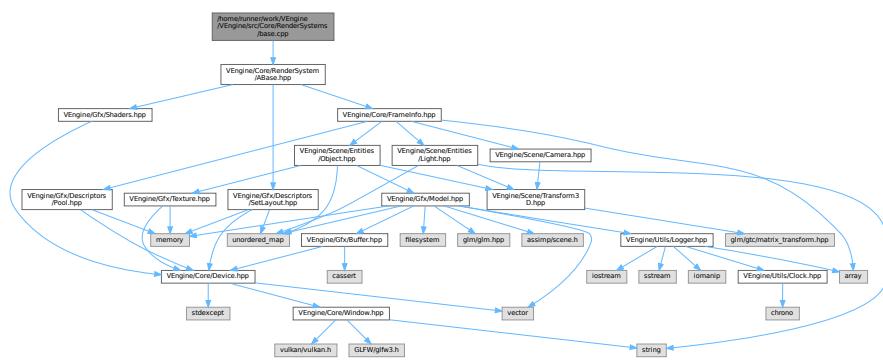
00203         if (ImGui::Button("Reset")) {
00204             m_intensity = DEFAULT_LIGHT_INTENSITY;
00205             tempIntensity = m_intensity;
00206             for (auto&[fst, snd] : lights) {
00207                 snd.color.a = m_intensity;
00208             }
00209         }
00210
00211         ImGui::TableNextColumn();
00212         if (ImGui::SliderFloat("Global Shininess", &tempShininess, 0.0F, 512.F)) {
00213             m_shininess = tempShininess;
00214             for (auto&[fst, snd] : lights) {
00215                 snd.setShininess(m_shininess);
00216             }
00217         }
00218
00219         ImGui::TableNextColumn();
00220         if (ImGui::Button("Reset")) {
00221             m_shininess = DEFAULT_SHININESS;
00222             tempShininess = m_shininess;
00223             for (auto&[fst, snd] : lights) {
00224                 snd.setShininess(m_shininess);
00225             }
00226         }
00227
00228         ImGui::EndTable();
00229     }
00230
00231     for (auto& [id, light] : lights) {
00232         ImGui::PushStyleColor(ImGuiCol_Text, {light.color.r, light.color.g, light.color.b, 1.0F});
00233         open = ImGui::TreeNode(std::string(light.getName() + "[" + std::to_string(light.getId())
00234         + "]").c_str());
00235         ImGui::PopStyleColor(1);
00236         if (open) {
00237             if (ImGui::Button(("Delete##" + light.getName()).c_str())) {
00238                 m_lightsToRemove.push_back(id);
00239                 sceneManager.setDestroyState(true);
00240             }
00241             ImGui::SameLine();
00242             if (ImGui::Button(("Duplicate##" + light.getName()).c_str())) {
00243                 LightFactory::duplicate(light);
00244             }
00245             ImGui::Text("Address: %p", static_cast<void*>(&light));
00246             ImGui::DragFloat3(("Position##" + std::to_string(light.getId())).c_str(),
00247                               glm::value_ptr(light.transform.translation), 0.1F);
00248             ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId())).c_str(),
00249                               glm::value_ptr(light.transform.rotation), 0.1F);
00250             ImGui::DragFloat3(("Scale##" + std::to_string(light.getId())).c_str(),
00251                               glm::value_ptr(light.transform.scale), 0.1F);
00252             if (ImGui::BeginTable("ColorTable", 2)) {
00253                 ImGui::TableNextColumn();
00254                 ImGui::ColorEdit4(("Color##" + std::to_string(light.getId())).c_str(),
00255                               glm::value_ptr(light.color));
00256                 ImGui::TableNextColumn();
00257                 static int item_current = 0;
00258                 if (ImGui::Combo("Color Presets",
00259                                 [&item_current,
00260                                     [](void*, const int idx, const char** out_text) -> bool {
00261                                         if (idx < 0 || idx >=
00262                                             static_cast<int>(std::size(Colors::COLOR_PRESETS_3))) { return false; }
00263                                         *out_text = Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00264                                         long>(idx)).first;
00265                                         return true;
00266                                     },
00267                                     nullptr,
00268                                     std::size(Colors::COLOR_PRESETS_3))) {
00269                             light.color = {Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00270                                         long>(item_current)).second, light.color.a};
00271                         }
00272                     ImGui::EndTable();
00273
00274                     ImGui::SliderFloat(("Intensity##" + std::to_string(light.getId())).c_str(),
00275                               &light.color.a, 0.0F, 5.F);
00276                     ImGui::SameLine();
00277                     if (ImGui::Button(("Reset##" + std::to_string(light.getId())).c_str())) {
00278                         light.color.a = DEFAULT_LIGHT_INTENSITY; }
00279                     float shininess = light.getShininess();
00280                     if (ImGui::SliderFloat("Shininess", &shininess, 0.0F, 512.F)) {
00281                         light.setShininess(shininess);
00282                     }
00283                     ImGui::SameLine();
00284                     if (ImGui::Button("Reset##shininess")) { light.setShininess(DEFAULT_SHININESS); }
00285                 }
00286             ImGui::TreePop();
00287         }
00288     }

```

```
00280 }
00281
00282 void ven::Gui::inputsSection(const ImGuiIO& io)
00283 {
00284     if (ImGui::CollapsingHeader("Input")) {
00285         ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io.MousePos.x, io.MousePos.y) :
00286             ImGui::Text("Mouse pos: <INVALID>");
00287         ImGui::Text("Mouse delta: (%g, %g)", io.MouseDelta.x, io.MouseDelta.y);
00288         ImGui::Text("Mouse down:");
00289         for (int i = 0; i < static_cast<int>(std::size(io.MouseDown)); i++) {
00290             if (ImGui::IsMouseDown(i)) {
00291                 ImGui::SameLine();
00292                 ImGui::Text("b%d (%.02f secs)", i, io.MouseDownDuration[i]);
00293             }
00294         }
00295         ImGui::Text("Mouse wheel: %.1f", io.MouseWheel);
00296         ImGui::Text("Keys down:");
00297         for (auto key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
00298             static_cast<ImGuiKey>(key + 1)) {
00299             if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
00300             ImGui::SameLine();
00301             ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\\""%s\"" : "\"%s\" %d",
00302                         ImGui::GetKeyName(key), key);
00303         }
00304     }
00305 }
00306     if (ImGui::CollapsingHeader("Device Properties")) {
00307         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00308             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00309             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
00310 VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion));
00311             ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00312             ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00313             ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00314             ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00315             ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00316             ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantsSize);
00317             ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00318             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00319             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",
deviceProperties.limits.maxImageDimension2D);
00320             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00321             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00322             ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00323             ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00324             ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00325             ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00326             ImGui::EndTable();
00327     }
00328 }
00329 }
```

## 8.92 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp File Reference

```
#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for base.cpp:
```



## 8.93 base.cpp

[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Core/RenderSystem/ABase.hpp"
00002
00003 void ven::ARenderSystemBase::createPipelineLayout(const VkDescriptorSetLayout globalsetLayout, const
00004 uint32_t pushConstantSize)
00005 {
00006     VkPushConstantRange pushConstantRange{};
00007     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00008     pushConstantRange.offset = 0;
00009     pushConstantRange.size = pushConstantSize;
00010
00011     renderSystemLayout =
00012         DescriptorsetLayout::Builder(m_device)
00013             .addBinding(
00014                 0,
00015                 VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
00016                 VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT)
00017             .addBinding(1, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_SHADER_STAGE_FRAGMENT_BIT)
00018             .build();
00019
00020     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{
00021         globalsetLayout,
00022         renderSystemLayout->getDescriptorsetLayout() };
00023
00024     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00025     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00026     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00027     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00028     pipelineLayoutInfo.pushConstantRangeCount = 1;
00029     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00030     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
00031         VK_SUCCESS)
00032     {
00033         throw std::runtime_error("Failed to create pipeline layout");
00034     }
00035
00036 void ven::ARenderSystemBase::createPipeline(const VkRenderPass renderPass, const std::string
00037     &shadersVertPath, const std::string &shadersFragPath, const bool isLight)
00038 {
00039     assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
00040     PipelineConfigInfo pipelineConfig{};
00041     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00042     if (isLight) {
00043         pipelineConfig.attributeDescriptions.clear();
00044         pipelineConfig.bindingDescriptions.clear();
00045     }
}
```

```

00044     pipelineConfig.renderPass = renderPass;
00045     pipelineConfig.pipelineLayout = m_pipelineLayout;
00046     m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
00047 }

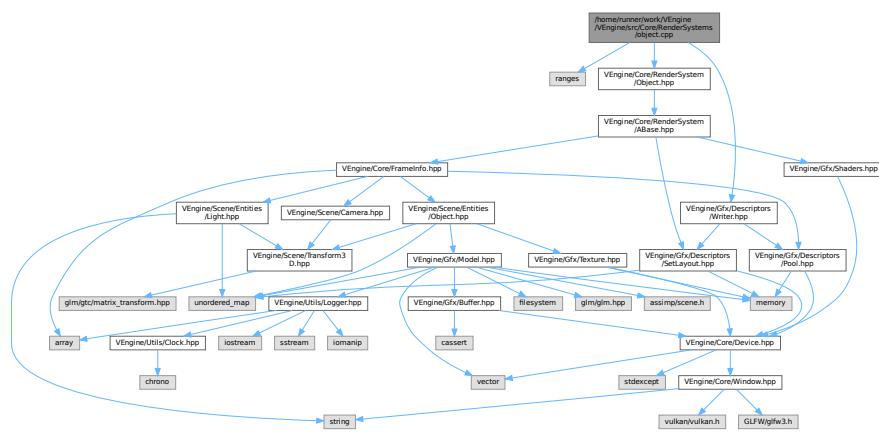
```

## 8.94 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp File Reference

```

#include <ranges>
#include "VEngine/Gfx/Descriptors/Writer.hpp"
#include "VEngine/Core/RenderSystem/Object.hpp"
Include dependency graph for object.cpp:

```



## 8.95 object.cpp

Go to the documentation of this file.

```

00001 #include <ranges>
00002
00003 #include "VEngine/Gfx/Descriptors/Writer.hpp"
00004 #include "VEngine/Core/RenderSystem/Object.hpp"
00005
00006 void ven::ObjectRenderSystem::render(const FrameInfo &frameInfo) const
00007 {
00008     getShaders() -> bind(frameInfo.commandBuffer);
00009
00010     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
00011     getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00012
00013     for (Object& object : frameInfo.objects | std::views::values) {
00014         if (object.getModel() == nullptr) { continue; }
00015         auto bufferInfo = object.getBufferInfo(static_cast<int>(frameInfo.frameIndex));
00016         auto imageInfo = object.getDiffuseMap() -> getImageInfo();
00017         VkDescriptorSet objectDescriptorSet = nullptr;
00018         DescriptorWriter(*renderSystemLayout, frameInfo.frameDescriptorPool)
00019             .writeBuffer(0, &bufferInfo)
00020             .writeImage(1, &imageInfo)
00021             .build(objectDescriptorSet);
00022
00023         vkCmdBindDescriptorSets(
00024             frameInfo.commandBuffer,
00025             VK_PIPELINE_BIND_POINT_GRAPHICS,
00026             getPipelineLayout(),
00027             1, // starting set (0 is the globalDescriptorSet, 1 is the set specific to this system)
00028             1, // set count
00029             &objectDescriptorSet,
00030             0,
00031             nullptr);

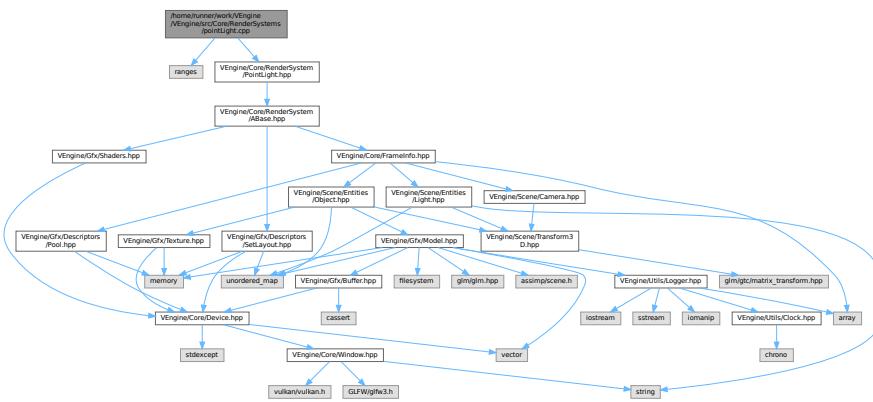
```

```

00031     const ObjectPushConstantData push{
00032         .modelMatrix = object.transform.transformMatrix(),
00033         .normalMatrix = object.transform.normalMatrix()
00034     };
00035 }
00036 vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(ObjectPushConstantData), &push);
00037 object.getModel() ->bind(frameInfo.commandBuffer);
00038 object.getModel() ->draw(frameInfo.commandBuffer);
00039 }
00040 }
```

## 8.96 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp File Reference

```
#include <ranges>
#include "VEngine/Core/RenderSystem/PointLight.hpp"
Include dependency graph for pointLight.cpp:
```



## 8.97 pointLight.cpp

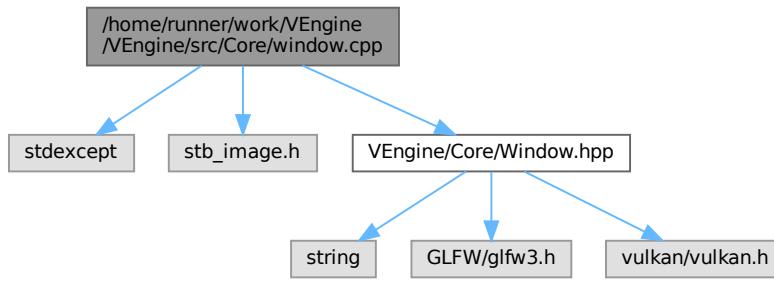
[Go to the documentation of this file.](#)

```

00001 #include <ranges>
00002
00003 #include "VEngine/Core/RenderSystem/PointLight.hpp"
00004
00005 void ven::PointLightRenderSystem::render(const FrameInfo &frameInfo) const
00006 {
00007     getShaders() ->bind(frameInfo.commandBuffer);
00008     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
00009     getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00010     for (const Light &light : frameInfo.lights | std::views::values) {
00011         const LightPushConstantData push{
00012             .position = glm::vec4(light.transform.translation, 1.F),
00013             .color = light.color,
00014             .radius = light.transform.scale.x
00015         };
00016         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
00017         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00018     }
00019 }
```

## 8.98 /home/runner/work/VEngine/VEngine/src/Core/window.cpp File Reference

```
#include <stdexcept>
#include "stb_image.h"
#include "VEngine/Core/Window.hpp"
Include dependency graph for window.cpp:
```



### Macros

- `#define STB_IMAGE_IMPLEMENTATION`

#### 8.98.1 Macro Definition Documentation

##### 8.98.1.1 STB\_IMAGE\_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

Definition at line 3 of file [window.cpp](#).

## 8.99 window.cpp

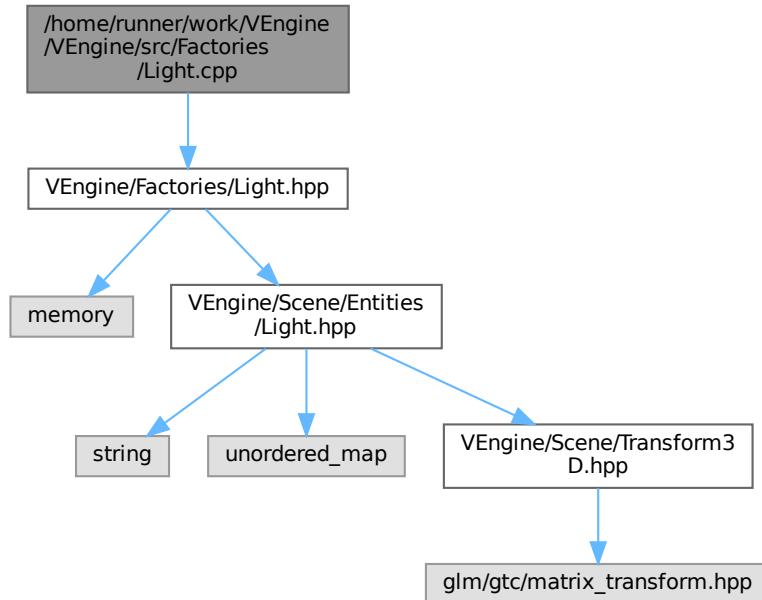
[Go to the documentation of this file.](#)

```
00001 #include <stdexcept>
00002
00003 #define STB_IMAGE_IMPLEMENTATION
00004 #include "stb_image.h"
00005
00006 #include "VEngine/Core/Window.hpp"
00007
00008 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string &title)
00009 {
00010     if (glfwInit() == GLFW_FALSE) {
00011         throw std::runtime_error("Failed to initialize GLFW");
00012     }
00013
00014     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
00015     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00016
00017     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
00018                                         title.c_str(), nullptr, nullptr);
```

```
00018     if (window == nullptr) {
00019         glfwTerminate();
00020         throw std::runtime_error("Failed to create window");
00021     }
00022     glfwSetWindowUserPointer(window, this);
00023     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
00024     return window;
00025 }
00026
00027 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
00028 {
00029     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
00030         throw std::runtime_error("Failed to create window surface");
00031     }
00032 }
00033
00034 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
00035 {
00036     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
00037     app->m_framebufferResized = true;
00038     app->m_width = static_cast<uint32_t>(width);
00039     app->m_height = static_cast<uint32_t>(height);
00040 }
00041
00042 void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
00043 {
00044     GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
00045     const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);
00046
00047     /*
00048     if (fullscreen) {
00049         glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
00050         mode->refreshRate);
00051     } else {
00052         // To restore a window that was originally windowed to its original size and position,
00053         // save these before making it full screen and then pass them in as above
00054         glfwSetWindowMonitor(m_window, nullptr, 0, 0, static_cast<int>(width),
00055         static_cast<int>(height), mode->refreshRate);
00056     }
00057     m_width = width;
00058     m_height = height;
00059     */
00060 }
00061
00062 void ven::Window::setWindowIcon(const std::string &path)
00063 {
00064     int width, height, channels;
00065
00066     if (unsigned char *pixels = stbi_load(path.c_str(), &width, &height, &channels, 4)) {
00067         GLFWimage icon;
00068         icon.width = width;
00069         icon.height = height;
00070         icon.pixels = pixels;
00071
00072         glfwSetWindowIcon(m_window, 1, &icon);
00073         stbi_image_free(pixels);
00074     } else {
00075         throw std::runtime_error("Failed to load window icon with path: " + path);
00076     }
00077 }
```

## 8.100 /home/runner/work/VEngine/VEngine/src/Factories/Light.cpp File Reference

```
#include "VEngine/Factories/Light.hpp"
Include dependency graph for Light.cpp:
```



## 8.101 Light.cpp

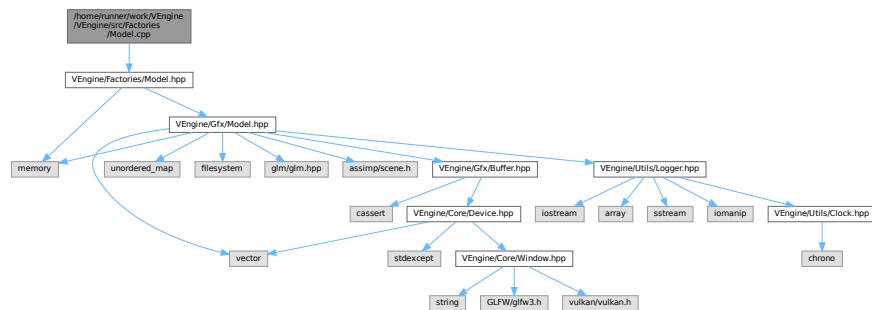
[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Factories/Light.hpp"
00002
00003 unsigned int ven::LightFactory::m_currentLightId = 0;
00004
00005 std::unique_ptr<ven::Light> ven::LightFactory::create(const Transform3D &transform, const glm::vec4
00006   color)
00007 {
00008     assert(m_currentLightId < MAX_LIGHTS && "Max light count exceeded!");
00009     auto light = std::make_unique<Light>(++m_currentLightId);
00010     light->color = color;
00011     light->transform = transform;
00012     return light;
00013 }
```

## 8.102 /home/runner/work/VEngine/VEngine/src/Factories/Model.cpp File Reference

```
#include "VEngine/Factories/Model.hpp"
Include dependency graph for Model.cpp:
```



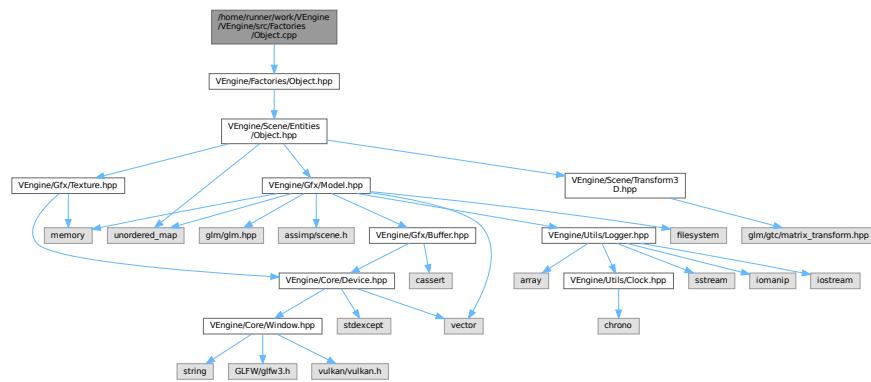
## 8.103 Model.cpp

[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Factories/Model.hpp"
00002
00003 std::unordered_map<std::string, std::shared_ptr<ven::Model>> ven::ModelFactory::loadAll(Device& device,
00004   const std::string& folderPath)
00005 {
00006     std::unordered_map<std::string, std::shared_ptr<Model>> modelCache;
00007
00008     for (const auto &entry : std::filesystem::directory_iterator(folderPath)) {
00009         if (entry.is_regular_file()) {
00010             Logger::logExecutionTime("Creating model " + entry.path().string(), [&]() {
00011                 const std::string &filepath = entry.path().string();
00012                 modelCache[filepath] = create(device, filepath);
00013             });
00014         } else {
00015             Logger::logWarning("Skipping non-regular file " + entry.path().string());
00016         }
00017     }
00018     return modelCache;
00019 }
00020 std::unique_ptr<ven::Model> ven::ModelFactory::create(Device& device, const std::string& filepath)
00021 {
00022     Model::Builder builder{};
00023     builder.loadModel(filepath);
00024     return std::make_unique<Model>(device, builder);
00025 }
```

## 8.104 /home/runner/work/VEngine/VEngine/src/Factories/Object.cpp File Reference

```
#include "VEngine/Factories/Object.hpp"
Include dependency graph for Object.cpp:
```



## 8.105 Object.cpp

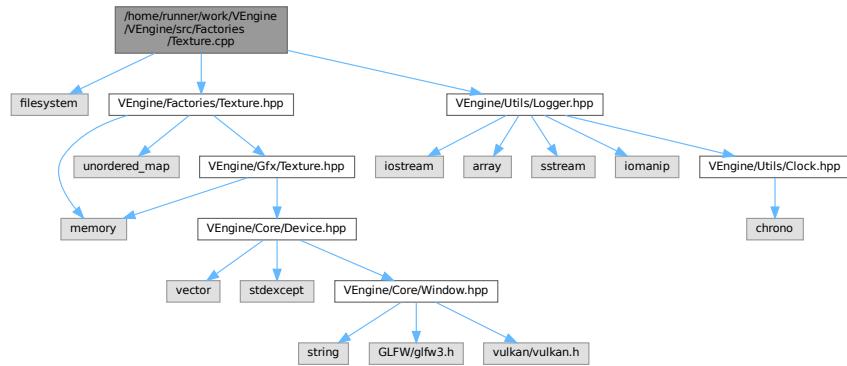
[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Factories/Object.hpp"
00002
00003 unsigned int ven::ObjectFactory::m_currentObjId = 0;
00004
00005 std::unique_ptr<ven::Object> ven::ObjectFactory::create(const std::shared_ptr<Texture>& texture, const
00006 std::shared_ptr<Model>& model, const std::string &name, const Transform3D &transform)
00007 {
00008     assert(m_currentObjId < MAX_OBJECTS && "Max object count exceeded!");
00009     auto object = std::make_unique<Object>(++m_currentObjId);
00010     object->setDiffuseMap(texture);
00011     object->setModel(model);
00012     object->setName(name);
00013     object->transform = transform;
00014     return object;
00015 }
```

## 8.106 /home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp File Reference

```
#include <filesystem>
#include "VEngine/Factories/Texture.hpp"
#include "VEngine/Utils/Logger.hpp"
```

Include dependency graph for Texture.cpp:



## 8.107 Texture.cpp

[Go to the documentation of this file.](#)

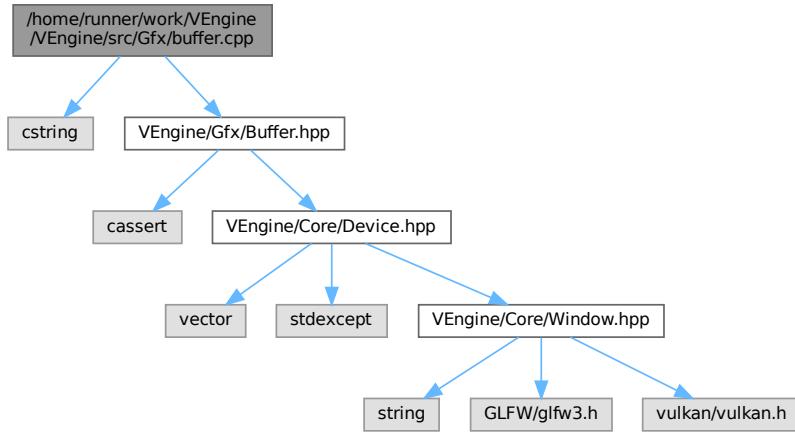
```

00001 #include <filesystem>
00002
00003 #include "VEngine/Factories/Texture.hpp"
00004 #include "VEngine/Utils/Logger.hpp"
00005
00006 std::unordered_map<std::string, std::shared_ptr<ven::Texture>> ven::TextureFactory::loadAll(Device&
device, const std::string& folderPath)
00007 {
00008     std::unordered_map<std::string, std::shared_ptr<Texture>> modelCache;
00009
00010     for (const auto &entry : std::filesystem::directory_iterator(folderPath)) {
00011         if (entry.is_regular_file()) {
00012             Logger::logExecutionTime("Creating texture " + entry.path().string(), [&]() {
00013                 const std::string &filepath = entry.path().string();
00014                 modelCache[filepath] = create(device, filepath);
00015             });
00016         } else {
00017             Logger::logWarning("Skipping non-regular file " + entry.path().string());
00018         }
00019     }
00020     return modelCache;
00021 }
```

## 8.108 /home/runner/work/VEngine/VEngine/src/Gfx/Buffer.cpp File Reference

```
#include <cstring>
#include "VEngine/Gfx/Buffer.hpp"
```

Include dependency graph for buffer.cpp:



## 8.109 buffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cstring>
00002
00003 #include "VEngine/Gfx/Buffer.hpp"
00004
00005 ven::Buffer::Buffer(Device &device, const VkDeviceSize instanceSize, const uint32_t instanceCount,
00006   const VkBufferUsageFlags usageFlags, const VkMemoryPropertyFlags memoryPropertyFlags, const
00007   VkDeviceSize minOffsetAlignment) : m_device{device}, m_instanceSize{instanceSize},
00008   m_instanceCount{instanceCount}, m_alignmentSize{getAlignment(instanceSize, minOffsetAlignment)},
00009   m_usageFlags{usageFlags}, m_memoryPropertyFlags{memoryPropertyFlags}
00010 {
00011     m_bufferSize = m_alignmentSize * m_instanceCount;
00012     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00013 }
00014
00015 ven::Buffer::~Buffer()
00016 {
00017     unmmap();
00018     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00019     vkFreeMemory(m_device.device(), m_memory, nullptr);
00020 }
00021
00022 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00023 {
00024     assert(m_buffer && m_memory && "Called map on buffer before create");
00025     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00026 }
00027
00028 void ven::Buffer::unmap()
00029 {
00030     if (m_mapped != nullptr) {
00031         vkUnmapMemory(m_device.device(), m_memory);
00032         m_mapped = nullptr;
00033     }
00034 }
00035
00036 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
00037 {
00038     const
00039     assert(m_mapped && "Cannot copy to unmapped buffer");
00040     if (size == VK_WHOLE_SIZE) {
00041         memcpy(m_mapped, data, m_bufferSize);
00042     } else {
00043         auto *memOffset = static_cast<char *>(m_mapped);
00044         memOffset += offset;
00045         memcpy(memOffset, data, size);
00046     }
  
```

```

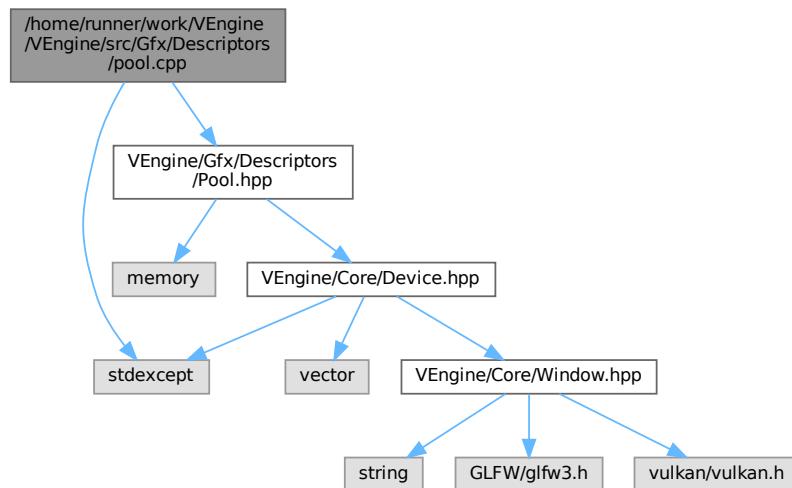
00043 }
00044
00045 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00046 {
00047     VkMappedMemoryRange mappedRange = {};
00048     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00049     mappedRange.memory = m_memory;
00050     mappedRange.offset = offset;
00051     mappedRange.size = size;
00052     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00053 }
00054
00055 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00056 {
00057     VkMappedMemoryRange mappedRange = {};
00058     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00059     mappedRange.memory = m_memory;
00060     mappedRange.offset = offset;
00061     mappedRange.size = size;
00062     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00063 }

```

## 8.110 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp

### File Reference

```
#include <stdexcept>
#include "VEngine/Gfx/Descriptors/Pool.hpp"
Include dependency graph for pool.cpp:
```



## 8.111 pool.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002
00003 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00004
00005 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device(device)
00006 {

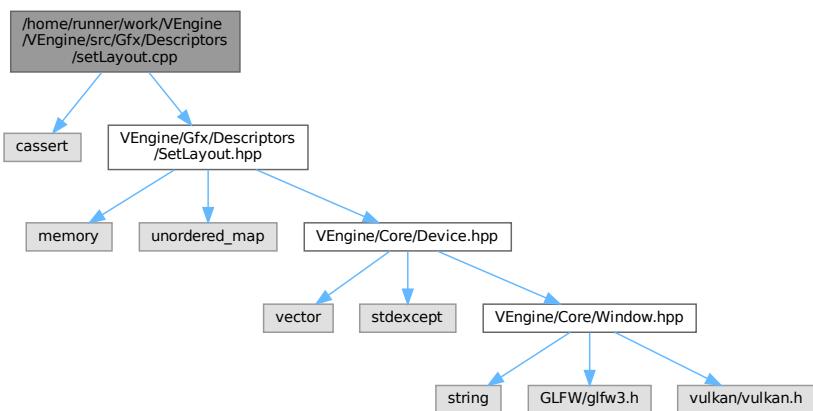
```

```

00007     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00008     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00009     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00010     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00011     descriptorPoolInfo.maxSets = maxSets;
00012     descriptorPoolInfo.flags = poolFlags;
00013
00014     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) != VK_SUCCESS) {
00015         throw std::runtime_error("failed to create descriptor pool!");
00016     }
00017 }
00018 }
00019
00020 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorsetLayout,
00021     VkDescriptorSet &descriptor) const
00022 {
00023     VkDescriptorSetAllocateInfo allocInfo{};
00024     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00025     allocInfo.descriptorPool = m_descriptorPool;
00026     allocInfo.pSetLayouts = &descriptorsetLayout;
00027     allocInfo.descriptorSetCount = 1;
00028
00029     // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00030     // a new pool whenever an old pool fills up. But this is beyond our current scope
00031     return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00032 }
```

## 8.112 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/setLayout.cpp File Reference

```
#include <cassert>
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
Include dependency graph for setLayout.cpp:
```



## 8.113 setLayout.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00004
00005 ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t
00006     binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t
00007     count)
00008 {
```

```

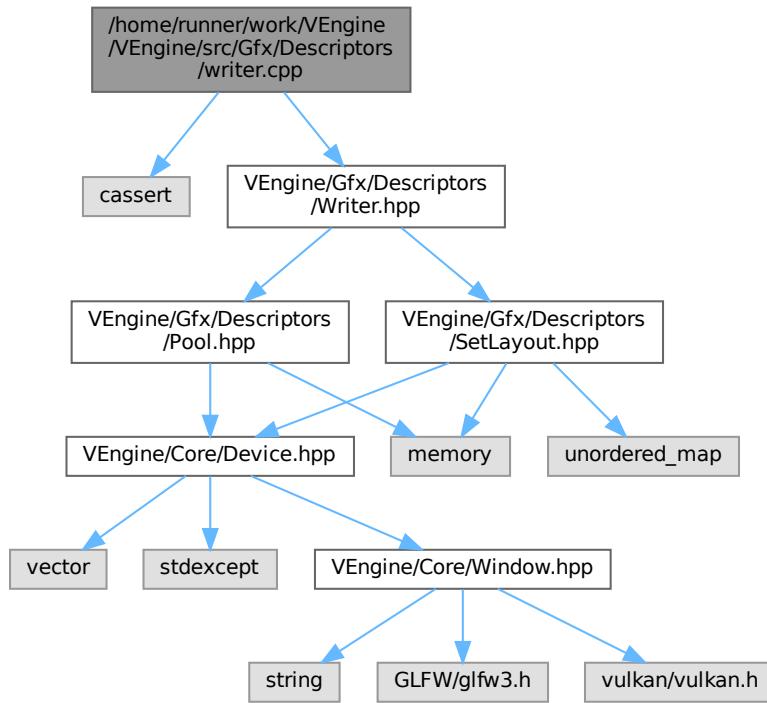
00007     assert(m_bindings.contains(binding) == 0 && "Binding already exists in layout");
00008     VkDescriptorSetLayoutBinding layoutBinding{};
00009     layoutBinding.binding = binding;
00010     layoutBinding.descriptorType = descriptorType;
00011     layoutBinding.descriptorCount = count;
00012     layoutBinding.stageFlags = stageFlags;
00013     m_bindings[binding] = layoutBinding;
00014     return *this;
00015 }
00016
00017 ven::DescriptorSetLayout::DescriptorsetLayout(Device &device, const std::unordered_map<uint32_t,
00018                                                 VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
00019 {
00020     std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
00021     setLayoutBindings.reserve(bindings.size());
00022     for (auto [fst, snd] : bindings) {
00023         setLayoutBindings.push_back(snd);
00024     }
00025     VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
00026     descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
00027     descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
00028     descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();
00029
00030     if (vkCreateDescriptorSetLayout(
00031         m_device.device(),
00032         &descriptorSetLayoutInfo,
00033         nullptr,
00034         &m_descriptorsetLayout) != VK_SUCCESS) {
00035         throw std::runtime_error("failed to create descriptor set layout!");
00036     }
00037 }

```

## 8.114 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/writer.cpp File Reference

```
#include <cassert>
#include "VEngine/Gfx/Descriptors/Writer.hpp"
```

Include dependency graph for writer.cpp:



## 8.115 writer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Gfx/Descriptors/Writer.hpp"
00004
00005 ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const
   VkDescriptorBufferInfo *bufferInfo)
00006 {
00007     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00008
00009     const auto &bindingDescription = m_setLayout.m_bindings.at(binding);
00010
00011     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
       expects multiple");
00012
00013     VkWriteDescriptorSet write{};
00014     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00015     write.descriptorType = bindingDescription.descriptorType;
00016     write.dstBinding = binding;
00017     write.pBufferInfo = bufferInfo;
00018     write.descriptorCount = 1;
00019
00020     m_writes.push_back(write);
00021     return *this;
00022 }
00023
00024 ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const
   VkDescriptorImageInfo *imageInfo)
00025 {
00026     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00027
00028     const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings.at(binding);
00029
  
```

```

00030     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
00031     expects multiple");
00032     VkWriteDescriptorSet write{};
00033     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00034     write.descriptorType = bindingDescription.descriptorType;
00035     write.dstBinding = binding;
00036     write.pImageInfo = imageInfo;
00037     write.descriptorCount = 1;
00038
00039     m_writes.push_back(write);
00040     return *this;
00041 }
00042
00043 bool ven::DescriptorWriter::build(VkDescriptorSet &set)
00044 {
00045     if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
00046         return false;
00047     }
00048     overwrite(set);
00049     return true;
00050 }
00051
00052 void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set) {
00053     for (auto &[sType, pNext, dstSet, dstBinding, dstArrayElement, descriptorCount, descriptorType,
00054     pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
00055         dstSet = set;
00056         vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
00057         m_writes.data(), 0, nullptr);
00057 }

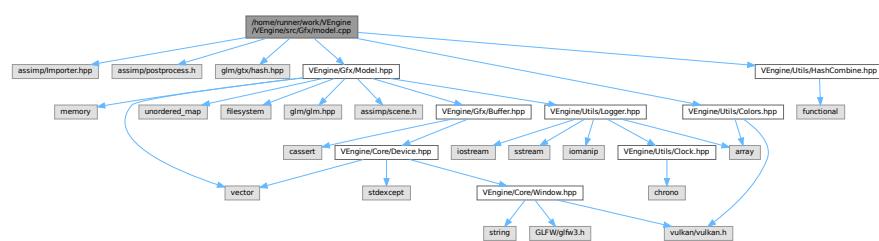
```

## 8.116 /home/runner/work/VEngine/VEngine/src/Gfx/model.cpp File Reference

```

#include <assimp/Importer.hpp>
#include <assimp/postprocess.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Utils/HashCombine.hpp"
Include dependency graph for model.cpp:

```



### Classes

- struct `std::hash< ven::Model::Vertex >`

### Macros

- `#define GLM_ENABLE_EXPERIMENTAL`

## 8.116.1 Macro Definition Documentation

### 8.116.1.1 GLM\_ENABLE\_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 4 of file [model.cpp](#).

## 8.117 model.cpp

[Go to the documentation of this file.](#)

```
00001 #include <assimp/Importer.hpp>
00002 #include <assimp/postprocess.h>
00003
00004 #define GLM_ENABLE_EXPERIMENTAL
00005 #include <glm/gtx/hash.hpp>
00006
00007 #include "VEngine/Gfx/Model.hpp"
00008 #include "VEngine/Utils/Colors.hpp"
00009 #include "VEngine/Utils/HashCombine.hpp"
00010
00011 template<>
00012 struct std::hash<ven::Model::Vertex> {
00013     size_t operator()(ven::Model::Vertex const &vertex) const noexcept {
00014         size_t seed = 0;
00015         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00016         return seed;
00017     }
00018 };
00019
00020 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
m_indexCount(0)
00021 {
00022     createVertexBuffer(builder.vertices);
00023     createIndexBuffer(builder.indices);
00024 }
00025
00026 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00027 {
00028     m_vertexCount = static_cast<uint32_t>(vertices.size());
00029     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00030     constexpr unsigned long vertexSize = sizeof(vertices[0]);
00031     const VkDeviceSize bufferSize = vertexSize * m_vertexCount;
00032
00033     Buffer stagingBuffer{m_device, vertexSize, m_vertexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00034
00035     stagingBuffer.map();
00036     stagingBuffer.writeToBuffer(vertices.data());
00037
00038     m_vertexBuffer = std::make_unique<Buffer>(m_device, vertexSize, m_vertexCount,
VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00039
00040     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00041 }
00042
00043 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00044 {
00045     m_indexCount = static_cast<uint32_t>(indices.size());
00046     m_hasIndexBuffer = m_indexCount > 0;
00047
00048     if (!m_hasIndexBuffer) {
00049         return;
00050     }
00051
00052     constexpr uint32_t indexSize = sizeof(indices[0]);
00053
00054     Buffer stagingBuffer{m_device, indexSize, m_indexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00055
00056     stagingBuffer.map();
00057     stagingBuffer.writeToBuffer(indices.data());
00058
00059     m_indexBuffer = std::make_unique<Buffer>(m_device, indexSize, m_indexCount,
VK_BUFFER_USAGE_INDEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
```

```

00060
00061     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), sizeof(indices[0]) *
00062     m_indexCount);
00063 }
00064 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00065 {
00066     if (m_hasIndexBuffer) {
00067         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00068     } else {
00069         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00070     }
00071 }
00072
00073 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00074 {
00075     const std::array buffers{m_vertexBuffer->getBuffer()};
00076     constexpr std::array<VkDeviceSize, 1> offsets{};
00077     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets.data());
00078
00079     if (m_hasIndexBuffer) {
00080         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00081     }
00082 }
00083
00084 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00085 {
00086     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00087     bindingDescriptions[0].binding = 0;
00088     bindingDescriptions[0].stride = sizeof(Vertex);
00089     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00090     return bindingDescriptions;
00091 }
00092
00093 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00094 {
00095     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00096
00097     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00098     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00099     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00100    attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00101
00102     return attributeDescriptions;
00103 }
00104
00105 void ven::Model::Builder::loadModel(const std::string &filename)
00106 {
00107     Assimp::Importer importer;
00108
00109     const aiScene* scene = importer.ReadFile(filename, aiProcess_Triangulate | aiProcess_FlipUVs |
00110     aiProcess_CalcTangentSpace | aiProcess_GenNormals);
00111
00112     if ((scene == nullptr) || ((scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE) != 0U) || (scene->mRootNode
00113     == nullptr)) {
00114         throw std::runtime_error("Failed to load model with Assimp: " +
00115         std::string(importer.GetErrorString()));
00116     }
00117
00118     vertices.clear();
00119     indices.clear();
00120
00121     processNode(scene->mRootNode, scene);
00122
00123 void ven::Model::Builder::processNode(const aiNode* node, const aiScene* scene) {
00124     for (unsigned int i = 0; i < node->mNumMeshes; i++) {
00125         const aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
00126         processMesh(mesh, scene);
00127     }
00128
00129     for (unsigned int i = 0; i < node->mNumChildren; i++) {
00130         processNode(node->mChildren[i], scene);
00131     }
00132
00133 void ven::Model::Builder::processMesh(const aiMesh* mesh, const aiScene* scene) {
00134     std::unordered_map<Vertex, uint32_t> uniqueVertices;
00135
00136     for (unsigned int i = 0; i < mesh->mNumVertices; i++) {
00137         Vertex vertex{};
00138
00139         vertex.position = glm::vec3(
00140             mesh->mVertices[i].x,
00141             mesh->mVertices[i].y,
00142             mesh->mVertices[i].z
00143         );

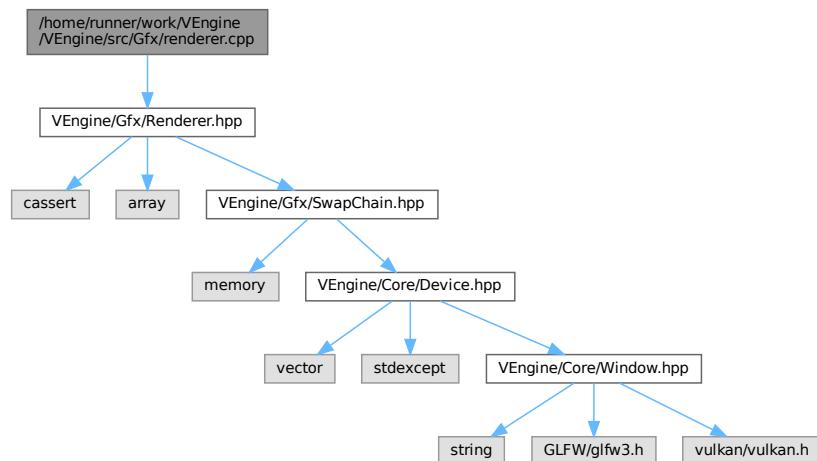
```

```

00143
00144     if (mesh->HasNormals()) {
00145         vertex.normal = glm::vec3(
00146             mesh->mNormals[i].x,
00147             mesh->mNormals[i].y,
00148             mesh->mNormals[i].z
00149         );
00150     }
00151
00152     if (mesh->mTextureCoords[0] != nullptr) {
00153         vertex.uv = glm::vec2(
00154             mesh->mTextureCoords[0][i].x,
00155             mesh->mTextureCoords[0][i].y
00156         );
00157     } else {
00158         vertex.uv = glm::vec2(0.0F, 0.0F);
00159     }
00160
00161     if (!uniqueVertices.contains(vertex)) {
00162         uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00163         vertices.push_back(vertex);
00164     }
00165
00166     indices.push_back(uniqueVertices[vertex]);
00167 }
00168 }
```

## 8.118 /home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp File Reference

#include "VEngine/Gfx/Renderer.hpp"  
Include dependency graph for renderer.cpp:



## 8.119 renderer.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Gfx/Renderer.hpp"
00002
00003 void ven::Renderer::createCommandBuffers()
00004 {
00005     m_commandBuffers.resize(MAX_FRAMES_IN_FLIGHT);
00006     VkCommandBufferAllocateInfo allocInfo{};

  
```

```

00007     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00008     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00009     allocInfo.commandPool = m_device.getCommandPool();
0010     allocInfo.commandBufferCount = static_cast<uint32_t>(m_commandBuffers.size());
0011
0012     if (vkAllocateCommandBuffers(m_device.device(), &allocInfo, m_commandBuffers.data()) != VK_SUCCESS) {
0013         throw std::runtime_error("Failed to allocate command buffers");
0014     }
0015 }
0016
0017 void ven::Renderer::freeCommandBuffers()
0018 {
0019     vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(),
0020     static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
0021     m_commandBuffers.clear();
0022
0023 void ven::Renderer::recreateSwapChain()
0024 {
0025     VkExtent2D extent = m_window.getExtent();
0026     while (extent.width == 0 || extent.height == 0) {
0027         extent = m_window.getExtent();
0028         glfwWaitEvents();
0029     }
0030     vkDeviceWaitIdle(m_device.device());
0031     if (m_swapChain == nullptr) {
0032         m_swapChain = std::make_unique<SwapChain>(m_device, extent);
0033     } else {
0034         std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
0035         m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
0036         if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
0037             throw std::runtime_error("Swap chain image/depth format changed");
0038         }
0039     }
0040     // well be back
0041 }
0042
0043 VkCommandBuffer ven::Renderer::beginFrame()
0044 {
0045     assert(!m_isFrameStarted && "Can't start new frame while previous one is still in progress");
0046
0047     const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
0048     if (result == VK_ERROR_OUT_OF_DATE_KHR) {
0049         recreateSwapChain();
0050         return nullptr;
0051     }
0052
0053     if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
0054         throw std::runtime_error("Failed to acquire swap chain image");
0055     }
0056
0057     m_isFrameStarted = true;
0058
0059     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
0060     VkCommandBufferBeginInfo beginInfo{};
0061     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
0062
0063     if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
0064         throw std::runtime_error("Failed to begin recording command buffer");
0065     }
0066     return commandBuffer;
0067 }
0068
0069 void ven::Renderer::endFrame()
0070 {
0071     assert(m_isFrameStarted && "Can't end frame that hasn't been started");
0072
0073     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
0074     if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
0075         throw std::runtime_error("Failed to record command buffer");
0076     }
0077     if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer,
0078     &m_currentImageIndex); result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
0079     m_window.wasWindowResized()) {
0080         m_window.resetWindowResizedFlag();
0081         recreateSwapChain();
0082     }
0083     else if (result != VK_SUCCESS) {
0084         throw std::runtime_error("Failed to submit command buffer");
0085     }
0086     m_isFrameStarted = false;
0087     m_currentFrameIndex = (m_currentFrameIndex + 1) % MAX_FRAMES_IN_FLIGHT;
0088 }
0089
0090 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const

```

```

00090 {
00091     assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");
00092     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
from a different frame");
00093
00094     VkRenderPassBeginInfo renderPassInfo{};
00095     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00096     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00097     renderPassInfo.framebuffer = m_swapChain->getFrameBuffer(m_currentImageIndex);
00098
00099     renderPassInfo.renderArea.offset = { .x=0, .y=0};
00100    renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent ();
00101
00102    renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
00103    renderPassInfo.pClearValues = m_clearValues.data();
00104
00105    vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00106
00107    VkViewport viewport{};
00108    viewport.x = 0.0F;
00109    viewport.y = 0.0F;
00110    viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00111    viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00112    viewport.minDepth = 0.0F;
00113    viewport.maxDepth = 1.0F;
00114    const VkRect2D scissor{0, 0}, m_swapChain->getSwapChainExtent ();
00115    vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00116    vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00117 }
00118
00119 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00120 {
00121     assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
00122     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
from a different frame");
00123
00124     vkCmdEndRenderPass(commandBuffer);
00125 }

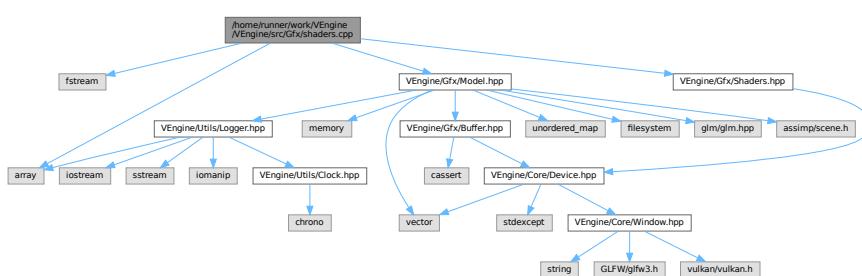
```

## 8.120 /home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp File Reference

```

#include <fstream>
#include <array>
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Gfx/Shaders.hpp"
Include dependency graph for shaders.cpp:

```



## 8.121 shaders.cpp

[Go to the documentation of this file.](#)

```

00001 #include <fstream>
00002 #include <array>

```

```

00003
00004 #include "VEngine/Gfx/Model.hpp"
00005 #include "VEngine/Gfx/Shaders.hpp"
00006
00007 ven::Shaders::~Shaders()
00008 {
00009     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00010    vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00011    vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
00012 }
00013
00014 std::vector<char> ven::Shaders::readFile(const std::string &filename) {
00015     std::ifstream file(filename, std::ios::binary | std::ios::ate);
00016     if (!file.is_open()) {
00017         throw std::runtime_error("failed to open file!");
00018     }
00019
00020     const long int fileSize = file.tellg();
00021     std::vector<char> buffer(static_cast<long unsigned int>(fileSize));
00022     file.seekg(0);
00023     file.read(buffer.data(), fileSize);
00024     return buffer;
00025 }
00026
00027 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo)
00028 {
00029     const std::vector<char> vertCode = readFile(vertFilepath);
00030     const std::vector<char> fragCode = readFile(fragFilepath);
00031
00032     createShaderModule(vertCode, &m_vertShaderModule);
00033     createShaderModule(fragCode, &m_fragShaderModule);
00034
00035     std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
00036     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00037     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00038     shaderStages[0].module = m_vertShaderModule;
00039     shaderStages[0].pName = "main";
00040     shaderStages[0].flags = 0;
00041     shaderStages[0].pNext = nullptr;
00042     shaderStages[0].pSpecializationInfo = nullptr;
00043
00044     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00045     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00046     shaderStages[1].module = m_fragShaderModule;
00047     shaderStages[1].pName = "main";
00048     shaderStages[1].flags = 0;
00049     shaderStages[1].pNext = nullptr;
00050     shaderStages[1].pSpecializationInfo = nullptr;
00051
00052     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00053     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00054     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00055     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00056     vertexInputInfo.vertexAttributeDescriptionCount =
00057         static_cast<uint32_t>(attributeDescriptions.size());
00058     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00059     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00060     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00061
00062     VkPipelineViewportStateCreateInfo viewportInfo{};
00063     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00064     viewportInfo.viewportCount = 1;
00065     viewportInfo.pViewports = nullptr;
00066     viewportInfo.scissorCount = 1;
00067     viewportInfo.pScissors = nullptr;
00068
00069
00070     VkGraphicsPipelineCreateInfo pipelineInfo{};
00071     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00072     pipelineInfo.stageCount = 2;
00073     pipelineInfo.pStages = shaderStages.data();
00074     pipelineInfo.pVertexInputState = &vertexInputInfo;
00075     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00076     pipelineInfo.pViewportState = &viewportInfo;
00077     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00078     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00079
00080     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00081     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00082     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00083
00084     pipelineInfo.layout = configInfo.pipelineLayout;
00085     pipelineInfo.renderPass = configInfo.renderPass;
00086     pipelineInfo.subpass = configInfo.subpass;
00087

```

```

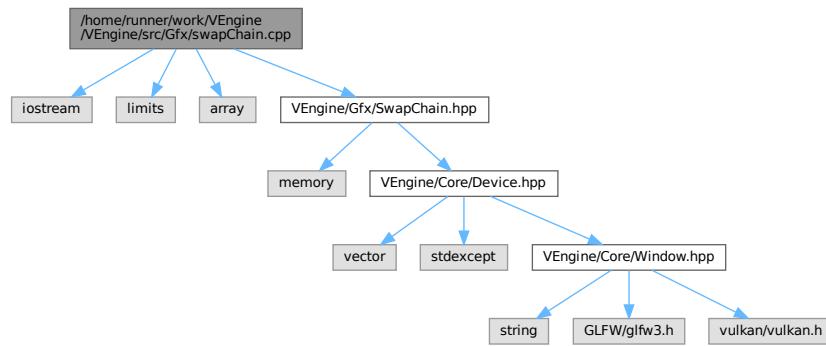
00088     pipelineInfo.basePipelineIndex = -1;
00089     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00090
00091     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
00092         &m_graphicsPipeline) != VK_SUCCESS) {
00093         throw std::runtime_error("failed to create graphics pipeline");
00094     }
00095
00096 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
00097 {
00098     VkShaderModuleCreateInfo createInfo{};
00099     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00100     createInfo.codeSize = code.size();
00101     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00102
00103     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00104         throw std::runtime_error("failed to create shader module");
00105     }
00106 }
00107
00108 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00109 {
00110     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00111     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00112     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00113
00114     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00115     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00116     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00117     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00118     configInfo.rasterizationInfo.lineWidth = 1.0F;
00119     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE; // to enable later
00120     configInfo.rasterizationInfo.backFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00121     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00122     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00123     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00124     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00125
00126     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00127     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00128     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00129     configInfo.multisampleInfo.minSampleShading = 1.0F;
00130     configInfo.multisampleInfo.pSampleMask = nullptr;
00131     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00132     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00133
00134     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
00135         VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00136     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00137     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00138     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00139     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00140     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00141     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00142     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00143
00144     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00145     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00146     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00147     configInfo.colorBlendInfo.attachmentCount = 1;
00148     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00149     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00150     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00151     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00152     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00153
00154     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00155     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00156     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00157     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00158     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00159     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00160     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00161     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00162     configInfo.depthStencilInfo.front = {};
00163     configInfo.depthStencilInfo.back = {};
00164
00165     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00166     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00167     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00168     configInfo.dynamicStateInfo.dynamicStateCount =
00169         static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00170     configInfo.dynamicStateInfo.flags = 0;
00171     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();

```

```
00170     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00171 }
```

## 8.122 /home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp File Reference

```
#include <iostream>
#include <limits>
#include <array>
#include "VEngine/Gfx/SwapChain.hpp"
Include dependency graph for swapChain.cpp:
```



## 8.123 swapChain.cpp

[Go to the documentation of this file.](#)

```
00001 #include <iostream>
00002 #include <limits>
00003 #include <array>
00004
00005 #include "VEngine/Gfx/SwapChain.hpp"
00006
00007 ven::SwapChain::~SwapChain()
00008 {
00009     for (VkImageView_T *imageView : m_swapChainImageViews) {
00010         vkDestroyImageView(m_device.device(), imageView, nullptr);
00011     }
00012     m_swapChainImageViews.clear();
00013
00014     if (m_swapChain != nullptr) {
00015         vkDestroySwapchainKHR(m_device.device(), m_swapChain, nullptr);
00016         m_swapChain = nullptr;
00017     }
00018
00019     for (size_t i = 0; i < m_depthImages.size(); i++) {
00020         vkDestroyImage(m_device.device(), m_depthImageViews[i], nullptr);
00021         vkDestroyImage(m_device.device(), m_depthImages[i], nullptr);
00022         vkFreeMemory(m_device.device(), m_depthImageMemory[i], nullptr);
00023     }
00024
00025     for (VkFramebuffer_T *framebuffer : m_swapChainFrameBuffers) {
00026         vkDestroyFramebuffer(m_device.device(), framebuffer, nullptr);
00027     }
00028
00029     vkDestroyRenderPass(m_device.device(), m_renderPass, nullptr);
00030
00031     // cleanup synchronization objects
00032     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00033         vkDestroySemaphore(m_device.device(), m_renderFinishedSemaphores[i], nullptr);
  
```

```

00034     vkDestroySemaphore(m_device.device(), m_imageAvailableSemaphores[i], nullptr);
00035     vkDestroyFence(m_device.device(), m_inFlightFences[i], nullptr);
00036 }
00037 }
00038
00039 void ven::SwapChain::init()
00040 {
00041     createSwapChain();
00042     createImageViews();
00043     createRenderPass();
00044     createDepthResources();
00045     createFrameBuffers();
00046     createSyncObjects();
00047 }
00048
00049 VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
00050 {
00051     vkWaitForFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame], VK_TRUE,
00052     std::numeric_limits<uint64_t>::max());
00053
00054     return vkAcquireNextImageKHR(m_device.device(), m_swapChain, std::numeric_limits<uint64_t>::max(),
00055     m_imageAvailableSemaphores[m_currentFrame], VK_NULL_HANDLE, imageIndex);
00056 }
00057
00058 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
00059 *imageIndex)
00060 {
00061     if (m_imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00062         vkWaitForFences(m_device.device(), 1, &m_imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00063     }
00064     m_imagesInFlight[*imageIndex] = m_inFlightFences[m_currentFrame];
00065
00066     VkSubmitInfo submitInfo = {};
00067     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00068
00069     const std::array<VkSemaphore, 1> waitSemaphores = {m_imageAvailableSemaphores[m_currentFrame]};
00070     constexpr std::array<VkPipelineStageFlags, 1> waitStages =
00071     {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00072     submitInfo.waitSemaphoreCount = 1;
00073     submitInfo.pWaitSemaphores = waitSemaphores.data();
00074     submitInfo.pWaitDstStageMask = waitStages.data();
00075
00076     submitInfo.commandBufferCount = 1;
00077     submitInfo.pCommandBuffers = buffers;
00078
00079     const std::array<VkSemaphore, 1> signalSemaphores = {m_renderFinishedSemaphores[m_currentFrame]};
00080     submitInfo.signalSemaphoreCount = 1;
00081     submitInfo.pSignalSemaphores = signalSemaphores.data();
00082
00083     vkResetFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame]);
00084     if (vkQueueSubmit(m_device.graphicsQueue(), 1, &submitInfo, m_inFlightFences[m_currentFrame]) !=
00085         VK_SUCCESS) {
00086         throw std::runtime_error("failed to submit draw command buffer!");
00087     }
00088
00089     VkPresentInfoKHR presentInfo = {};
00090     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00091
00092     presentInfo.waitSemaphoreCount = 1;
00093     presentInfo.pWaitSemaphores = signalSemaphores.data();
00094
00095     presentInfo.pImageIndices = imageIndex;
00096
00097     const VkResult result = vkQueuePresentKHR(m_device.presentQueue(), &presentInfo);
00098
00099     m_currentFrame = (m_currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00100
00101     return result;
00102 }
00103
00104 void ven::SwapChain::createSwapChain()
00105 {
00106     const auto [capabilities, formats, presentModes] = m_device.getSwapChainSupport();
00107
00108     const auto [format, colorSpace] = chooseSwapSurfaceFormat(formats);
00109     const VkPresentModeKHR presentMode = chooseSwapPresentMode(presentModes);
00110     const VkExtent2D extent = chooseSwapExtent(capabilities);
00111
00112     uint32_t imageCount = capabilities.minImageCount + 1;
00113     if (capabilities.maxImageCount > 0 && imageCount > capabilities.maxImageCount) {
00114         imageCount = capabilities.maxImageCount;
00115     }

```

```

00116     VkSwapchainCreateInfoKHR createInfo = {};
00117     createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
00118     createInfo.surface = m_device.surface();
00119
00120     createInfo.minImageCount = imageCount;
00121     createInfo.imageFormat = format;
00122     createInfo.imageColorSpace = colorSpace;
00123     createInfo.imageExtent = extent;
00124     createInfo.imageArrayLayers = 1;
00125     createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
00126
00127     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
00128         m_device.findPhysicalQueueFamilies();
00129     const std::array<uint32_t, 2> queueFamilyIndices = {graphicsFamily, presentFamily};
00130
00131     if (graphicsFamily != presentFamily) {
00132         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00133         createInfo.queueFamilyIndexCount = 2;
00134         createInfo.pQueueFamilyIndices = queueFamilyIndices.data();
00135     } else {
00136         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00137         createInfo.queueFamilyIndexCount = 0; // Optional
00138         createInfo.pQueueFamilyIndices = nullptr; // Optional
00139
00140     createInfo.preTransform = capabilities.currentTransform;
00141     createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
00142
00143     createInfo.presentMode = presentMode;
00144     createInfo.clipped = VK_TRUE;
00145
00146     createInfo.oldSwapchain = m_oldSwapChain == nullptr ? VK_NULL_HANDLE :
00147         m_oldSwapChain->m_swapChain;
00148
00149     if (vkCreateSwapchainKHR(m_device.device(), &createInfo, nullptr, &m_swapChain) != VK_SUCCESS) {
00150         throw std::runtime_error("failed to create swap chain!");
00151     }
00152
00153     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, nullptr);
00154     m_swapChainImages.resize(imageCount);
00155     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, m_swapChainImages.data());
00156
00157     m_swapChainImageFormat = format;
00158     m_swapChainExtent = extent;
00159
00160 void ven::SwapChain::createImageViews()
00161 {
00162     m_swapChainImageViews.resize(m_swapChainImages.size());
00163     for (size_t i = 0; i < m_swapChainImages.size(); i++) {
00164         VkImageViewCreateInfo viewInfo{};
00165         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00166         viewInfo.image = m_swapChainImages[i];
00167         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00168         viewInfo.format = m_swapChainImageFormat;
00169         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00170         viewInfo.subresourceRange.baseMipLevel = 0;
00171         viewInfo.subresourceRange.levelCount = 1;
00172         viewInfo.subresourceRange.baseArrayLayer = 0;
00173         viewInfo.subresourceRange.layerCount = 1;
00174
00175         if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_swapChainImageViews[i]) !=
00176             VK_SUCCESS) {
00177             throw std::runtime_error("failed to create texture image view!");
00178         }
00179     }
00180
00181 void ven::SwapChain::createRenderPass()
00182 {
00183     VkAttachmentDescription depthAttachment{};
00184     depthAttachment.format = findDepthFormat();
00185     depthAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00186     depthAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00187     depthAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00188     depthAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00189     depthAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00190     depthAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00191     depthAttachment.finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00192
00193     VkAttachmentReference depthAttachmentRef{};
00194     depthAttachmentRef.attachment = 1;
00195     depthAttachmentRef.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00196
00197     VkAttachmentDescription colorAttachment = {};
00198     colorAttachment.format = getSwapChainImageFormat();
00199     colorAttachment.samples = VK_SAMPLE_COUNT_1_BIT;

```

```
00200     colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00201     colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
00202     colorAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00203     colorAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00204     colorAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00205     colorAttachment.finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
00206
00207     VkAttachmentReference colorAttachmentRef = {};
00208     colorAttachmentRef.attachment = 0;
00209     colorAttachmentRef.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00210
00211     VkSubpassDescription subpass = {};
00212     subpass.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
00213     subpass.colorAttachmentCount = 1;
00214     subpass.pColorAttachments = &colorAttachmentRef;
00215     subpass.pDepthStencilAttachment = &depthAttachmentRef;
00216
00217     VkSubpassDependency dependency = {};
00218     dependency.srcSubpass = VK_SUBPASS_EXTERNAL;
00219     dependency.srcAccessMask = 0;
00220     dependency.srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00221     dependency.dstSubpass = 0;
00222     dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00223     dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00224
00225     const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
00226     VkRenderPassCreateInfo renderPassInfo = {};
00227     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
00228     renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00229     renderPassInfo.pAttachments = attachments.data();
00230     renderPassInfo.subpassCount = 1;
00231     renderPassInfo.pSubpasses = &subpass;
00232     renderPassInfo.dependencyCount = 1;
00233     renderPassInfo.pDependencies = &dependency;
00234
00235     if (vkCreateRenderPass(m_device.device(), &renderPassInfo, nullptr, &m_renderPass) != VK_SUCCESS)
{
00236         throw std::runtime_error("failed to create render pass!");
00237     }
00238 }
00239
00240 void ven::SwapChain::createFrameBuffers()
00241 {
00242     m_swapChainFrameBuffers.resize(imageCount());
00243     for (size_t i = 0; i < imageCount(); i++) {
00244         std::array<VkImageView, 2> attachments = {m_swapChainImageViews[i], m_depthImageViews[i]};
00245
00246         const auto [width, height] = getSwapChainExtent();
00247         VkFramebufferCreateInfo framebufferInfo = {};
00248         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00249         framebufferInfo.renderPass = m_renderPass;
00250         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00251         framebufferInfo.pAttachments = attachments.data();
00252         framebufferInfo.width = width;
00253         framebufferInfo.height = height;
00254         framebufferInfo.layers = 1;
00255
00256         if (vkCreateFramebuffer(m_device.device(), &framebufferInfo, nullptr,
&m_swapChainFrameBuffers[i]) != VK_SUCCESS) {
00257             throw std::runtime_error("failed to create framebuffer!");
00258         }
00259     }
00260 }
00261
00262 void ven::SwapChain::createDepthResources()
00263 {
00264     const VkFormat depthFormat = findDepthFormat();
00265     const auto [width, height] = getSwapChainExtent();
00266
00267     m_swapChainDepthFormat = depthFormat;
00268     m_depthImages.resize(imageCount());
00269     m_depthImageMemory.resize(imageCount());
00270     m_depthImageViews.resize(imageCount());
00271
00272     for (size_t i = 0; i < m_depthImages.size(); i++) {
00273         VkImageCreateInfo imageInfo{};
00274         imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00275         imageInfo.imageType = VK_IMAGE_TYPE_2D;
00276         imageInfo.extent.width = width;
00277         imageInfo.extent.height = height;
00278         imageInfo.extent.depth = 1;
00279         imageInfo.mipLevels = 1;
00280         imageInfo.arrayLayers = 1;
00281         imageInfo.format = depthFormat;
```

```

00282     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00283     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00284     imageInfo.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;
00285     imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00286     imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00287     imageInfo.flags = 0;
00288
00289     m_device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_depthImages[i],
00290     m_depthImageMemory[i]);
00291
00292     VkImageViewCreateInfo viewInfo{};
00293     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00294     viewInfo.image = m_depthImages[i];
00295     viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00296     viewInfo.format = depthFormat;
00297     viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00298     viewInfo.subresourceRange.baseMipLevel = 0;
00299     viewInfo.subresourceRange.levelCount = 1;
00300     viewInfo.subresourceRange.baseArrayLayer = 0;
00301     viewInfo.subresourceRange.layerCount = 1;
00302
00303     if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_depthImageViews[i]) != VK_SUCCESS) {
00304         throw std::runtime_error("failed to create texture image view!");
00305     }
00306 }
00307
00308 void ven::SwapChain::createSyncObjects()
00309 {
00310     m_imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00311     m_renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00312     m_inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
00313     m_imagesInFlight.resize(imageCount(), VK_NULL_HANDLE);
00314
00315     VkSemaphoreCreateInfo semaphoreInfo = {};
00316     semaphoreInfo.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
00317
00318     VkFenceCreateInfo fenceInfo = {};
00319     fenceInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
00320     fenceInfo.flags = VK_FENCE_CREATE_SIGNALED_BIT;
00321
00322     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00323         if (vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
00324             &m_imageAvailableSemaphores[i]) != VK_SUCCESS ||
00325             vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
00326             &m_renderFinishedSemaphores[i]) != VK_SUCCESS ||
00327             vkCreateFence(m_device.device(), &fenceInfo, nullptr, &m_inFlightFences[i]) != VK_SUCCESS)
00328         {
00329             throw std::runtime_error("failed to create synchronization objects for a frame!");
00330         }
00331     }
00332 }
00333 VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
00334     &availableFormats)
00335 {
00336     for (const auto &availableFormat : availableFormats) {
00337         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
00338             VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00339             return availableFormat;
00340         }
00341     }
00342     return availableFormats[0];
00343 }
00344 VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
00345     &availablePresentModes)
00346 {
00347     for (const auto &availablePresentMode : availablePresentModes) {
00348         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00349             std::cout << "Present mode: Mailbox\n";
00350             return availablePresentMode;
00351         }
00352     }
00353     for (const auto &availablePresentMode : availablePresentModes) {
00354         if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00355             std::cout << "Present mode: Immediate" << '\n';
00356             return availablePresentMode;
00357         }
00358     }
00359     std::cout << "Present mode: V-Sync\n";
00360     return VK_PRESENT_MODE_FIFO_KHR;
00361 }
```

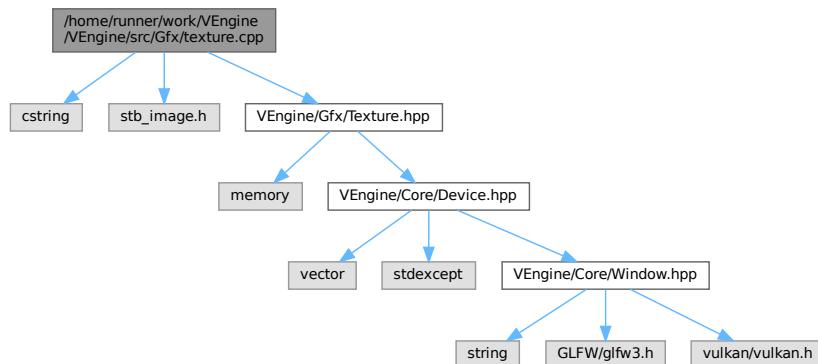
```

00361
00362     VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00363     {
00364         if (capabilities.currentExtent.width != std::numeric_limits<uint32_t>::max()) {
00365             return capabilities.currentExtent;
00366         }
00367         VkExtent2D actualExtent = m_windowExtent;
00368         actualExtent.width = std::max(capabilities.minImageExtent.width,
00369             std::min(capabilities.maxImageExtent.width, actualExtent.width));
00369         actualExtent.height = std::max(capabilities.minImageExtent.height,
00370             std::min(capabilities.maxImageExtent.height, actualExtent.height));
00370
00371         return actualExtent;
00372     }
00373
00374     VkFormat ven::SwapChain::findDepthFormat() const
00375     {
00376         return m_device.findSupportedFormat(
00377             {VK_FORMAT_D32_SFLOAT, VK_FORMAT_D32_SFLOAT_S8_UINT, VK_FORMAT_D24_UNORM_S8_UINT},
00378             VK_IMAGE_TILING_OPTIMAL,
00379             VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT);
00380     }

```

## 8.124 /home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp File Reference

```
#include <cstring>
#include <stb_image.h>
#include "VEngine/Gfx/Texture.hpp"
Include dependency graph for texture.cpp:
```



## 8.125 texture.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cstring>
00002
00003 #include <stb_image.h>
00004
00005 #include "VEngine/Gfx/Texture.hpp"
00006
00007 ven::Texture::Texture(Device &device, const std::string &textureFilepath) : m_device(device)
00008 {
00009     createTextureImage(textureFilepath);
0010     createTextureImageView(VK_IMAGE_VIEW_TYPE_2D);
0011     createTextureSampler();
0012     updateDescriptor();
0013 }

```

```

00014
00015 ven::Texture::Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00016     VkSampleCountFlagBits sampleCount)
00017 {
00018     m_device{device}, m_format(format), m_extent(extent)
00019     VkImageAspectFlags aspectMask = 0;
00020     VkImageLayout imageLayout;
00021
00022     if ((usage & VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT) != 0u) {
00023         aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00024         imageLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00025     }
00026     if ((usage & VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT) != 0u) {
00027         aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00028         imageLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00029     }
00030
00031     // Don't like this, should I be using an image array instead of multiple images?
00032     VkImageCreateInfo imageInfo{};
00033     imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00034     imageInfo.imageType = VK_IMAGE_TYPE_2D;
00035     imageInfo.format = format;
00036     imageInfo.extent = extent;
00037     imageInfo.mipLevels = 1;
00038     imageInfo.arrayLayers = 1;
00039     imageInfo.samples = sampleCount;
00040     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00041     imageInfo.usage = usage;
00042     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00043     device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_textureImage,
00044     m_textureImageMemory);
00045
00046     VkImageViewCreateInfo viewInfo{};
00047     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00048     viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00049     viewInfo.format = format;
00050     viewInfo.subresourceRange = {};
00051     viewInfo.subresourceRange.aspectMask = aspectMask;
00052     viewInfo.subresourceRange.baseMipLevel = 0;
00053     viewInfo.subresourceRange.levelCount = 1;
00054     viewInfo.subresourceRange.baseArrayLayer = 0;
00055     viewInfo.subresourceRange.layerCount = 1;
00056     viewInfo.image = m_textureImage;
00057     if (vkCreateImageView(device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00058         throw std::runtime_error("failed to create texture image view!");
00059     }
00060
00061     // Sampler should be separated out
00062     if ((usage & VK_IMAGE_USAGE_SAMPLED_BIT) != 0u) {
00063         // Create sampler to sample from the attachment in the fragment shader
00064         VkSamplerCreateInfo samplerInfo{};
00065         samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00066         samplerInfo.magFilter = VK_FILTER_LINEAR;
00067         samplerInfo.minFilter = VK_FILTER_LINEAR;
00068         samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00069         samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER;
00070         samplerInfo.addressModeV = samplerInfo.addressModeU;
00071         samplerInfo.addressModeW = samplerInfo.addressModeU;
00072         samplerInfo.mipLodBias = 0.0F;
00073         samplerInfo.maxAnisotropy = 1.0F;
00074         samplerInfo.minLod = 0.0F;
00075         samplerInfo.maxLod = 1.0F;
00076         samplerInfo.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
00077
00078         if (vkCreateSampler(device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS)
00079         {
00080             throw std::runtime_error("failed to create sampler!");
00081         }
00082
00083         VkImageLayout samplerImageLayout = imageLayout == VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
00084             ? VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
00085             : VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL;
00086         m_descriptor.sampler = m_textureSampler;
00087         m_descriptor.imageView = m_textureImageView;
00088         m_descriptor.imageLayout = samplerImageLayout;
00089     }
00090
00091     vkDestroySampler(m_device.device(), m_textureSampler, nullptr);
00092     vkDestroyImageView(m_device.device(), m_textureImageView, nullptr);
00093     vkDestroyImage(m_device.device(), m_textureImage, nullptr);
00094     vkFreeMemory(m_device.device(), m_textureImageMemory, nullptr);
00095 }
00096
00097 void ven::Texture::updateDescriptor()

```

```
00098 {
00099     m_descriptor.sampler = m_textureSampler;
00100     m_descriptor.imageView = m_textureImageView;
00101     m_descriptor.imageLayout = m_textureLayout;
00102 }
00103
00104 void ven::Texture::createTextureImage(const std::string &filepath)
00105 {
00106     int texWidth = 0;
00107     int texHeight = 0;
00108     int texChannels = 0;
00109     void *data = nullptr;
00110     stbi_uc *pixels = nullptr;
00111
00112     stbi_set_flip_vertically_on_load(1);
00113     pixels = stbi_load(filepath.c_str(), &texWidth, &texHeight, &texChannels, STBI_rgb_alpha);
00114     const auto imageSize = static_cast<VkDeviceSize>(texWidth * texHeight * 4);
00115
00116     if (pixels == nullptr) {
00117         throw std::runtime_error("failed to load texture image!");
00118     }
00119
00120 // mMipLevels = static_cast<uint32_t>(std::floor(std::log2(std::max(texWidth, texHeight)))) + 1;
00121     mMipLevels = 1;
00122
00123     VkBuffer stagingBuffer = nullptr;
00124     VkDeviceMemory stagingBufferMemory = nullptr;
00125
00126     m_device.createBuffer(
00127         imageSize,
00128         VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
00129         VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT,
00130         stagingBuffer,
00131         stagingBufferMemory);
00132
00133     vkMapMemory(m_device.device(), stagingBufferMemory, 0, imageSize, 0, &data);
00134     memcpy(data, pixels, imageSize);
00135     vkUnmapMemory(m_device.device(), stagingBufferMemory);
00136
00137     stbi_image_free(pixels);
00138
00139     m_format = VK_FORMAT_R8G8B8A8_SRGB;
00140     m_extent = {.width=static_cast<uint32_t>(texWidth), .height=static_cast<uint32_t>(texHeight),
00141     .depth=1};
00142
00143     VkImageCreateInfo imageInfo{};
00144     imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00145     imageInfo.imageType = VK_IMAGE_TYPE_2D;
00146     imageInfo.extent = m_extent;
00147     imageInfo.mipLevels = mMipLevels;
00148     imageInfo.arrayLayers = m_layerCount;
00149     imageInfo.format = m_format;
00150     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00151     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00152     imageInfo.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT |
VK_IMAGE_USAGE_SAMPLED_BIT;
00153     imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00154     imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00155
00156     m_device.createImageWithInfo(
00157         imageInfo,
00158         VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
00159         m_textureImage,
00160         m_textureImageMemory);
00161     m_device.transitionImageLayout(
00162         m_textureImage,
00163         VK_FORMAT_R8G8B8A8_SRGB,
00164         VK_IMAGE_LAYOUT_UNDEFINED,
00165         VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00166         mMipLevels,
00167         m_layerCount);
00168     m_device.copyBufferToImage(
00169         stagingBuffer,
00170         m_textureImage,
00171         static_cast<uint32_t>(texWidth),
00172         static_cast<uint32_t>(texHeight),
00173         m_layerCount);
00174
// comment this out if using mips
00175     m_device.transitionImageLayout(
00176         m_textureImage,
00177         VK_FORMAT_R8G8B8A8_SRGB,
00178         VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00179         VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL,
00180         mMipLevels,
00181         m_layerCount);
00182
```

```

00183 // If we generate mip maps then the final image will already be READ_ONLY_OPTIMAL
00184 // mDevice.generateMipmaps(mTextureImage, mFormat, texWidth, texHeight, mMipLevels);
00185 m_textureLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
00186
00187 vkDestroyBuffer(m_device.device(), stagingBuffer, nullptr);
00188 vkFreeMemory(m_device.device(), stagingBufferMemory, nullptr);
00189 }
00190
00191 void ven::Texture::createTextureImageView(const VkImageViewType viewType)
00192 {
00193     VkImageViewCreateInfo viewInfo{};
00194     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00195     viewInfo.image = m_textureImage;
00196     viewInfo.viewType = viewType;
00197     viewInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
00198     viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00199     viewInfo.subresourceRange.baseMipLevel = 0;
00200     viewInfo.subresourceRange.levelCount = m_mipLevels;
00201     viewInfo.subresourceRange.baseArrayLayer = 0;
00202     viewInfo.subresourceRange.layerCount = m_layerCount;
00203
00204     if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00205         throw std::runtime_error("failed to create texture image view!");
00206     }
00207 }
00208
00209 void ven::Texture::createTextureSampler()
00210 {
00211     VkSamplerCreateInfo samplerInfo{};
00212     samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00213     samplerInfo.magFilter = VK_FILTER_LINEAR;
00214     samplerInfo.minFilter = VK_FILTER_LINEAR;
00215
00216     samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00217     samplerInfo.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00218     samplerInfo.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00219
00220     samplerInfo.anisotropyEnable = VK_TRUE;
00221     samplerInfo.maxAnisotropy = 16.0F;
00222     samplerInfo.borderColor = VK_BORDER_COLOR_INT_OPAQUE_BLACK;
00223     samplerInfo.unnormalizedCoordinates = VK_FALSE;
00224
00225     // these fields useful for percentage close filtering for shadow maps
00226     samplerInfo.compareEnable = VK_FALSE;
00227     samplerInfo.compareOp = VK_COMPARE_OP_ALWAYS;
00228
00229     samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00230     samplerInfo.mipLodBias = 0.0F;
00231     samplerInfo.minLod = 0.0F;
00232     samplerInfo.maxLod = static_cast<float>(m_mipLevels);
00233
00234     if (vkCreateSampler(m_device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS) {
00235         throw std::runtime_error("failed to create texture sampler!");
00236     }
00237 }
00238
00239 void ven::Texture::transitionLayout(const VkCommandBuffer commandBuffer, const VkImageLayout
00240 oldLayout, const VkImageLayout newLayout) const
00241 {
00242     VkPipelineStageFlags sourceStage = 0;
00243     VkPipelineStageFlags destinationStage = 0;
00244     VkImageMemoryBarrier barrier{};
00245
00246     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00247     barrier.oldLayout = oldLayout;
00248     barrier.newLayout = newLayout;
00249
00250     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00251     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00252
00253     barrier.image = m_textureImage;
00254     barrier.subresourceRange.baseMipLevel = 0;
00255     barrier.subresourceRange.levelCount = m_mipLevels;
00256     barrier.subresourceRange.baseArrayLayer = 0;
00257     barrier.subresourceRange.layerCount = m_layerCount;
00258
00259     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00260         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00261         if (m_format == VK_FORMAT_D32_SFLOAT_S8_UINT || m_format == VK_FORMAT_D24_UNORM_S8_UINT) {
00262             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00263         }
00264     } else {
00265         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00266     }
00267     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00268         barrier.srcAccessMask = 0;
00269         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;

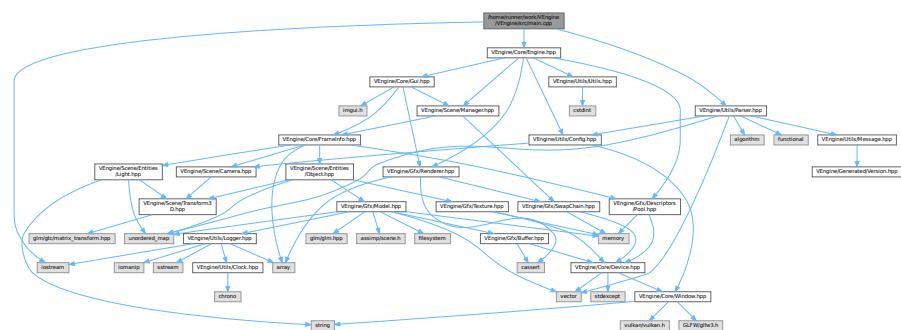
```

```

00269     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00270     destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00271 } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
00272     VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00273     barrier.srcAccessMask = 0;
00274     barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00275     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00276     destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00277 } else if (oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL && newLayout ==
00278     VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00279     barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00280     barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00281
00282     sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00283     destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00284 } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
00285     VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00286     barrier.srcAccessMask = 0;
00287     barrier.dstAccessMask = VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT |
00288         VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00289     sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00290     destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00291 } else if (oldLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL && newLayout ==
00292     VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL) {
00293     // This says that any cmd that acts in color output or after (dstStage)
00294     // that needs read or write access to a resource
00295     // must wait until all previous read accesses in fragment shader
00296     barrier.srcAccessMask = VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_SHADER_WRITE_BIT;
00297     barrier.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
00298         VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
00299     sourceStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00300     destinationStage = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
00301 } else {
00302     throw std::invalid_argument("unsupported layout transition!");
00303 }
00304 vkCmdPipelineBarrier(commandBuffer, sourceStage, destinationStage, 0, 0, nullptr, 0, nullptr, 1,
00305     &barrier);
00306 }
```

## 8.126 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```
#include <iostream>
#include "VEngine/Core/Engine.hpp"
#include "VEngine/Utils/Parser.hpp"
Include dependency graph for main.cpp:
```



## Functions

- int `main` (const int argc, char \*argv[], char \*envp[])

## 8.126.1 Function Documentation

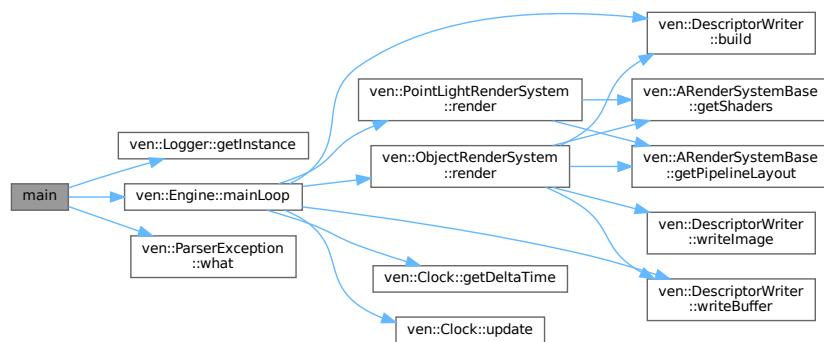
### 8.126.1.1 main()

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

Definition at line 8 of file [main.cpp](#).

References [ven::Logger::getInstance\(\)](#), [ven::Engine::mainLoop\(\)](#), and [ven::ParserException::what\(\)](#).

Here is the call graph for this function:



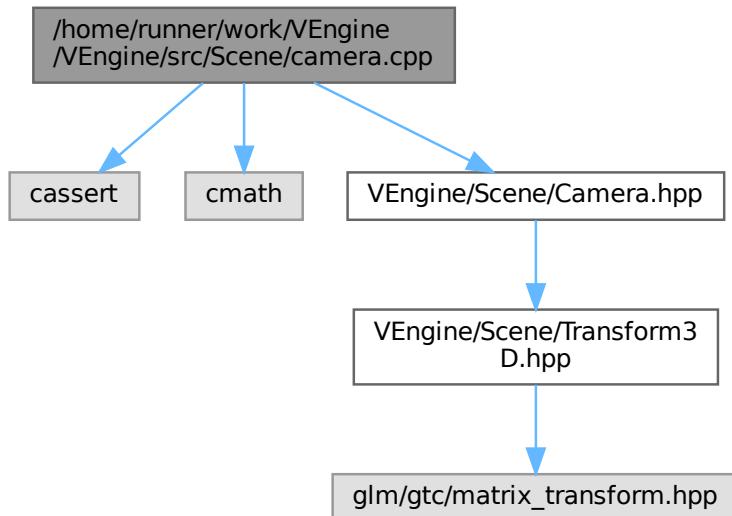
## 8.127 main.cpp

[Go to the documentation of this file.](#)

```
00001 #include <iostream>
00002
00003 #include "VEngine/Core/Engine.hpp"
00004 #include "VEngine/Utils/Parser.hpp"
00005
00006 using namespace ven;
00007
00008 int main(const int argc, char *argv[], char *envp[])
00009 {
00010     try {
00011         Logger::getInstance();
00012         Engine(Parser(argc, argv, envp).getConfig()).mainLoop();
00013     } catch (const ParserException &e) {
00014         return EXIT_SUCCESS;
00015     } catch (const std::exception &e) {
00016         std::cerr << e.what() << '\n';
00017         return EXIT_FAILURE;
00018     } catch (...) {
00019         std::cerr << "Unknown error\n";
00020         return EXIT_FAILURE;
00021     }
00022     return EXIT_SUCCESS;
00023 }
```

## 8.128 /home/runner/work/VEngine/VEngine/src/Scene/camera.cpp File Reference

```
#include <cassert>
#include <cmath>
#include "VEngine/Scene/Camera.hpp"
Include dependency graph for camera.cpp:
```



## 8.129 camera.cpp

[Go to the documentation of this file.](#)

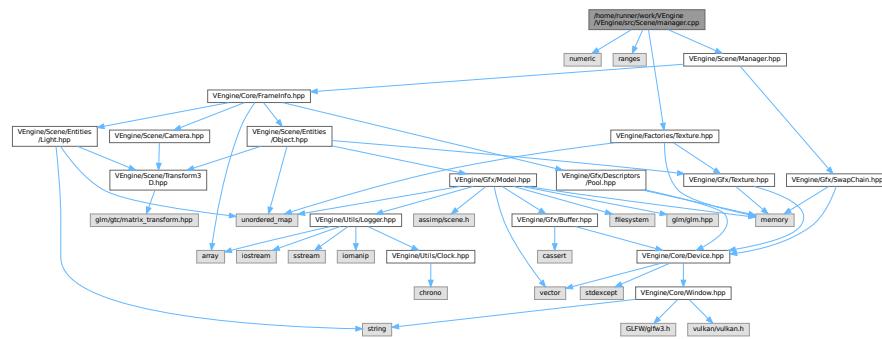
```

00001 #include <cassert>
00002 #include <cmath>
00003
00004 #include "VEngine/Scene/Camera.hpp"
00005
00006 void ven::Camera::setOrthographicProjection(const float left, const float right, const float top,
00007                                                 const float bottom, const float near, const float far)
00008 {
00009     m_projectionMatrix = glm::mat4(1.0F);
00010     m_projectionMatrix[0][0] = 2.F / (right - left);
00011     m_projectionMatrix[1][1] = 2.F / (top - bottom);
00012     m_projectionMatrix[2][2] = 1.F / (far - near);
00013     m_projectionMatrix[3][0] = -(right + left) / (right - left);
00014     m_projectionMatrix[3][1] = -(bottom + top) / (top - bottom);
00015     m_projectionMatrix[3][2] = -near / (far - near);
00016 }
00017 void ven::Camera::setPerspectiveProjection(const float aspect)
00018 {
00019     assert(glm::abs(aspect - std::numeric_limits<float>::epsilon()) > 0.0F);
00020     const float tanHalfFov = std::tan(m_fov / 2.F);
00021     m_projectionMatrix = glm::mat4(0.0F);
00022     m_projectionMatrix[0][0] = 1.F / (aspect * tanHalfFov);
00023     m_projectionMatrix[1][1] = 1.F / (tanHalfFov);
00024     m_projectionMatrix[2][2] = m_far / (m_far - m_near);
00025     m_projectionMatrix[2][3] = 1.F;
00026     m_projectionMatrix[3][2] = -(m_far * m_near) / (m_far - m_near);
00027 }
```

```
00028
00029 void ven::Camera::setViewDirection(const glm::vec3 position, const glm::vec3 direction, const
00030   glm::vec3 up)
00031 {
00032   const glm::vec3 w{normalize(direction)};
00033   const glm::vec3 u{normalize(cross(w, up))};
00034   const glm::vec3 v{cross(w, u)};
00035
00036   m_viewMatrix = glm::mat4{1.F};
00037   m_viewMatrix[0][0] = u.x;
00038   m_viewMatrix[1][0] = u.y;
00039   m_viewMatrix[2][0] = u.z;
00040   m_viewMatrix[0][1] = v.x;
00041   m_viewMatrix[1][1] = v.y;
00042   m_viewMatrix[2][1] = v.z;
00043   m_viewMatrix[0][2] = w.x;
00044   m_viewMatrix[1][2] = w.y;
00045   m_viewMatrix[2][2] = w.z;
00046   m_viewMatrix[3][0] = -dot(u, position);
00047   m_viewMatrix[3][1] = -dot(v, position);
00048   m_viewMatrix[3][2] = -dot(w, position);
00049
00050   m_inverseViewMatrix = glm::mat4{1.F};
00051   m_inverseViewMatrix[0][0] = u.x;
00052   m_inverseViewMatrix[0][1] = u.y;
00053   m_inverseViewMatrix[0][2] = u.z;
00054   m_inverseViewMatrix[1][0] = v.x;
00055   m_inverseViewMatrix[1][1] = v.y;
00056   m_inverseViewMatrix[1][2] = v.z;
00057   m_inverseViewMatrix[2][0] = w.x;
00058   m_inverseViewMatrix[2][1] = w.y;
00059   m_inverseViewMatrix[2][2] = w.z;
00060   m_inverseViewMatrix[3][0] = position.x;
00061   m_inverseViewMatrix[3][1] = position.y;
00062   m_inverseViewMatrix[3][2] = position.z;
00063 }
00064 void ven::Camera::setViewXYZ(const glm::vec3 position, const glm::vec3 rotation)
00065 {
00066   const float c3 = glm::cos(rotation.z);
00067   const float s3 = glm::sin(rotation.z);
00068   const float c2 = glm::cos(rotation.x);
00069   const float s2 = glm::sin(rotation.x);
00070   const float c1 = glm::cos(rotation.y);
00071   const float s1 = glm::sin(rotation.y);
00072   const glm::vec3 u{(c1 * c3 + s1 * s2 * s3), (c2 * s3), (c1 * s2 * s3 - c3 * s1)};
00073   const glm::vec3 v{(c3 * s1 * s2 - c1 * s3), (c2 * c3), (c1 * c3 * s2 + s1 * s3)};
00074   const glm::vec3 w{(c2 * s1), (-s2), (c1 * c2)};
00075
00076   m_viewMatrix = glm::mat4{1.F};
00077   m_viewMatrix[0][0] = u.x;
00078   m_viewMatrix[1][0] = u.y;
00079   m_viewMatrix[2][0] = u.z;
00080   m_viewMatrix[0][1] = v.x;
00081   m_viewMatrix[1][1] = v.y;
00082   m_viewMatrix[2][1] = v.z;
00083   m_viewMatrix[0][2] = w.x;
00084   m_viewMatrix[1][2] = w.y;
00085   m_viewMatrix[2][2] = w.z;
00086   m_viewMatrix[3][0] = -dot(u, position);
00087   m_viewMatrix[3][1] = -dot(v, position);
00088   m_viewMatrix[3][2] = -dot(w, position);
00089
00090   m_inverseViewMatrix = glm::mat4{1.F};
00091   m_inverseViewMatrix[0][0] = u.x;
00092   m_inverseViewMatrix[0][1] = u.y;
00093   m_inverseViewMatrix[0][2] = u.z;
00094   m_inverseViewMatrix[1][0] = v.x;
00095   m_inverseViewMatrix[1][1] = v.y;
00096   m_inverseViewMatrix[1][2] = v.z;
00097   m_inverseViewMatrix[2][0] = w.x;
00098   m_inverseViewMatrix[2][1] = w.y;
00099   m_inverseViewMatrix[2][2] = w.z;
00100   m_inverseViewMatrix[3][0] = position.x;
00101   m_inverseViewMatrix[3][1] = position.y;
00102   m_inverseViewMatrix[3][2] = position.z;
00103 }
```

## 8.130 /home/runner/work/VEngine/VEngine/src/Scene/manager.cpp File Reference

```
#include <numeric>
#include <ranges>
#include "VEngine/Scene/Manager.hpp"
#include "VEngine/Factories/Texture.hpp"
Include dependency graph for manager.cpp:
```



## 8.131 manager.cpp

[Go to the documentation of this file.](#)

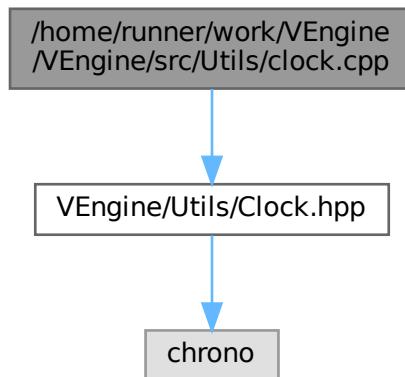
```
00001 #include <numeric>
00002 #include <ranges>
00003
00004 #include "VEngine/Scene/Manager.hpp"
00005 #include "VEngine/Factories/Texture.hpp"
00006
00007 ven::SceneManager::SceneManager(Device& device)
00008 {
00009     // including nonCoherentAtomSize allows us to flush a specific index at once
00010     unsigned long alignment = std::lcm(
00011         device.getProperties().limits.nonCoherentAtomSize,
00012         device.getProperties().limits.minUniformBufferOffsetAlignment
00013     );
00014     for (auto & uboBuffer : m_uboBuffers) {
00015         uboBuffer = std::make_unique<Buffer>(
00016             device,
00017             sizeof(ObjectBufferData),
00018             MAX_OBJECTS,
00019             VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT,
00020             VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT,
00021             alignment);
00022         uboBuffer->map();
00023     }
00024     Logger::logExecutionTime("Creating default texture", [&] {
00025         m_textureDefault = TextureFactory::create(device, "assets/textures/default.png");
00026     });
00027 }
00028
00029 void ven::SceneManager::updateBuffer(GlobalUbo &ubo, const unsigned long frameIndex, const float
frameTime)
00030 {
00031     uint8_t lightIndex = 0;
00032     const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameTime, {0.F, -1.F, 0.F});
00033
00034     for (Object& object : m_objects | std::views::values) {
00035         const ObjectBufferData data{
00036             .modelMatrix = object.transform.transformMatrix(),
00037             .normalMatrix = object.transform.normalMatrix()
00038         };
00039         m_uboBuffers.at(frameIndex)->writeToIndex(&data, object.getId());
00040         object.setBufferInfo(static_cast<int>(frameIndex),
00041             m_uboBuffers.at(frameIndex)->descriptorInfoForIndex(object.getId()));
00041     }
}
```

```

00042     for (Light &light : m_lights | std::views::values) {
00043         auto&[position, color, shininess, padding] = ubo.pointLights.at(lightIndex);
00044         light.transform.translation = glm::vec3(rotateLight * glm::vec4(light.transform.translation,
00045             light.transform.scale.x));
00046         position = glm::vec4(light.transform.translation, light.transform.scale.x);
00047         color = light.color;
00048         shininess = light.getShininess();
00049         lightIndex++;
00050     }
00051     ubo.numLights = lightIndex;
00052 }
00053
00054 void ven::SceneManager::destroyEntity(std::vector<unsigned int> *objectIds, std::vector<unsigned int>
    *lightsIds)
00055 {
00056     for (const unsigned int objectId : *objectIds) {
00057         m_objects.erase(objectId);
00058     }
00059     for (const unsigned int lightId : *lightsIds) {
00060         m_lights.erase(lightId);
00061     }
00062     objectsIds->clear();
00063     lightsIds->clear();
00064     m_destroyState = false;
00065 }
```

## 8.132 /home/runner/work/VEngine/VEngine/src/Utils/clock.cpp File Reference

#include "VEngine/Utils/Clock.hpp"  
Include dependency graph for clock.cpp:



## 8.133 clock.cpp

[Go to the documentation of this file.](#)

```

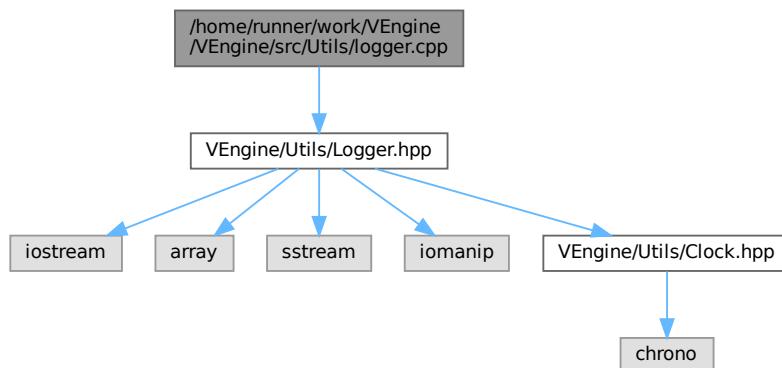
00001 #include "VEngine/Utils/Clock.hpp"
00002
00003 void ven::Clock::update()
00004 {
00005     auto newTime = std::chrono::high_resolution_clock::now();
00006     m_deltaTime = newTime - m_startTime;
00007     m_startTime = newTime;
```

```

00008 }
00009
00010 void ven::Clock::stop()
00011 {
00012     if (m_isStopped) {
00013         return;
00014     }
00015
00016     m_stopTime = std::chrono::high_resolution_clock::now();
00017     m_isStopped = true;
00018 }
00019
00020 void ven::Clock::resume()
00021 {
00022     if (!m_isStopped) {
00023         return;
00024     }
00025
00026     m_startTime += std::chrono::high_resolution_clock::now() - m_stopTime;
00027     m_isStopped = false;
00028 }
```

## 8.134 /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp File Reference

#include "VEngine/Utils/Logger.hpp"  
 Include dependency graph for logger.cpp:



## 8.135 logger.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Utils/Logger.hpp"
00002
00003 ven::Logger::Logger()
00004 {
00005     #ifdef _WIN32
00006         const HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
00007         DWORD dwMode = 0;
00008         if (hOut != INVALID_HANDLE_VALUE && (GetConsoleMode(hOut, &dwMode) != 0)) {
00009             SetConsoleMode(hOut, dwMode | ENABLE_VIRTUAL_TERMINAL_PROCESSING);
00010         }
00011     #endif
00012 }
00013
00014 std::string ven::Logger::formatLogMessage(const LogLevel level, const std::string& message)
00015 {
```

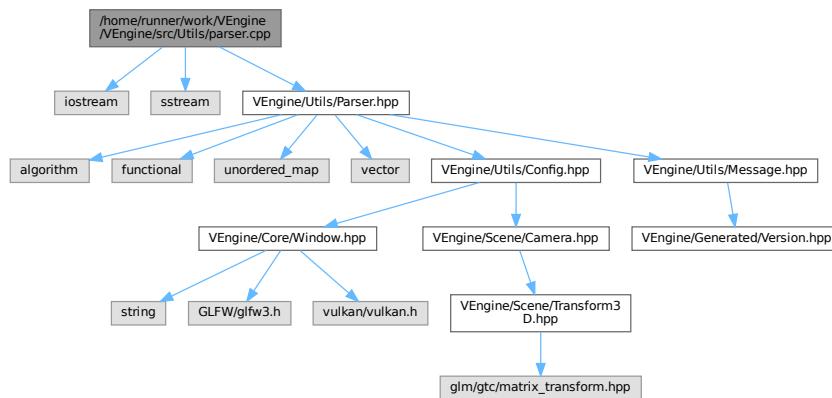
```

00016     const auto inTime = std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
00017
00018     std::ostringstream ss;
00019     ss << "[" << std::put_time(std::localtime(&inTime), "%Y-%m-%d %X") << "] ";
00020     ss << "[" << LOG_LEVEL_STRING.at(static_cast<uint8_t>(level)) << "] " << message;
00021
00022     return ss.str();
00023 }

```

## 8.136 /home/runner/work/VEngine/VEngine/src/Utils/parser.cpp File Reference

```
#include <iostream>
#include <sstream>
#include "VEngine/Utils/Parser.hpp"
Include dependency graph for parser.cpp:
```



## 8.137 parser.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002 #include <sstream>
00003
00004 #include "VEngine/Utils/Parser.hpp"
00005
00006 void ven::Parser::handleLongOption(const std::string_view& arg, const std::vector<std::string_view>&
00007     argv, size_t& index) {
00008     const auto key = arg.substr(2);
00009     if (!isValidOption(key)) {
00010         throw std::invalid_argument("Unknown option: --" + std::string(key));
00011     }
00012     index + 1 < argv.size() && !argv[index + 1].starts_with('-') ?
00013         FUNCTION_MAP_OPT_LONG.at(key)(m_config, argv[++index]) : FUNCTION_MAP_OPT_LONG.at(key)(m_config, "");
00014
00015 void ven::Parser::handleShortOptions(const std::string_view& arg, const std::vector<std::string_view>&
00016     argv, size_t& index) {
00017     const auto key = arg.substr(1);
00018     for (size_t j = 0; j < key.length(); ++j) {
00019         const std::string_view singleOpt = key.substr(j, 1);
00020         if (!isValidOption(singleOpt)) {
00021             throw std::invalid_argument("Unknown option: -" + std::string(singleOpt));
00022         }
00023         FUNCTION_MAP_OPT_SHORT.at(singleOpt)(m_config);

```

```
00024      }
00025  }
00026
00027 void ven::Parser::parseEnv(const std::unordered_map<std::string, std::string>& envp)
00028 {
00029 #ifdef __linux__
00030
00031     if (const auto display = envp.find("DISPLAY"); display == envp.end()) {
00032         throw std::runtime_error("DISPLAY environment variable not set");
00033     }
00034
00035 #elif _WIN32
00036 #endif
00037 }
00038
00039 void ven::Parser::parseArgs(const std::vector<std::string_view>& argv)
00040 {
00041     for (size_t i = 0; i < argv.size(); ++i) {
00042         const std::string_view& arg = argv[i];
00043         if (arg.empty()) { continue; }
00044
00045         if (arg.starts_with("--")) {
00046             handleLongOption(arg, argv, i);
00047         } else if (arg.starts_with("-")) {
00048             handleShortOptions(arg, argv, i);
00049         } else {
00050             throw std::invalid_argument("Unknown option: " + std::string(arg));
00051         }
00052     }
00053 }
00054
00055 ven::Parser::Parser(const int argc, char* argv[], char* envp[])
00056 {
00057     if (envp == nullptr) {
00058         throw std::runtime_error("cannot access environment variables");
00059     }
00060
00061     std::unordered_map<std::string, std::string> envVariables;
00062     for (int i = 0; envp[i] != nullptr; ++i) {
00063         const std::string env(envp[i]);
00064         if (const auto pos = env.find('='); pos != std::string_view::npos) {
00065             auto key = env.substr(0, pos);
00066             const auto value = env.substr(pos + 1);
00067             envVariables.insert({key, value});
00068         }
00069     }
00070     parseEnv(envVariables);
00071
00072     if (argc > 1) {
00073         parseArgs(std::vector<std::string_view>(argv + 1, argv + argc));
00074     }
00075 }
```



# Index

/home/runner/work/VEngine/VEngine/README.md, 349  
/home/runner/work/VEngine/VEngine/assets/shaders/fragments/lighting/pointLight.frag, 271  
/home/runner/work/VEngine/VEngine/assets/shaders/fragments/madeForWork/swapChain.frag, 271  
/home/runner/work/VEngine/VEngine/assets/shaders/vertex/lighting/lightVert.fvert, 272  
/home/runner/work/VEngine/VEngine/assets/shaders/vertex/shaderVertex.fvert, 273  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp, 274, 275  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/EntityManager.hpp, 276, 278  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp, 278, 280  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp, 280, 282  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/Game.hpp, 282, 284  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/HashCombine.hpp, 285, 286  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/Hasher.hpp, 298, 300  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/HasherSystem.hpp, 287, 288  
/home/runner/work/VEngine/VEngine/include/VEngine/Core/HasherSystem/Config.hpp, 288, 290  
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp, 291, 292  
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp, 295, 296  
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/Object.hpp, 300, 301  
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/Texture.hpp, 304, 305  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Buffer.hpp, 308, 309  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/BufferPool.hpp, 309, 310  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/DescriptorPool.hpp, 312, 314  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/DescriptorSetLayout.hpp, 315, 316  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/DescriptorWrite.hpp, 317, 318  
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Model.hpp, 296, 297  
/home/runner/work/VEngine/include/VEngine/Gfx/Renderer.hpp, 318, 320  
/home/runner/work/VEngine/include/VEngine/Gfx/Shaders.hpp, 321, 322  
/home/runner/work/VEngine/include/VEngine/Gfx/SwapChain.hpp, 323, 324  
/home/runner/work/VEngine/include/VEngine/Gfx/Texture.hpp, 306, 307  
/home/runner/work/VEngine/include/VEngine/Scene/Camera.hpp, 326, 327  
/home/runner/work/VEngine/include/VEngine/Scene/Entities/Light.hpp, 293, 294  
/home/runner/work/VEngine/include/VEngine/Scene/Entities/Object.hpp, 302, 303  
/home/runner/work/VEngine/include/VEngine/Scene/Manager.hpp, 328, 329  
/home/runner/work/VEngine/include/VEngine/Scene/Transform3D.hpp, 330, 331  
/home/runner/work/VEngine/include/VEngine/Utils/Clock.hpp, 331, 333  
/home/runner/work/VEngine/include/VEngine/Utils/Colors.hpp, 333, 335  
/home/runner/work/VEngine/include/VEngine/Utils/Config.hpp, 337, 338  
/home/runner/work/VEngine/include/VEngine/Utils/HashCombine.hpp, 339, 340  
/home/runner/work/VEngine/include/VEngine/Utils/Logger.hpp, 340, 341  
/home/runner/work/VEngine/include/VEngine/Utils/Message.hpp, 342, 343  
/home/runner/work/VEngine/include/VEngine/Utils/Parser.hpp, 344, 345  
/home/runner/work/VEngine/include/VEngine/Utils/Utils.hpp, 347, 349  
/home/runner/work/VEngine/src/Core/GUI/init.cpp, 363  
/home/runner/work/VEngine/src/Core/GUI/render.cpp, 364, 365  
/home/runner/work/VEngine/src/Core/RenderSystems/base.cpp, 370  
/home/runner/work/VEngine/src/Core/RenderSystems/object.cpp, 371  
/home/runner/work/VEngine/src/Core/RenderSystems/pointLight.cpp, 372  
/home/runner/work/VEngine/src/Core/device.cpp, 349, 351  
/home/runner/work/VEngine/src/Core/engine.cpp, 358, 359



beginFrame  
    ven::Renderer, 213  
beginSingleTimeCommands  
    ven::Device, 105  
beginSwapChainRenderPass  
    ven::Renderer, 213  
bind  
    ven::Model, 169  
    ven::Shaders, 230  
bindingDescriptions  
    ven::PipelineConfigInfo, 200  
BLACK\_3  
    ven::Colors, 77  
BLACK\_4  
    ven::Colors, 77  
BLACK\_V  
    ven::Colors, 78  
BLUE\_3  
    ven::Colors, 78  
BLUE\_4  
    ven::Colors, 78  
BLUE\_V  
    ven::Colors, 78  
Buffer  
    ven::Buffer, 36  
build  
    ven::DescriptorPool::Builder, 48  
    ven::DescriptorSetLayout::Builder, 53  
    ven::DescriptorWriter, 99  
Builder  
    ven::DescriptorPool::Builder, 48  
    ven::DescriptorSetLayout::Builder, 53  
  
Camera  
    ven::Camera, 59, 60  
camera  
    ven::Config, 88  
    ven::FrameInfo, 132  
cameraSection  
    ven::Gui, 139  
capabilities  
    ven::SwapChainSupportDetails, 245  
checkDeviceExtensionSupport  
    ven::Device, 105  
checkValidationLayerSupport  
    ven::Device, 105  
chooseSwapExtent  
    ven::SwapChain, 237  
chooseSwapPresentMode  
    ven::SwapChain, 237  
chooseSwapSurfaceFormat  
    ven::SwapChain, 237  
cleanup  
    ven::Gui, 140  
Clock  
    ven::Clock, 70, 71  
color  
    ven::Light, 156  
    ven::LightPushConstantData, 161  
                ven::Model::Vertex, 261  
                ven::PointLightData, 203  
COLOR\_MAX  
    ven, 20  
COLOR\_PRESETS\_3  
    ven::Colors, 78  
COLOR\_PRESETS\_4  
    ven::Colors, 78  
COLOR\_PRESETS\_VK  
    ven::Colors, 79  
colorBlendAttachment  
    ven::PipelineConfigInfo, 201  
colorBlendInfo  
    ven::PipelineConfigInfo, 201  
commandBuffer  
    ven::FrameInfo, 132  
compareSwapFormats  
    ven::SwapChain, 238  
copyBuffer  
    ven::Device, 105  
copyBufferToImage  
    ven::Device, 106  
create  
    ven::LightFactory, 159  
    ven::ModelFactory, 173  
    ven::ObjectFactory, 183  
    ven::TextureFactory, 256  
createBuffer  
    ven::Device, 106  
createCommandBuffers  
    ven::Renderer, 213  
createCommandPool  
    ven::Device, 106  
CreateDebugUtilsMessengerEXT  
    device.cpp, 350  
createDepthResources  
    ven::SwapChain, 238  
createFrameBuffers  
    ven::SwapChain, 238  
createGraphicsPipeline  
    ven::Shaders, 230  
createImageViews  
    ven::SwapChain, 238  
createImageWithInfo  
    ven::Device, 107  
createIndexBuffer  
    ven::Model, 169  
createInstance  
    ven::Device, 107  
createLogicalDevice  
    ven::Device, 108  
createPipeline  
    ven::ARenderSystemBase, 30  
createPipelineLayout  
    ven::ARenderSystemBase, 30  
createRenderPass  
    ven::SwapChain, 238  
createShaderModule

ven::Shaders, 231  
 createSurface  
     ven::Device, 108  
 createSwapChain  
     ven::SwapChain, 238  
 createSyncObjects  
     ven::SwapChain, 238  
 createTextureImage  
     ven::Texture, 249  
 createTextureImageView  
     ven::Texture, 249  
 createTextureSampler  
     ven::Texture, 250  
 createVertexBuffer  
     ven::Model, 169  
 createWindow  
     ven::Window, 264  
 createWindowSurface  
     ven::Window, 264  
 CYAN\_3  
     ven::Colors, 79  
 CYAN\_4  
     ven::Colors, 79  
 CYAN\_V  
     ven::Colors, 80  
 debugCallback  
     device.cpp, 350  
 DEFAULT\_AMBIENT\_LIGHT\_COLOR  
     ven, 20  
 DEFAULT\_AMBIENT\_LIGHT\_INTENSITY  
     ven, 20  
 DEFAULT\_CLEAR\_COLOR  
     ven, 20  
 DEFAULT\_CLEAR\_DEPTH  
     ven, 20  
 DEFAULT\_FAR  
     ven, 20  
 DEFAULT\_FOV  
     ven, 20  
 DEFAULT\_HEIGHT  
     ven, 21  
 DEFAULT\_KEY\_MAPPINGS  
     ven, 21  
 DEFAULT\_LIGHT\_COLOR  
     ven, 21  
 DEFAULT\_LIGHT\_INTENSITY  
     ven, 21  
 DEFAULT\_LIGHT\_RADIUS  
     ven, 21  
 DEFAULT\_LOOK\_SPEED  
     ven, 21  
 DEFAULT\_MAX\_SETS  
     ven, 21  
 DEFAULT\_MOVE\_SPEED  
     ven, 22  
 DEFAULT\_NEAR  
     ven, 22  
 DEFAULT\_POSITION  
     ven, 22  
 DEFAULT\_ROTATION  
     ven, 22  
 DEFAULT\_SHININESS  
     ven, 22  
 DEFAULT\_TITLE  
     ven, 22  
 DEFAULT\_TRANSFORM  
     ven, 23  
 DEFAULT\_WIDTH  
     ven, 23  
 defaultPipelineConfigInfo  
     ven::Shaders, 231  
 deltaTimeMS  
     ven::Gui::ClockData, 74  
 depthStencilInfo  
     ven::PipelineConfigInfo, 201  
 DESCRIPTOR\_COUNT  
     ven, 23  
 descriptorInfo  
     ven::Buffer, 36  
 descriptorInfoForIndex  
     ven::Buffer, 37  
 DescriptorPool  
     ven::DescriptorPool, 90, 91  
 DescriptorsetLayout  
     ven::DescriptorsetLayout, 95, 96  
 DescriptorWriter  
     ven::DescriptorPool, 92  
     ven::DescriptorsetLayout, 96  
     ven::DescriptorWriter, 99  
 DestroyDebugUtilsMessengerEXT  
     device.cpp, 350  
 destroyEntity  
     ven::SceneManager, 222  
 destroyLight  
     ven::SceneManager, 223  
 destroyObject  
     ven::SceneManager, 223  
 Device  
     ven::Device, 104, 105  
 device  
     ven::Device, 109  
 device.cpp  
     CreateDebugUtilsMessengerEXT, 350  
     debugCallback, 350  
     DestroyDebugUtilsMessengerEXT, 350  
 devicePropertiesSection  
     ven::Gui, 140  
 dir  
     ven::KeyAction, 149  
 draw  
     ven::Model, 170  
 duplicate  
     ven::LightFactory, 159  
     ven::ObjectFactory, 183  
 dynamicStateEnables  
     ven::PipelineConfigInfo, 201

dynamicStateInfo  
    ven::PipelineConfigInfo, 201

EDITOR  
    ven, 18

enableValidationLayers  
    ven::Device, 118

endFrame  
    ven::Renderer, 213

endSingleTimeCommands  
    ven::Device, 110

endSwapChainRenderPass  
    ven::Renderer, 214

Engine  
    ven::Engine, 122, 123

ENGINE\_STATE  
    ven, 18

EPSILON  
    ven, 23

EventManager  
    ven::EventManager, 128

eventManager.cpp  
    GLM\_ENABLE\_EXPERIMENTAL, 362

EXIT  
    ven, 18

extentAspectRatio  
    ven::SwapChain, 239

far  
    ven::CameraConf, 68

findDepthFormat  
    ven::SwapChain, 239

findMemoryType  
    ven::Device, 110

findPhysicalQueueFamilies  
    ven::Device, 110

findQueueFamilies  
    ven::Device, 111

findSupportedFormat  
    ven::Device, 111

flush  
    ven::Buffer, 37

flushIndex  
    ven::Buffer, 38

formatLogMessage  
    ven::Logger, 163

formats  
    ven::SwapChainSupportDetails, 245

fov  
    ven::CameraConf, 68

fps  
    ven::Gui::ClockData, 74

framebufferResizeCallback  
    ven::Window, 264

frameDescriptorPool  
    ven::FrameInfo, 132

frameIndex  
    ven::FrameInfo, 132

frameTime

    ven::FrameInfo, 132

freeCommandBuffers  
    ven::Renderer, 214

freeDescriptors  
    ven::DescriptorPool, 91

FUCHSIA\_3  
    ven::Colors, 80

FUCHSIA\_4  
    ven::Colors, 80

FUCHSIA\_V  
    ven::Colors, 80

fullscreen  
    ven::WindowConf, 269

FUNCTION\_MAP\_OPT\_LONG  
    ven, 23

FUNCTION\_MAP\_OPT\_SHORT  
    ven, 23

getAlignment  
    ven::Buffer, 39

getAlignmentSize  
    ven::Buffer, 39

getAspectRatio  
    ven::Renderer, 214

getAttributeDescriptions  
    ven::Model::Vertex, 260

getBindingDescriptions  
    ven::Model::Vertex, 260

getBuffer  
    ven::Buffer, 39

getBufferInfo  
    ven::Object, 177

getBufferInfoForObject  
    ven::SceneManager, 223

getBufferSize  
    ven::Buffer, 40

getClearColor  
    ven::Renderer, 214

getColorForDuration  
    ven::Logger, 163

getCommandPool  
    ven::Device, 112

getConfig  
    ven::Parser, 194

getCurrentCommandBuffer  
    ven::Renderer, 215

getDeltaTime  
    ven::Clock, 71

getDeltaTimeMS  
    ven::Clock, 71

getDescriptorPool  
    ven::DescriptorPool, 91

getDescriptorSetLayout  
    ven::DescriptorsetLayout, 96

getDestroyState  
    ven::SceneManager, 223

getDevice  
    ven::ARenderSystemBase, 31

getDiffuseMap

ven::Object, 177  
getExtent  
    ven::Texture, 250  
    ven::Window, 265  
getFar  
    ven::Camera, 60  
getFormat  
    ven::Texture, 250  
getFov  
    ven::Camera, 60  
getFPS  
    ven::Clock, 71  
getFrameBuffer  
    ven::SwapChain, 239  
getFrameIndex  
    ven::Renderer, 215  
getGLFWWindow  
    ven::Window, 265  
getGraphicsQueue  
    ven::Device, 112  
getId  
    ven::Light, 155  
    ven::Object, 177  
getImage  
    ven::Texture, 251  
getImageInfo  
    ven::Texture, 251  
getImageLayout  
    ven::Texture, 251  
getImageView  
    ven::SwapChain, 239  
    ven::Texture, 251  
getInstance  
    ven::Device, 112  
    ven::Logger, 163  
getInstanceCount  
    ven::Buffer, 40  
getInstanceSize  
    ven::Buffer, 40  
getInverseView  
    ven::Camera, 60  
getLights  
    ven::SceneManager, 223  
getLightsToRemove  
    ven::Gui, 141  
getLookSpeed  
    ven::Camera, 61  
getMappedMemory  
    ven::Buffer, 40  
getMemoryPropertyFlags  
    ven::Buffer, 40  
getModel  
    ven::Object, 178  
getMoveSpeed  
    ven::Camera, 61  
getName  
    ven::Light, 155  
    ven::Object, 178  
getNear  
    ven::Camera, 61  
getObjects  
    ven::SceneManager, 224  
getObjectsToRemove  
    ven::Gui, 141  
getPhysicalDevice  
    ven::Device, 112  
getPipelineLayout  
    ven::ARenderSystemBase, 31  
getProjection  
    ven::Camera, 62  
getProperties  
    ven::Device, 113  
getRenderPass  
    ven::SwapChain, 239  
getRequiredExtensions  
    ven::Device, 113  
getShaders  
    ven::ARenderSystemBase, 32  
getShininess  
    ven::Light, 155  
getState  
    ven::Gui, 141  
getSwapChainExtent  
    ven::SwapChain, 239  
getSwapChainImageFormat  
    ven::SwapChain, 240  
getSwapChainRenderPass  
    ven::Renderer, 216  
getSwapChainSupport  
    ven::Device, 113  
getTextureDefault  
    ven::SceneManager, 224  
getUboBuffers  
    ven::SceneManager, 224  
getUsageFlags  
    ven::Buffer, 40  
getView  
    ven::Camera, 62  
getWindow  
    ven::Renderer, 216  
GLFW\_INCLUDE\_VULKAN  
    Window.hpp, 290  
GLM\_ENABLE\_EXPERIMENTAL  
    eventManager.cpp, 362  
    model.cpp, 385  
globalDescriptorSet  
    ven::FrameInfo, 132  
graphicsFamily  
    ven::QueueFamilyIndices, 209  
graphicsFamilyHasValue  
    ven::QueueFamilyIndices, 209  
graphicsQueue  
    ven::Device, 114  
GRAY\_3  
    ven::Colors, 80  
GRAY\_4

ven::Colors, 80  
GRAY\_V  
    ven::Colors, 80  
GREEN\_3  
    ven::Colors, 81  
GREEN\_4  
    ven::Colors, 81  
GREEN\_V  
    ven::Colors, 81  
Gui  
    ven::Gui, 139  
GUI\_STATE  
    ven, 18

handleEvents  
    ven::EventManager, 128  
handleLongOption  
    ven::Parser, 194  
handleShortOptions  
    ven::Parser, 194  
hasGlfwRequiredInstanceExtensions  
    ven::Device, 114  
hashCombine  
    ven, 19  
height  
    ven::SwapChain, 240  
    ven::WindowConf, 269  
HELP\_MESSAGE  
    ven, 24  
HIDDEN  
    ven, 18

imageCount  
    ven::SwapChain, 240  
imageView  
    ven::Texture, 251  
indices  
    ven::Model::Builder, 57  
INFO  
    ven, 18  
init  
    ven::Gui, 141  
    ven::SwapChain, 240  
initStyle  
    ven::Gui, 142  
inputAssemblyInfo  
    ven::PipelineConfigInfo, 201  
inputsSection  
    ven::Gui, 143  
invalidate  
    ven::Buffer, 41  
invalidateIndex  
    ven::Buffer, 41  
inverseView  
    ven::GlobalUbo, 136  
isComplete  
    ven::QueueFamilyIndices, 208  
isDeviceSuitable  
    ven::Device, 114

isFrameInProgress  
    ven::Renderer, 216  
isKeyJustPressed  
    ven::EventManager, 128  
IsLegacyNativeDupe  
    ven::Gui::funcs, 134  
isNumeric  
    ven, 19  
isValidOption  
    ven::Parser, 194

key  
    ven::KeyAction, 149

Light  
    ven::Light, 154, 155  
LightFactory  
    ven::LightFactory, 158  
lights  
    ven::FrameInfo, 133  
lightsSection  
    ven::Gui, 143  
LIME\_3  
    ven::Colors, 81  
LIME\_4  
    ven::Colors, 81  
LIME\_V  
    ven::Colors, 81  
loadAll  
    ven::ModelFactory, 173  
    ven::TextureFactory, 256  
loadModel  
    ven::Model::Builder, 56  
loadObjects  
    ven::Engine, 123  
LOG\_LEVEL\_COLOR  
    ven::Logger, 165  
LOG\_LEVEL\_STRING  
    ven::Logger, 165  
logExecutionTime  
    ven::Logger, 164  
Logger  
    ven::Logger, 163  
LogLevel  
    ven, 18  
logWarning  
    ven::Logger, 165  
look\_speed  
    ven::CameraConf, 68  
lookDown  
    ven::KeyMappings, 151  
lookLeft  
    ven::KeyMappings, 151  
lookRight  
    ven::KeyMappings, 151  
lookUp  
    ven::KeyMappings, 151

m\_alignmentSize

m\_bindings  
 m\_buffer  
 m\_bufferInfo  
 m\_bufferSize  
 m\_camera  
 m\_clearValues  
 m\_commandBuffers  
 m\_commandPool  
 m\_config  
 m\_currentFrame  
 m\_currentFrameIndex  
 m\_currentImageIndex  
 m\_currentLightId  
 m\_currentObjId  
 m\_debugMessenger  
 m\_deltaTime  
 m\_depthImageMemory  
 m\_depthImages  
 m\_depthImageViews  
 m\_descriptor  
 m\_descriptorPool  
 m\_descriptorSetLayout  
 m\_destroyState  
 m\_device

ven::Buffer, 44  
 ven::DescriptorsetLayout, 97  
 ven::DescriptorsetLayout::Builder, 54  
 ven::Buffer, 44  
 ven::Object, 179  
 ven::Buffer, 44  
 ven::Engine, 125  
 ven::Renderer, 218  
 ven::Device, 118  
 ven::Parser, 196  
 ven::SwapChain, 241  
 ven::Renderer, 218  
 ven::SceneManager, 226  
 ven::ObjectFactory, 185  
 ven::SceneManager, 226  
 ven::Device, 118  
 ven::Clock, 73  
 ven::SwapChain, 241  
 ven::SwapChain, 241  
 ven::SwapChain, 241  
 ven::Texture, 252  
 ven::DescriptorPool, 92  
 ven::DescriptorsetLayout, 97  
 ven::SceneManager, 226  
 ven::ARenderSystemBase, 32  
 ven::Buffer, 44  
 ven::DescriptorPool, 92  
 ven::DescriptorPool::Builder, 50  
 ven::DescriptorsetLayout, 97  
 ven::DescriptorsetLayout::Builder, 54  
 ven::Device, 118

ven::Engine, 125  
 ven::Model, 171  
 ven::Renderer, 219  
 ven::Shaders, 232  
 ven::SwapChain, 242  
 ven::Texture, 252  
 m\_deviceExtensions  
 m\_diffuseMap  
 m\_extent  
 m\_far  
 m\_format  
 m\_fov  
 m\_fragShaderModule  
 m\_framebufferResized  
 m\_framePools  
 m\_globalPool  
 m\_graphicsPipeline  
 m\_graphicsQueue  
 m\_gui  
 m\_hasIndexBuffer  
 m\_height  
 m\_imageAvailableSemaphores  
 m\_imagesInFlight  
 m\_indexBuffer  
 m\_indexCount  
 m\_inFlightFences  
 m\_instance  
 m\_instanceCount  
 m\_instanceSize  
 m\_intensity  
 m\_inverseViewMatrix  
 m\_io

ven::Object, 179  
 ven::Texture, 253  
 ven::Camera, 66  
 ven::Texture, 253  
 ven::Engine, 125  
 ven::Shaders, 232  
 ven::Window, 267  
 ven::Device, 118  
 ven::Engine, 125  
 ven::Engine, 125  
 ven::Shaders, 232  
 ven::SwapChain, 242  
 ven::Device, 118  
 ven::Engine, 125  
 ven::Model, 171  
 ven::Window, 267  
 ven::SwapChain, 242  
 ven::SwapChain, 242  
 ven::Model, 171  
 ven::Buffer, 45  
 ven::Buffer, 45  
 ven::Gui, 146  
 ven::Camera, 66  
 ven::Gui, 146

m\_isFrameStarted  
    ven::Renderer, 219  
m\_isStopped  
    ven::Clock, 73  
m\_keyState  
    ven::EventManager, 130  
m\_layerCount  
    ven::Texture, 253  
m\_lightId  
    ven::Light, 156  
m\_lights  
    ven::SceneManager, 226  
m\_lightsToRemove  
    ven::Gui, 146  
m\_lookSpeed  
    ven::Camera, 66  
m\_mapped  
    ven::Buffer, 45  
m\_maxSets  
    ven::DescriptorPool::Builder, 50  
m\_memory  
    ven::Buffer, 45  
m\_memoryPropertyFlags  
    ven::Buffer, 45  
m\_mipLevels  
    ven::Texture, 253  
m\_model  
    ven::Object, 180  
m\_moveSpeed  
    ven::Camera, 66  
m\_msg  
    ven::ParserException, 199  
m\_name  
    ven::Light, 156  
    ven::Object, 180  
m\_near  
    ven::Camera, 67  
m\_objects  
    ven::SceneManager, 226  
m\_objectsToRemove  
    ven::Gui, 146  
m\_objId  
    ven::Object, 180  
m\_oldSwapChain  
    ven::SwapChain, 242  
m\_physicalDevice  
    ven::Device, 119  
m\_pipelineLayout  
    ven::ARenderSystemBase, 32  
m\_pool  
    ven::DescriptorWriter, 101  
m\_poolFlags  
    ven::DescriptorPool::Builder, 50  
m\_poolSizes  
    ven::DescriptorPool::Builder, 50  
m\_presentQueue  
    ven::Device, 119  
m\_projectionMatrix  
    ven::Camera, 67  
m\_properties  
    ven::Device, 119  
m\_renderer  
    ven::Engine, 125  
m\_renderFinishedSemaphores  
    ven::SwapChain, 242  
m\_renderPass  
    ven::SwapChain, 243  
m\_sceneManager  
    ven::Engine, 126  
m\_setLayout  
    ven::DescriptorWriter, 101  
m\_shaders  
    ven::ARenderSystemBase, 33  
m\_shininess  
    ven::Gui, 146  
    ven::Light, 156  
m\_startTime  
    ven::Clock, 73  
m\_state  
    ven::Engine, 126  
    ven::Gui, 147  
    ven::Parser, 196  
m\_stopTime  
    ven::Clock, 73  
m\_surface  
    ven::Device, 119  
m\_swapChain  
    ven::Renderer, 219  
    ven::SwapChain, 243  
m\_swapChainDepthFormat  
    ven::SwapChain, 243  
m\_swapChainExtent  
    ven::SwapChain, 243  
m\_swapChainFrameBuffers  
    ven::SwapChain, 243  
m\_swapChainImageFormat  
    ven::SwapChain, 243  
m\_swapChainImages  
    ven::SwapChain, 244  
m\_swapChainImageViews  
    ven::SwapChain, 244  
m\_textureDefault  
    ven::SceneManager, 226  
m\_textureImage  
    ven::Texture, 253  
m\_textureImageMemory  
    ven::Texture, 253  
m\_textureImageView  
    ven::Texture, 253  
m\_textureLayout  
    ven::Texture, 254  
m\_textureSampler  
    ven::Texture, 254  
m\_uboBuffers  
    ven::SceneManager, 226  
m\_usageFlags

ven::Buffer, 45  
 m\_validationLayers  
     ven::Device, 119  
 m\_vertexBuffer  
     ven::Model, 171  
 m\_vertexCount  
     ven::Model, 171  
 m\_vertShaderModule  
     ven::Shaders, 232  
 m\_viewMatrix  
     ven::Camera, 67  
 m\_width  
     ven::Window, 267  
 m\_window  
     ven::Device, 119  
     ven::Engine, 126  
     ven::Renderer, 219  
     ven::Window, 268  
 m\_windowExtent  
     ven::SwapChain, 244  
 m\_writes  
     ven::DescriptorWriter, 101  
 MAGENTA\_3  
     ven::Colors, 81  
 MAGENTA\_4  
     ven::Colors, 82  
 MAGENTA\_V  
     ven::Colors, 82  
 main  
     main.cpp, 402  
 main.cpp  
     main, 402  
 mainLoop  
     ven::Engine, 124  
 Map  
     ven::Light, 154  
     ven::Object, 176  
 map  
     ven::Buffer, 42  
 MAROON\_3  
     ven::Colors, 82  
 MAROON\_4  
     ven::Colors, 82  
 MAROON\_V  
     ven::Colors, 82  
 MAX\_FRAMES\_IN\_FLIGHT  
     ven, 24  
 MAX\_LIGHTS  
     ven, 24  
 MAX\_OBJECTS  
     ven, 24  
 Model  
     ven::Model, 168, 169  
 model.cpp  
     GLM\_ENABLE\_EXPERIMENTAL, 385  
 ModelFactory  
     ven::ModelFactory, 173  
 modelMatrix  
     ven::ObjectBufferData, 181  
     ven::ObjectPushConstantData, 186  
 move\_speed  
     ven::CameraConf, 69  
 moveBackward  
     ven::KeyMappings, 151  
 moveCamera  
     ven::EventManager, 129  
 moveDown  
     ven::KeyMappings, 151  
 moveForward  
     ven::KeyMappings, 151  
 moveLeft  
     ven::KeyMappings, 152  
 moveRight  
     ven::KeyMappings, 152  
 moveUp  
     ven::KeyMappings, 152  
 multisampleInfo  
     ven::PipelineConfigInfo, 202  
 NAVY\_3  
     ven::Colors, 82  
 NAVY\_4  
     ven::Colors, 82  
 NAVY\_V  
     ven::Colors, 83  
 near  
     ven::CameraConf, 69  
 NIGHT\_BLUE\_3  
     ven::Colors, 83  
 NIGHT\_BLUE\_4  
     ven::Colors, 83  
 NIGHT\_BLUE\_V  
     ven::Colors, 83  
 normal  
     ven::Model::Vertex, 261  
 normalMatrix  
     ven::ObjectBufferData, 181  
     ven::ObjectPushConstantData, 186  
     ven::Transform3D, 258  
 now  
     ven::Clock, 72  
 numLights  
     ven::GlobalUbo, 136  
 Object  
     ven::Object, 176, 177  
 ObjectFactory  
     ven::ObjectFactory, 183  
 ObjectRenderSystem  
     ven::ObjectRenderSystem, 189  
 objects  
     ven::FrameInfo, 133  
 objectsSection  
     ven::Gui, 143  
 OLIVE\_3  
     ven::Colors, 83  
 OLIVE\_4

ven::Colors, 83  
OLIVE\_V  
    ven::Colors, 83  
operator()  
    std::hash< ven::Model::Vertex >, 148  
operator=  
    ven::Buffer, 42  
    ven::Camera, 62  
    ven::Clock, 72  
    ven::DescriptorPool, 92  
    ven::DescriptorSetLayout, 96  
    ven::DescriptorWriter, 99  
    ven::Device, 115  
    ven::Engine, 124  
    ven::EventManager, 129  
    ven::Gui, 144  
    ven::Light, 155  
    ven::LightFactory, 159, 160  
    ven::Model, 170  
    ven::ModelFactory, 174  
    ven::Object, 178  
    ven::ObjectFactory, 184  
    ven::ObjectRenderSystem, 190  
    ven::Parser, 195  
    ven::ParserException, 198  
    ven::PipelineConfigInfo, 200  
    ven::PointLightRenderSystem, 207  
    ven::Renderer, 217  
    ven::SceneManager, 225  
    ven::Shaders, 232  
    ven::SwapChain, 240  
    ven::Texture, 251  
    ven::TextureFactory, 256, 257  
    ven::Window, 266  
operator==  
    ven::Model::Vertex, 261  
overwrite  
    ven::DescriptorWriter, 100  
padding  
    ven::PointLightData, 203  
parseArgs  
    ven::Parser, 195  
parseEnv  
    ven::Parser, 195  
Parser  
    ven::Parser, 193  
ParserException  
    ven::ParserException, 198  
PAUSED  
    ven, 18  
pickPhysicalDevice  
    ven::Device, 115  
PipelineConfigInfo  
    ven::PipelineConfigInfo, 200  
pipelineLayout  
    ven::PipelineConfigInfo, 202  
PLAYER  
    ven, 18  
PointLightRenderSystem  
    ven::PointLightRenderSystem, 206  
pointLights  
    ven::GlobalUbo, 136  
populateDebugMessengerCreateInfo  
    ven::Device, 115  
position  
    ven::LightPushConstantData, 161  
    ven::Model::Vertex, 261  
    ven::PointLightData, 203  
presentFamily  
    ven::QueueFamilyIndices, 209  
presentFamilyHasValue  
    ven::QueueFamilyIndices, 209  
presentModes  
    ven::SwapChainSupportDetails, 245  
presentQueue  
    ven::Device, 116  
printArguments  
    ven::Parser, 195  
processKeyActions  
    ven::EventManager, 129  
processMesh  
    ven::Model::Builder, 56  
processNode  
    ven::Model::Builder, 56  
PROJECT\_VERSION  
    Version.hpp, 308  
PROJECT\_VERSION\_MAJOR  
    Version.hpp, 308  
PROJECT\_VERSION\_MINOR  
    Version.hpp, 308  
PROJECT\_VERSION\_PATCH  
    Version.hpp, 309  
projection  
    ven::GlobalUbo, 136  
querySwapChainSupport  
    ven::Device, 116  
radius  
    ven::LightPushConstantData, 161  
rasterizationInfo  
    ven::PipelineConfigInfo, 202  
readFile  
    ven::Shaders, 232  
recreateSwapChain  
    ven::Renderer, 217  
RED\_3  
    ven::Colors, 84  
RED\_4  
    ven::Colors, 84  
RED\_V  
    ven::Colors, 84  
render  
    ven::ARenderSystemBase, 32  
    ven::Gui, 144  
    ven::ObjectRenderSystem, 190  
    ven::PointLightRenderSystem, 207

Renderer  
     ven::Renderer, 212  
 rendererSection  
     ven::Gui, 144  
 renderFrameWindow  
     ven::Gui, 145  
 renderPass  
     ven::PipelineConfigInfo, 202  
 renderSystemLayout  
     ven::ARenderSystemBase, 33  
 resetPool  
     ven::DescriptorPool, 92  
 resetWindowResizedFlag  
     ven::Window, 266  
 resume  
     ven::Clock, 72  
 rotation  
     ven::Transform3D, 259  
 sampler  
     ven::Texture, 252  
 scale  
     ven::Transform3D, 259  
 SceneManager  
     ven::SceneManager, 221, 222  
 setBufferInfo  
     ven::Object, 179  
 setClearValue  
     ven::Renderer, 217  
 setDestroyState  
     ven::SceneManager, 225  
 setDiffuseMap  
     ven::Object, 179  
 setFar  
     ven::Camera, 62  
 setFov  
     ven::Camera, 63  
 setFullscreen  
     ven::Window, 266  
 setLookSpeed  
     ven::Camera, 63  
 setMaxSets  
     ven::DescriptorPool::Builder, 49  
 setModel  
     ven::Object, 179  
 setMoveSpeed  
     ven::Camera, 64  
 setName  
     ven::Light, 156  
     ven::Object, 179  
 setNear  
     ven::Camera, 64  
 setOrthographicProjection  
     ven::Camera, 65  
 setPerspectiveProjection  
     ven::Camera, 65  
 setPoolFlags  
     ven::DescriptorPool::Builder, 49  
 setShininess  
     ven::Light, 156  
 setState  
     ven::Gui, 145  
 setupDebugMessenger  
     ven::Device, 116  
 setViewDirection  
     ven::Camera, 65  
 setViewTarget  
     ven::Camera, 65  
 setViewXYZ  
     ven::Camera, 66  
 setWindowIcon  
     ven::Window, 266  
 Shaders  
     ven::Shaders, 229, 230  
 SHADERS\_BIN\_PATH  
     ven, 25  
 shininess  
     ven::PointLightData, 204  
 SHOW\_EDITOR  
     ven, 18  
 SHOW\_PLAYER  
     ven, 18  
 SILVER\_3  
     ven::Colors, 84  
 SILVER\_4  
     ven::Colors, 84  
 SILVER\_V  
     ven::Colors, 84  
 SKY\_BLUE\_3  
     ven::Colors, 84  
 SKY\_BLUE\_4  
     ven::Colors, 85  
 SKY\_BLUE\_V  
     ven::Colors, 85  
 start  
     ven::Clock, 72  
 STB\_IMAGE\_IMPLEMENTATION  
     window.cpp, 373  
 std::hash< ven::Model::Vertex >, 147  
     operator(), 148  
 stop  
     ven::Clock, 72  
 submitCommandBuffers  
     ven::SwapChain, 241  
 subpass  
     ven::PipelineConfigInfo, 202  
 SUNSET\_3  
     ven::Colors, 85  
 SUNSET\_4  
     ven::Colors, 85  
 SUNSET\_V  
     ven::Colors, 85  
 surface  
     ven::Device, 117  
 SwapChain  
     ven::SwapChain, 236, 237  
 TEAL\_3

ven::Colors, 85  
TEAL\_4  
    ven::Colors, 85  
TEAL\_V  
    ven::Colors, 86  
Texture  
    ven::Texture, 248, 249  
TextureFactory  
    ven::TextureFactory, 255  
TimePoint  
    ven, 18  
toggleGui  
    ven::KeyMappings, 152  
transform  
    ven::Camera, 67  
    ven::Light, 157  
    ven::Object, 180  
transformMatrix  
    ven::Transform3D, 258  
transitionImageLayout  
    ven::Device, 117  
transitionLayout  
    ven::Texture, 252  
translation  
    ven::Transform3D, 259  
  
unmap  
    ven::Buffer, 43  
update  
    ven::Clock, 73  
updateBuffer  
    ven::SceneManager, 225  
updateDescriptor  
    ven::Texture, 252  
updateEngineState  
    ven::EventManager, 129  
uv  
    ven::Model::Vertex, 262  
  
value  
    ven::KeyAction, 149  
ven, 15  
    COLOR\_MAX, 20  
    DEFAULT\_AMBIENT\_LIGHT\_COLOR, 20  
    DEFAULT\_AMBIENT\_LIGHT\_INTENSITY, 20  
    DEFAULT\_CLEAR\_COLOR, 20  
    DEFAULT\_CLEAR\_DEPTH, 20  
    DEFAULT\_FAR, 20  
    DEFAULT\_FOV, 20  
    DEFAULT\_HEIGHT, 21  
    DEFAULT\_KEY\_MAPPINGS, 21  
    DEFAULT\_LIGHT\_COLOR, 21  
    DEFAULT\_LIGHT\_INTENSITY, 21  
    DEFAULT\_LIGHT\_RADIUS, 21  
    DEFAULT\_LOOK\_SPEED, 21  
    DEFAULT\_MAX\_SETS, 21  
    DEFAULT\_MOVE\_SPEED, 22  
    DEFAULT\_NEAR, 22  
    DEFAULT\_POSITION, 22  
  
    DEFAULT\_ROTATION, 22  
    DEFAULT\_SHININESS, 22  
    DEFAULT\_TITLE, 22  
    DEFAULT\_TRANSFORM, 23  
    DEFAULT\_WIDTH, 23  
    DESCRIPTOR\_COUNT, 23  
    EDITOR, 18  
    ENGINE\_STATE, 18  
    EPSILON, 23  
    EXIT, 18  
    FUNCTION\_MAP\_OPT\_LONG, 23  
    FUNCTION\_MAP\_OPT\_SHORT, 23  
    GUI\_STATE, 18  
    hashCombine, 19  
    HELP\_MESSAGE, 24  
    HIDDEN, 18  
    INFO, 18  
    isNumeric, 19  
    LogLevel, 18  
    MAX\_FRAMES\_IN\_FLIGHT, 24  
    MAX\_LIGHTS, 24  
    MAX\_OBJECTS, 24  
    PAUSED, 18  
    PLAYER, 18  
    SHADERS\_BIN\_PATH, 25  
    SHOW\_EDITOR, 18  
    SHOW\_PLAYER, 18  
    TimePoint, 18  
    VERSION\_MESSAGE, 25  
    WARNING, 18  
ven::ARenderSystemBase, 27  
    ~ARenderSystemBase, 29  
    ARenderSystemBase, 29  
    createPipeline, 30  
    createPipelineLayout, 30  
    getDevice, 31  
    getPipelineLayout, 31  
    getShaders, 32  
    m\_device, 32  
    m\_pipelineLayout, 32  
    m\_shaders, 33  
    render, 32  
    renderSystemLayout, 33  
ven::Buffer, 33  
    ~Buffer, 36  
    Buffer, 36  
    descriptorInfo, 36  
    descriptorInfoForIndex, 37  
    flush, 37  
    flushIndex, 38  
    getAlignment, 39  
    getAlignmentSize, 39  
    getBuffer, 39  
    getBufferSize, 40  
    getInstanceCount, 40  
    getInstanceSize, 40  
    getMappedMemory, 40  
    getMemoryPropertyFlags, 40

getUsageFlags, 40  
 invalidate, 41  
 invalidateIndex, 41  
 m\_alignmentSize, 44  
 m\_buffer, 44  
 m\_bufferSize, 44  
 m\_device, 44  
 m\_instanceCount, 45  
 m\_instanceSize, 45  
 m\_mapped, 45  
 m\_memory, 45  
 m\_memoryPropertyFlags, 45  
 m\_usageFlags, 45  
 map, 42  
 operator=, 42  
 unmap, 43  
 writeToBuffer, 43  
 writeToIndex, 43  
**ven::Camera**, 57  
 ~Camera, 59  
 Camera, 59, 60  
 getFar, 60  
 getFov, 60  
 getInverseView, 60  
 getLookSpeed, 61  
 getMoveSpeed, 61  
 getNear, 61  
 getProjection, 62  
 getView, 62  
 m\_far, 66  
 m\_fov, 66  
 m\_inverseViewMatrix, 66  
 m\_lookSpeed, 66  
 m\_moveSpeed, 66  
 m\_near, 67  
 m\_projectionMatrix, 67  
 m\_viewMatrix, 67  
 operator=, 62  
 setFar, 62  
 setFov, 63  
 setLookSpeed, 63  
 setMoveSpeed, 64  
 setNear, 64  
 setOrthographicProjection, 65  
 setPerspectiveProjection, 65  
 setViewDirection, 65  
 setViewTarget, 65  
 setViewXYZ, 66  
 transform, 67  
**ven::CameraConf**, 68  
 far, 68  
 fov, 68  
 look\_speed, 68  
 move\_speed, 69  
 near, 69  
**ven::Clock**, 69  
 ~Clock, 70  
 Clock, 70, 71  
 getDeltaTime, 71  
 getDeltaTimeMS, 71  
 getFPS, 71  
 m\_deltaTime, 73  
 m\_isStopped, 73  
 m\_startTime, 73  
 m\_stopTime, 73  
 now, 72  
 operator=, 72  
 resume, 72  
 start, 72  
 stop, 72  
 update, 73  
**ven::Colors**, 75  
 AQUA\_3, 77  
 AQUA\_4, 77  
 AQUA\_V, 77  
 BLACK\_3, 77  
 BLACK\_4, 77  
 BLACK\_V, 78  
 BLUE\_3, 78  
 BLUE\_4, 78  
 BLUE\_V, 78  
 COLOR\_PRESETS\_3, 78  
 COLOR\_PRESETS\_4, 78  
 COLOR\_PRESETS\_VK, 79  
 CYAN\_3, 79  
 CYAN\_4, 79  
 CYAN\_V, 80  
 FUCHSIA\_3, 80  
 FUCHSIA\_4, 80  
 FUCHSIA\_V, 80  
 GRAY\_3, 80  
 GRAY\_4, 80  
 GRAY\_V, 80  
 GREEN\_3, 81  
 GREEN\_4, 81  
 GREEN\_V, 81  
 LIME\_3, 81  
 LIME\_4, 81  
 LIME\_V, 81  
 MAGENTA\_3, 81  
 MAGENTA\_4, 82  
 MAGENTA\_V, 82  
 MAROON\_3, 82  
 MAROON\_4, 82  
 MAROON\_V, 82  
 NAVY\_3, 82  
 NAVY\_4, 82  
 NAVY\_V, 83  
 NIGHT\_BLUE\_3, 83  
 NIGHT\_BLUE\_4, 83  
 NIGHT\_BLUE\_V, 83  
 OLIVE\_3, 83  
 OLIVE\_4, 83  
 OLIVE\_V, 83  
 RED\_3, 84  
 RED\_4, 84

RED\_V, 84  
SILVER\_3, 84  
SILVER\_4, 84  
SILVER\_V, 84  
SKY\_BLUE\_3, 84  
SKY\_BLUE\_4, 85  
SKY\_BLUE\_V, 85  
SUNSET\_3, 85  
SUNSET\_4, 85  
SUNSET\_V, 85  
TEAL\_3, 85  
TEAL\_4, 85  
TEAL\_V, 86  
WHITE\_3, 86  
WHITE\_4, 86  
WHITE\_V, 86  
YELLOW\_3, 86  
YELLOW\_4, 86  
YELLOW\_V, 86  
ven::Config, 87  
    camera, 88  
    vsync, 88  
    window, 88  
ven::DescriptorPool, 88  
    ~DescriptorPool, 90  
    allocateDescriptor, 91  
    DescriptorPool, 90, 91  
    DescriptorWriter, 92  
    freeDescriptors, 91  
    getDescriptorPool, 91  
    m\_descriptorPool, 92  
    m\_device, 92  
    operator=, 92  
    resetPool, 92  
ven::DescriptorPool::Builder, 46  
    addPoolSize, 48  
    build, 48  
    Builder, 48  
    m\_device, 50  
    m\_maxSets, 50  
    m\_poolFlags, 50  
    m\_poolSizes, 50  
    setMaxSets, 49  
    setPoolFlags, 49  
ven::DescriptorsetLayout, 93  
    ~DescriptorsetLayout, 95  
    DescriptorsetLayout, 95, 96  
    DescriptorWriter, 96  
    getDescriptorsetLayout, 96  
    m\_bindings, 97  
    m\_descriptorsetLayout, 97  
    m\_device, 97  
    operator=, 96  
ven::DescriptorsetLayout::Builder, 51  
    addBinding, 53  
    build, 53  
    Builder, 53  
    m\_bindings, 54  
            m\_device, 54  
            ven::DescriptorWriter, 97  
                ~DescriptorWriter, 99  
                build, 99  
                DescriptorWriter, 99  
                m\_pool, 101  
                m\_setLayout, 101  
                m\_writes, 101  
                operator=, 99  
                overwrite, 100  
                writeBuffer, 100  
                writeImage, 100  
            ven::Device, 101  
                ~Device, 104  
                beginSingleTimeCommands, 105  
                checkDeviceExtensionSupport, 105  
                checkValidationLayerSupport, 105  
                copyBuffer, 105  
                copyBufferToImage, 106  
                createBuffer, 106  
                createCommandPool, 106  
                createImageWithInfo, 107  
                createInstance, 107  
                createLogicalDevice, 108  
                createSurface, 108  
                Device, 104, 105  
                device, 109  
                enableValidationLayers, 118  
                endSingleTimeCommands, 110  
                findMemoryType, 110  
                findPhysicalQueueFamilies, 110  
                findQueueFamilies, 111  
                findSupportedFormat, 111  
                getCommandPool, 112  
                getGraphicsQueue, 112  
                getInstance, 112  
                getPhysicalDevice, 112  
                getProperties, 113  
                getRequiredExtensions, 113  
                getSwapChainSupport, 113  
                graphicsQueue, 114  
                hasGlfwRequiredInstanceExtensions, 114  
                isDeviceSuitable, 114  
                m\_commandPool, 118  
                m\_debugMessenger, 118  
                m\_device, 118  
                m\_deviceExtensions, 118  
                m\_graphicsQueue, 118  
                m\_instance, 118  
                m\_physicalDevice, 119  
                m\_presentQueue, 119  
                m\_properties, 119  
                m\_surface, 119  
                m\_validationLayers, 119  
                m\_window, 119  
                operator=, 115  
                pickPhysicalDevice, 115  
                populateDebugMessengerCreateInfo, 115

presentQueue, 116  
 querySwapChainSupport, 116  
 setupDebugMessenger, 116  
 surface, 117  
 transitionImageLayout, 117  
 ven::Engine, 120  
     ~Engine, 122  
     Engine, 122, 123  
     loadObjects, 123  
     m\_camera, 125  
     m\_device, 125  
     m\_framePools, 125  
     m\_globalPool, 125  
     m\_gui, 125  
     m\_renderer, 125  
     m\_sceneManager, 126  
     m\_state, 126  
     m\_window, 126  
     mainLoop, 124  
     operator=, 124  
 ven::EventManager, 126  
     ~EventManager, 128  
     EventManager, 128  
     handleEvents, 128  
     isKeyJustPressed, 128  
     m\_keyState, 130  
     moveCamera, 129  
     operator=, 129  
     processKeyActions, 129  
     updateEngineState, 129  
 ven::FrameInfo, 130  
     camera, 132  
     commandBuffer, 132  
     frameDescriptorPool, 132  
     frameIndex, 132  
     frameTime, 132  
     globalDescriptorSet, 132  
     lights, 133  
     objects, 133  
 ven::GlobalUbo, 135  
     ambientLightColor, 136  
     inverseView, 136  
     numLights, 136  
     pointLights, 136  
     projection, 136  
     view, 136  
 ven::Gui, 137  
     ~Gui, 139  
     cameraSection, 139  
     cleanup, 140  
     devicePropertiesSection, 140  
     getLightsToRemove, 141  
     getObjectsToRemove, 141  
     getState, 141  
     Gui, 139  
     init, 141  
     initStyle, 142  
     inputsSection, 143  
     lightsSection, 143  
     m\_intensity, 146  
     m\_io, 146  
     m\_lightsToRemove, 146  
     m\_objectsToRemove, 146  
     m\_shininess, 146  
     m\_state, 147  
     objectsSection, 143  
     operator=, 144  
     render, 144  
     rendererSection, 144  
     renderFrameWindow, 145  
     setState, 145  
 ven::Gui::ClockData, 74  
     deltaTimeMS, 74  
     fps, 74  
 ven::Gui::funcs, 133  
     IsLegacyNativeDupe, 134  
 ven::KeyAction, 148  
     dir, 149  
     key, 149  
     value, 149  
 ven::KeyMappings, 150  
     lookDown, 151  
     lookLeft, 151  
     lookRight, 151  
     lookUp, 151  
     moveBackward, 151  
     moveDown, 151  
     moveForward, 151  
     moveLeft, 152  
     moveRight, 152  
     moveUp, 152  
     toggleGui, 152  
 ven::Light, 153  
     ~Light, 154  
     color, 156  
     getId, 155  
     getName, 155  
     getShininess, 155  
     Light, 154, 155  
     m\_lightId, 156  
     m\_name, 156  
     m\_shininess, 156  
     Map, 154  
     operator=, 155  
     setName, 156  
     setShininess, 156  
     transform, 157  
 ven::LightFactory, 157  
     ~LightFactory, 158  
     create, 159  
     duplicate, 159  
     LightFactory, 158  
     m\_currentLightId, 160  
     operator=, 159, 160  
 ven::LightPushConstantData, 160  
     color, 161

position, 161  
radius, 161  
ven::Logger, 161  
formatLogMessage, 163  
getColorForDuration, 163  
getInstance, 163  
LOG\_LEVEL\_COLOR, 165  
LOG\_LEVEL\_STRING, 165  
logExecutionTime, 164  
Logger, 163  
logWarning, 165  
ven::Model, 166  
~Model, 168  
bind, 169  
createIndexBuffer, 169  
createVertexBuffer, 169  
draw, 170  
m\_device, 171  
m\_hasIndexBuffer, 171  
m\_indexBuffer, 171  
m\_indexCount, 171  
m\_vertexBuffer, 171  
m\_vertexCount, 171  
Model, 168, 169  
operator=, 170  
ven::Model::Builder, 55  
indices, 57  
loadModel, 56  
processMesh, 56  
processNode, 56  
vertices, 57  
ven::Model::Vertex, 259  
color, 261  
getAttributeDescriptions, 260  
getBindingDescriptions, 260  
normal, 261  
operator==, 261  
position, 261  
uv, 262  
ven::ModelFactory, 172  
~ModelFactory, 173  
create, 173  
loadAll, 173  
ModelFactory, 173  
operator=, 174  
ven::Object, 175  
~Object, 176  
getBufferInfo, 177  
getDiffuseMap, 177  
getId, 177  
getModel, 178  
getName, 178  
m\_bufferInfo, 179  
m\_diffuseMap, 179  
m\_model, 180  
m\_name, 180  
m\_objId, 180  
Map, 176  
Object, 176, 177  
operator=, 178  
setBufferInfo, 179  
setDiffuseMap, 179  
setModel, 179  
setName, 179  
transform, 180  
ven::ObjectBufferData, 181  
modelMatrix, 181  
normalMatrix, 181  
ven::ObjectFactory, 182  
~ObjectFactory, 183  
create, 183  
duplicate, 183  
m\_currentObjId, 185  
ObjectFactory, 183  
operator=, 184  
ven::ObjectPushConstantData, 185  
modelMatrix, 186  
normalMatrix, 186  
ven::ObjectRenderSystem, 186  
ObjectRenderSystem, 189  
operator=, 190  
render, 190  
ven::Parser, 191  
~Parser, 193  
getConfig, 194  
handleLongOption, 194  
handleShortOptions, 194  
isValidOption, 194  
m\_config, 196  
m\_state, 196  
operator=, 195  
parseArgs, 195  
parseEnv, 195  
Parser, 193  
printArguments, 195  
ven::ParserException, 196  
~ParserException, 198  
m\_msg, 199  
operator=, 198  
ParserException, 198  
what, 198  
ven::PipelineConfigInfo, 199  
attributeDescriptions, 200  
bindingDescriptions, 200  
colorBlendAttachment, 201  
colorBlendInfo, 201  
depthStencilInfo, 201  
dynamicStateEnables, 201  
dynamicStateInfo, 201  
inputAssemblyInfo, 201  
multisampleInfo, 202  
operator=, 200  
PipelineConfigInfo, 200  
pipelineLayout, 202  
rasterizationInfo, 202  
renderPass, 202

subpass, 202  
 ven::PointLightData, 203  
 color, 203  
 padding, 203  
 position, 203  
 shininess, 204  
 ven::PointLightRenderSystem, 204  
 operator=, 207  
 PointLightRenderSystem, 206  
 render, 207  
 ven::QueueFamilyIndices, 208  
 graphicsFamily, 209  
 graphicsFamilyHasValue, 209  
 isComplete, 208  
 presentFamily, 209  
 presentFamilyHasValue, 209  
 ven::Renderer, 210  
 ~Renderer, 212  
 beginFrame, 213  
 beginSwapChainRenderPass, 213  
 createCommandBuffers, 213  
 endFrame, 213  
 endSwapChainRenderPass, 214  
 freeCommandBuffers, 214  
 getAspectRatio, 214  
 getClearColor, 214  
 getCurrentCommandBuffer, 215  
 getFrameIndex, 215  
 getSwapChainRenderPass, 216  
 getWindow, 216  
 isFrameInProgress, 216  
 m\_clearValues, 218  
 m\_commandBuffers, 218  
 m\_currentFrameIndex, 218  
 m\_currentImageIndex, 218  
 m\_device, 219  
 m\_isFrameStarted, 219  
 m\_swapChain, 219  
 m\_window, 219  
 operator=, 217  
 recreateSwapChain, 217  
 Renderer, 212  
 setClearValue, 217  
 ven::SceneManager, 220  
 addLight, 222  
 addObject, 222  
 destroyEntity, 222  
 destroyLight, 223  
 destroyObject, 223  
 getBufferInfoForObject, 223  
 getDestroyState, 223  
 getLights, 223  
 getObjects, 224  
 getTextureDefault, 224  
 getUboBuffers, 224  
 m\_currentLightId, 226  
 m\_currentObjId, 226  
 m\_destroyState, 226  
 m\_lights, 226  
 m\_objects, 226  
 m\_textureDefault, 226  
 m\_uboBuffers, 226  
 operator=, 225  
 SceneManager, 221, 222  
 setDestroyState, 225  
 updateBuffer, 225  
 ven::Shaders, 227  
 ~Shaders, 229  
 bind, 230  
 createGraphicsPipeline, 230  
 createShaderModule, 231  
 defaultPipelineConfigInfo, 231  
 m\_device, 232  
 m\_fragShaderModule, 232  
 m\_graphicsPipeline, 232  
 m\_vertShaderModule, 232  
 operator=, 232  
 readFile, 232  
 Shaders, 229, 230  
 ven::SwapChain, 233  
 ~SwapChain, 236  
 acquireNextImage, 237  
 chooseSwapExtent, 237  
 chooseSwapPresentMode, 237  
 chooseSwapSurfaceFormat, 237  
 compareSwapFormats, 238  
 createDepthResources, 238  
 createFrameBuffers, 238  
 createImageViews, 238  
 createRenderPass, 238  
 createSwapChain, 238  
 createSyncObjects, 238  
 extentAspectRatio, 239  
 findDepthFormat, 239  
 getFrameBuffer, 239  
 getImageView, 239  
 getRenderPass, 239  
 getSwapChainExtent, 239  
 getSwapChainImageFormat, 240  
 height, 240  
 imageCount, 240  
 init, 240  
 m\_currentFrame, 241  
 m\_depthImageMemory, 241  
 m\_depthImages, 241  
 m\_depthImageViews, 241  
 m\_device, 242  
 m\_imageAvailableSemaphores, 242  
 m\_imagesInFlight, 242  
 m\_inFlightFences, 242  
 m\_oldSwapChain, 242  
 m\_renderFinishedSemaphores, 242  
 m\_renderPass, 243  
 m\_swapChain, 243  
 m\_swapChainDepthFormat, 243  
 m\_swapChainExtent, 243

m\_swapChainFrameBuffers, 243  
m\_swapChainImageFormat, 243  
m\_swapChainImages, 244  
m\_swapChainImageViews, 244  
m\_windowExtent, 244  
operator=, 240  
submitCommandBuffers, 241  
SwapChain, 236, 237  
width, 241  
ven::SwapChainSupportDetails, 244  
capabilities, 245  
formats, 245  
presentModes, 245  
ven::Texture, 246  
~Texture, 249  
createTextureImage, 249  
createTextureImageView, 249  
createTextureSampler, 250  
getExtent, 250  
getFormat, 250  
getImage, 251  
getImageInfo, 251  
getImageLayout, 251  
getImageView, 251  
imageView, 251  
m\_descriptor, 252  
m\_device, 252  
m\_extent, 253  
m\_format, 253  
m\_layerCount, 253  
m\_mipLevels, 253  
m\_textureImage, 253  
m\_textureImageMemory, 253  
m\_textureImageView, 253  
m\_textureLayout, 254  
m\_textureSampler, 254  
operator=, 251  
sampler, 252  
Texture, 248, 249  
transitionLayout, 252  
updateDescriptor, 252  
ven::TextureFactory, 254  
~TextureFactory, 255  
create, 256  
loadAll, 256  
operator=, 256, 257  
TextureFactory, 255  
ven::Transform3D, 257  
normalMatrix, 258  
rotation, 259  
scale, 259  
transformMatrix, 258  
translation, 259  
ven::Window, 262  
~Window, 263  
createWindow, 264  
createWindowSurface, 264  
framebufferResizeCallback, 264  
getExtent, 265  
getGLFWWindow, 265  
m\_framebufferResized, 267  
m\_height, 267  
m\_width, 267  
m\_window, 268  
operator=, 266  
resetWindowResizedFlag, 266  
setFullscreen, 266  
setWindowIcon, 266  
wasWindowResized, 267  
Window, 263, 264  
ven::WindowConf, 268  
fullscreen, 269  
height, 269  
width, 269  
vengine, 1  
Version.hpp  
PROJECT\_VERSION, 308  
PROJECT\_VERSION\_MAJOR, 308  
PROJECT\_VERSION\_MINOR, 308  
PROJECT\_VERSION\_PATCH, 309  
VERSION\_MESSAGE  
ven, 25  
vertices  
ven::Model::Builder, 57  
view  
ven::GlobalUbo, 136  
vsync  
ven::Config, 88  
WARNING  
ven, 18  
wasWindowResized  
ven::Window, 267  
what  
ven::ParserException, 198  
WHITE\_3  
ven::Colors, 86  
WHITE\_4  
ven::Colors, 86  
WHITE\_V  
ven::Colors, 86  
width  
ven::SwapChain, 241  
ven::WindowConf, 269  
Window  
ven::Window, 263, 264  
window  
ven::Config, 88  
window.cpp  
STB\_IMAGE\_IMPLEMENTATION, 373  
Window.hpp  
GLFW\_INCLUDE\_VULKAN, 290  
writeBuffer  
ven::DescriptorWriter, 100  
writeImage  
ven::DescriptorWriter, 100  
writeToBuffer

ven::Buffer, [43](#)

writeToIndex

ven::Buffer, [43](#)

YELLOW\_3

ven::Colors, [86](#)

YELLOW\_4

ven::Colors, [86](#)

YELLOW\_V

ven::Colors, [86](#)