

LLGL

1.00 Alpha

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>LLGL 1.00 Alpha Documentation</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	LLGL Namespace Reference . . . . .	13
6.1.1	Typedef Documentation . . . . .	21
6.1.1.1	ByteBuffer . . . . .	21
6.1.1.2	ColorRGB . . . . .	22
6.1.1.3	ColorRGBA . . . . .	22
6.1.1.4	ColorRGBAb . . . . .	22
6.1.1.5	ColorRGBAd . . . . .	22
6.1.1.6	ColorRGBAf . . . . .	22
6.1.1.7	ColorRGBAT . . . . .	22
6.1.1.8	ColorRGBAub . . . . .	22
6.1.1.9	ColorRGBb . . . . .	22

6.1.1.10	ColorRGBd . . . . .	22
6.1.1.11	ColorRGBf . . . . .	22
6.1.1.12	ColorRGBT . . . . .	22
6.1.1.13	ColorRGBub . . . . .	22
6.1.1.14	DebugCallback . . . . .	22
6.1.1.15	Point . . . . .	22
6.1.1.16	Size . . . . .	22
6.1.2	Enumeration Type Documentation . . . . .	23
6.1.2.1	AxisDirection . . . . .	23
6.1.2.2	BlendArithmetic . . . . .	23
6.1.2.3	BlendOp . . . . .	23
6.1.2.4	BufferCPUAccess . . . . .	24
6.1.2.5	BufferUsage . . . . .	24
6.1.2.6	ClippingRange . . . . .	24
6.1.2.7	CompareOp . . . . .	25
6.1.2.8	CullMode . . . . .	25
6.1.2.9	DataType . . . . .	25
6.1.2.10	ErrorType . . . . .	26
6.1.2.11	ImageFormat . . . . .	26
6.1.2.12	Key . . . . .	26
6.1.2.13	OpenGLVersion . . . . .	30
6.1.2.14	PolygonMode . . . . .	30
6.1.2.15	PrimitiveTopology . . . . .	31
6.1.2.16	QueryType . . . . .	32
6.1.2.17	RenderConditionMode . . . . .	33
6.1.2.18	RendererInfo . . . . .	33
6.1.2.19	ScreenOrigin . . . . .	33
6.1.2.20	ShaderType . . . . .	34
6.1.2.21	ShadingLanguage . . . . .	34
6.1.2.22	StencilOp . . . . .	34

6.1.2.23	StorageBufferType . . . . .	35
6.1.2.24	SwapChainMode . . . . .	35
6.1.2.25	TextureFilter . . . . .	35
6.1.2.26	TextureFormat . . . . .	36
6.1.2.27	TextureType . . . . .	37
6.1.2.28	TextureWrap . . . . .	37
6.1.2.29	UniformType . . . . .	38
6.1.2.30	WarningType . . . . .	38
6.1.3	Function Documentation . . . . .	38
6.1.3.1	CompareSWO(const VideoDisplayMode &lhs, const VideoDisplayMode &rhs) . .	38
6.1.3.2	ConvertImageBuffer(ImageFormat srcFormat, DataType srcDataType, const void *srcBuffer, std::size_t srcBufferSize, ImageFormat dstFormat, DataType dst← DataType, std::size_t threadCount=0) . . . . .	38
6.1.3.3	DataTypeSize(const DataType dataType) . . . . .	39
6.1.3.4	ImageFormatSize(const ImageFormat imageFormat) . . . . .	39
6.1.3.5	IsCompressedFormat(const ImageFormat format) . . . . .	40
6.1.3.6	IsCompressedFormat(const TextureFormat format) . . . . .	40
6.1.3.7	IsDepthStencilFormat(const ImageFormat format) . . . . .	40
6.1.3.8	MaxColorValue() . . . . .	40
6.1.3.9	MaxColorValue< bool >() . . . . .	40
6.1.3.10	MaxColorValue< unsigned char >() . . . . .	40
6.1.3.11	NumMipLevels(const Gs::Vector3i &textureSize) . . . . .	41
6.1.3.12	operator!=(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . . .	41
6.1.3.13	operator!=(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) . . . . .	41
6.1.3.14	operator!=(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) .	41
6.1.3.15	operator!=(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . .	41
6.1.3.16	operator*(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . .	41
6.1.3.17	operator*(const Color< T, N > &lhs, const T &rhs) . . . . .	41
6.1.3.18	operator*(const T &lhs, const Color< T, N > &rhs) . . . . .	41
6.1.3.19	operator+(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . .	41
6.1.3.20	operator-(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . .	41

6.1.3.21	operator/(const Color< T, N > &lhs, const Color< T, N > &rhs)	41
6.1.3.22	operator/(const Color< T, N > &lhs, const T &rhs)	41
6.1.3.23	operator==(const VertexAttribute &lhs, const VertexAttribute &rhs)	41
6.1.3.24	operator==(const VideoDisplayMode &lhs, const VideoDisplayMode &rhs)	41
6.1.3.25	operator==(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)	41
6.1.3.26	operator==(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)	41
6.1.3.27	operator==(const Color< T, N > &lhs, const Color< T, N > &rhs)	41
6.2	LLGL::Desktop Namespace Reference	41
6.2.1	Function Documentation	42
6.2.1.1	GetColorDepth()	42
6.2.1.2	GetResolution()	42
6.2.1.3	ResetVideoMode()	42
6.2.1.4	SetVideoMode(const VideoModeDescriptor &videoMode)	42
6.3	LLGL::Log Namespace Reference	42
6.3.1	Function Documentation	42
6.3.1.1	SetStdErr(std::ostream &stream)	42
6.3.1.2	SetStdOut(std::ostream &stream)	43
6.3.1.3	StdErr()	43
6.3.1.4	StdOut()	43
<b>7</b>	<b>Class Documentation</b>	<b>45</b>
7.1	LLGL::AntiAliasingDescriptor Struct Reference	45
7.1.1	Member Data Documentation	45
7.1.1.1	enabled	45
7.1.1.2	samples	45
7.2	LLGL::BlendDescriptor Struct Reference	45
7.2.1	Detailed Description	46
7.2.2	Member Data Documentation	46
7.2.2.1	blendEnabled	46
7.2.2.2	targets	46
7.3	LLGL::BlendTargetDescriptor Struct Reference	46

7.3.1	Detailed Description	47
7.3.2	Member Data Documentation	47
7.3.2.1	alphaArithmetic	47
7.3.2.2	colorArithmetic	47
7.3.2.3	colorMask	47
7.3.2.4	destAlpha	47
7.3.2.5	destColor	47
7.3.2.6	srcAlpha	47
7.3.2.7	srcColor	47
7.4	LLGL::ClearBuffersFlags Struct Reference	48
7.4.1	Detailed Description	48
7.4.2	Member Enumeration Documentation	48
7.4.2.1	anonymous enum	48
7.5	LLGL::Color< T, N > Class Template Reference	48
7.5.1	Detailed Description	49
7.5.2	Constructor & Destructor Documentation	49
7.5.2.1	Color()	49
7.5.2.2	Color(const Color< T, N > &rhs)	49
7.5.2.3	Color(Gs::UninitializeTag)	50
7.5.3	Member Function Documentation	50
7.5.3.1	Cast() const	50
7.5.3.2	operator*=(const Color< T, N > &rhs)	50
7.5.3.3	operator*=(const T &rhs)	50
7.5.3.4	operator+=(const Color< T, N > &rhs)	50
7.5.3.5	operator-() const	50
7.5.3.6	operator-=(const Color< T, N > &rhs)	50
7.5.3.7	operator/=(const Color< T, N > &rhs)	50
7.5.3.8	operator/=(const T &rhs)	50
7.5.3.9	operator[](std::size_t component)	50
7.5.3.10	operator[](std::size_t component) const	50

7.5.3.11	Ptr()	51
7.5.3.12	Ptr() const	51
7.5.4	Member Data Documentation	51
7.5.4.1	components	51
7.6	LLGL::Color< T, 3u > Class Template Reference	51
7.6.1	Detailed Description	52
7.6.2	Constructor & Destructor Documentation	52
7.6.2.1	Color()	52
7.6.2.2	Color(const Color< T, 3 > &rhs)	52
7.6.2.3	Color(const T &scalar)	52
7.6.2.4	Color(const T &r, const T &g, const T &b)	52
7.6.2.5	Color(Gs::UninitializeTag)	52
7.6.3	Member Function Documentation	52
7.6.3.1	Cast() const	52
7.6.3.2	operator*=(const Color< T, 3 > &rhs)	53
7.6.3.3	operator*=(const T &rhs)	53
7.6.3.4	operator+=(const Color< T, 3 > &rhs)	53
7.6.3.5	operator-() const	53
7.6.3.6	operator-=(const Color< T, 3 > &rhs)	53
7.6.3.7	operator/=(const Color< T, 3 > &rhs)	53
7.6.3.8	operator/=(const T &rhs)	53
7.6.3.9	operator[](std::size_t component)	53
7.6.3.10	operator[](std::size_t component) const	53
7.6.3.11	Ptr()	53
7.6.3.12	Ptr() const	54
7.6.4	Member Data Documentation	54
7.6.4.1	b	54
7.6.4.2	components	54
7.6.4.3	g	54
7.6.4.4	r	54



7.7	LLGL::Color< T, 4u > Class Template Reference . . . . .	54
7.7.1	Detailed Description . . . . .	55
7.7.2	Constructor & Destructor Documentation . . . . .	55
7.7.2.1	Color() . . . . .	55
7.7.2.2	Color(const Color< T, 4 > &rhs) . . . . .	55
7.7.2.3	Color(const T &brightness) . . . . .	55
7.7.2.4	Color(const T &r, const T &g, const T &b) . . . . .	55
7.7.2.5	Color(const T &r, const T &g, const T &b, const T &a) . . . . .	55
7.7.2.6	Color(Gs::UninitializeTag) . . . . .	55
7.7.3	Member Function Documentation . . . . .	55
7.7.3.1	Cast() const . . . . .	55
7.7.3.2	operator*=(const Color< T, 4 > &rhs) . . . . .	56
7.7.3.3	operator*=(const T &rhs) . . . . .	56
7.7.3.4	operator+=(const Color< T, 4 > &rhs) . . . . .	56
7.7.3.5	operator-() const . . . . .	56
7.7.3.6	operator-=(const Color< T, 4 > &rhs) . . . . .	56
7.7.3.7	operator/=(const Color< T, 4 > &rhs) . . . . .	56
7.7.3.8	operator/=(const T &rhs) . . . . .	56
7.7.3.9	operator[](std::size_t component) . . . . .	56
7.7.3.10	operator[](std::size_t component) const . . . . .	56
7.7.3.11	Ptr() . . . . .	56
7.7.3.12	Ptr() const . . . . .	57
7.7.4	Member Data Documentation . . . . .	57
7.7.4.1	a . . . . .	57
7.7.4.2	b . . . . .	57
7.7.4.3	components . . . . .	57
7.7.4.4	g . . . . .	57
7.7.4.5	r . . . . .	57
7.8	LLGL::ComputePipeline Class Reference . . . . .	57
7.8.1	Detailed Description . . . . .	57

7.8.2	Constructor & Destructor Documentation . . . . .	57
7.8.2.1	~ComputePipeline() . . . . .	57
7.9	LLGL::ComputePipelineDescriptor Struct Reference . . . . .	58
7.9.1	Detailed Description . . . . .	58
7.9.2	Constructor & Destructor Documentation . . . . .	58
7.9.2.1	ComputePipelineDescriptor()=default . . . . .	58
7.9.2.2	ComputePipelineDescriptor(ShaderProgram *shaderProgram) . . . . .	58
7.9.3	Member Data Documentation . . . . .	58
7.9.3.1	shaderProgram . . . . .	58
7.10	LLGL::ConstantBuffer Class Reference . . . . .	59
7.10.1	Detailed Description . . . . .	59
7.10.2	Constructor & Destructor Documentation . . . . .	59
7.10.2.1	~ConstantBuffer() . . . . .	59
7.11	LLGL::ConstantBufferDescriptor Struct Reference . . . . .	59
7.11.1	Detailed Description . . . . .	59
7.11.2	Constructor & Destructor Documentation . . . . .	60
7.11.2.1	ConstantBufferDescriptor()=default . . . . .	60
7.11.2.2	ConstantBufferDescriptor(unsigned int size, BufferUsage usage) . . . . .	60
7.11.3	Member Data Documentation . . . . .	60
7.11.3.1	size . . . . .	60
7.11.3.2	usage . . . . .	60
7.12	LLGL::ConstantBufferViewDescriptor Struct Reference . . . . .	60
7.12.1	Detailed Description . . . . .	60
7.12.2	Member Data Documentation . . . . .	61
7.12.2.1	index . . . . .	61
7.12.2.2	name . . . . .	61
7.12.2.3	size . . . . .	61
7.13	LLGL::RenderingProfiler::Counter Class Reference . . . . .	61
7.13.1	Member Typedef Documentation . . . . .	61
7.13.1.1	ValueType . . . . .	61

7.13.2	Member Function Documentation	61
7.13.2.1	Count() const	61
7.13.2.2	Inc()	61
7.13.2.3	Inc(ValueType value)	61
7.13.2.4	operator unsigned int() const	61
7.13.2.5	Reset()	61
7.14	LLGL::DepthDescriptor Struct Reference	62
7.14.1	Detailed Description	62
7.14.2	Member Data Documentation	62
7.14.2.1	compareOp	62
7.14.2.2	testEnabled	62
7.14.2.3	writeEnabled	62
7.15	LLGL::Window::EventListener Class Reference	62
7.15.1	Constructor & Destructor Documentation	63
7.15.1.1	~EventListener()	63
7.15.2	Member Function Documentation	63
7.15.2.1	OnChar(Window &sender, wchar_t chr)	63
7.15.2.2	OnDoubleClick(Window &sender, Key keyCode)	63
7.15.2.3	OnGlobalMotion(Window &sender, const Point &motion)	63
7.15.2.4	OnKeyDown(Window &sender, Key keyCode)	63
7.15.2.5	OnKeyUp(Window &sender, Key keyCode)	63
7.15.2.6	OnLocalMotion(Window &sender, const Point &position)	63
7.15.2.7	OnProcessEvents(Window &sender)	63
7.15.2.8	OnQuit(Window &sender)	63
7.15.2.9	OnResize(Window &sender, const Size &clientAreaSize)	64
7.15.2.10	OnWheelMotion(Window &sender, int motion)	64
7.15.3	Friends And Related Function Documentation	64
7.15.3.1	Window	64
7.16	LLGL::ShaderSource::GLSL Struct Reference	64
7.16.1	Detailed Description	64

7.16.2	Member Data Documentation . . . . .	64
7.16.2.1	sourceCode . . . . .	64
7.17	LLGL::GraphicsAPIDependentStateDescriptor Union Reference . . . . .	64
7.17.1	Detailed Description . . . . .	65
7.17.2	Constructor & Destructor Documentation . . . . .	65
7.17.2.1	GraphicsAPIDependentStateDescriptor() . . . . .	65
7.17.3	Member Data Documentation . . . . .	65
7.17.3.1	stateOpenGL . . . . .	65
7.18	LLGL::GraphicsPipeline Class Reference . . . . .	65
7.18.1	Detailed Description . . . . .	66
7.18.2	Constructor & Destructor Documentation . . . . .	66
7.18.2.1	~GraphicsPipeline() . . . . .	66
7.19	LLGL::GraphicsPipelineDescriptor Struct Reference . . . . .	66
7.19.1	Detailed Description . . . . .	66
7.19.2	Member Data Documentation . . . . .	66
7.19.2.1	blend . . . . .	66
7.19.2.2	depth . . . . .	67
7.19.2.3	primitiveTopology . . . . .	67
7.19.2.4	rasterizer . . . . .	67
7.19.2.5	shaderProgram . . . . .	67
7.19.2.6	stencil . . . . .	67
7.20	LLGL::ShaderSource::HLSL Struct Reference . . . . .	67
7.20.1	Detailed Description . . . . .	68
7.20.2	Member Data Documentation . . . . .	68
7.20.2.1	entryPoint . . . . .	68
7.20.2.2	flags . . . . .	68
7.20.2.3	sourceCode . . . . .	68
7.20.2.4	target . . . . .	68
7.21	LLGL::ImageDescriptor Struct Reference . . . . .	68
7.21.1	Detailed Description . . . . .	69

7.21.2	Constructor & Destructor Documentation . . . . .	69
7.21.2.1	ImageDescriptor()=default . . . . .	69
7.21.2.2	ImageDescriptor(ImageFormat format, DataType dataType, const void *buffer) . . . . .	69
7.21.2.3	ImageDescriptor(ImageFormat format, const void *buffer, unsigned int compressedSize) . . . . .	69
7.21.3	Member Data Documentation . . . . .	69
7.21.3.1	buffer . . . . .	69
7.21.3.2	compressedSize . . . . .	69
7.21.3.3	dataType . . . . .	70
7.21.3.4	format . . . . .	70
7.22	LLGL::IndexBuffer Class Reference . . . . .	70
7.22.1	Detailed Description . . . . .	70
7.22.2	Constructor & Destructor Documentation . . . . .	70
7.22.2.1	~IndexBuffer() . . . . .	70
7.22.3	Member Function Documentation . . . . .	70
7.22.3.1	GetIndexFormat() const . . . . .	70
7.22.3.2	SetIndexFormat(const IndexFormat &indexFormat) . . . . .	70
7.23	LLGL::IndexBufferDescriptor Struct Reference . . . . .	71
7.23.1	Detailed Description . . . . .	71
7.23.2	Constructor & Destructor Documentation . . . . .	71
7.23.2.1	IndexBufferDescriptor()=default . . . . .	71
7.23.2.2	IndexBufferDescriptor(unsigned int size, BufferUsage usage, const IndexFormat &indexFormat) . . . . .	71
7.23.3	Member Data Documentation . . . . .	71
7.23.3.1	indexFormat . . . . .	71
7.23.3.2	size . . . . .	72
7.23.3.3	usage . . . . .	72
7.24	LLGL::IndexFormat Class Reference . . . . .	72
7.24.1	Constructor & Destructor Documentation . . . . .	72
7.24.1.1	IndexFormat()=default . . . . .	72
7.24.1.2	IndexFormat(const DataType dataType) . . . . .	72

7.24.2	Member Function Documentation	72
7.24.2.1	GetDataType() const	72
7.24.2.2	GetFormatSize() const	72
7.25	LLGL::Input Class Reference	73
7.25.1	Constructor & Destructor Documentation	73
7.25.1.1	Input()	73
7.25.2	Member Function Documentation	73
7.25.2.1	GetEnteredChars() const	73
7.25.2.2	GetMouseMotion() const	73
7.25.2.3	GetMousePosition() const	74
7.25.2.4	GetWheelMotion() const	74
7.25.2.5	KeyDoubleClick(Key keyCode) const	74
7.25.2.6	KeyDown(Key keyCode) const	74
7.25.2.7	KeyPressed(Key keyCode) const	74
7.25.2.8	KeyUp(Key keyCode) const	74
7.26	LLGL::RenderingDebugger::Message Class Reference	74
7.26.1	Detailed Description	75
7.26.2	Constructor & Destructor Documentation	75
7.26.2.1	Message()=default	75
7.26.2.2	Message(const Message &)=default	75
7.26.2.3	Message(const std::string &text, const std::string &source)	75
7.26.3	Member Function Documentation	75
7.26.3.1	Block()	75
7.26.3.2	BlockAfter(std::size_t occurrences)	75
7.26.3.3	GetOccurrences() const	75
7.26.3.4	GetSource() const	75
7.26.3.5	GetText() const	75
7.26.3.6	IncOccurrence()	75
7.26.3.7	IsBlocked() const	75
7.26.3.8	operator=(const Message &)=default	75

7.26.4 Friends And Related Function Documentation . . . . .	75
7.26.4.1 RenderingDebugger . . . . .	75
7.27 LLGL::NativeContextHandle Struct Reference . . . . .	76
7.27.1 Detailed Description . . . . .	76
7.27.2 Member Data Documentation . . . . .	76
7.27.2.1 colorMap . . . . .	76
7.27.2.2 display . . . . .	76
7.27.2.3 parentWindow . . . . .	76
7.27.2.4 parentWindow . . . . .	76
7.27.2.5 parentWindow . . . . .	76
7.27.2.6 screen . . . . .	76
7.27.2.7 visual . . . . .	76
7.28 LLGL::NativeHandle Struct Reference . . . . .	77
7.28.1 Detailed Description . . . . .	77
7.28.2 Member Data Documentation . . . . .	77
7.28.2.1 display . . . . .	77
7.28.2.2 visual . . . . .	77
7.28.2.3 window . . . . .	77
7.28.2.4 window . . . . .	77
7.28.2.5 window . . . . .	77
7.29 LLGL::ProfileOpenGLDescriptor Struct Reference . . . . .	77
7.29.1 Member Data Documentation . . . . .	78
7.29.1.1 coreProfile . . . . .	78
7.29.1.2 debugDump . . . . .	78
7.29.1.3 extProfile . . . . .	78
7.29.1.4 version . . . . .	78
7.30 LLGL::Query Class Reference . . . . .	78
7.30.1 Detailed Description . . . . .	79
7.30.2 Constructor & Destructor Documentation . . . . .	79
7.30.2.1 Query(const Query &)=delete . . . . .	79

7.30.2.2	<code>~Query()</code>	79
7.30.2.3	<code>Query(QueryType type)</code>	79
7.30.3	Member Function Documentation	79
7.30.3.1	<code>GetType() const</code>	79
7.30.3.2	<code>operator=(const Query &amp;)=delete</code>	79
7.31	<code>LLGL::QueryDescriptor</code> Struct Reference	79
7.31.1	Detailed Description	80
7.31.2	Member Data Documentation	80
7.31.2.1	<code>renderCondition</code>	80
7.31.2.2	<code>type</code>	80
7.32	<code>LLGL::RasterizerDescriptor</code> Struct Reference	80
7.32.1	Detailed Description	81
7.32.2	Member Data Documentation	81
7.32.2.1	<code>antiAliasedLineEnabled</code>	81
7.32.2.2	<code>conservativeRasterization</code>	81
7.32.2.3	<code>cullMode</code>	81
7.32.2.4	<code>depthBias</code>	81
7.32.2.5	<code>depthBiasClamp</code>	81
7.32.2.6	<code>depthClampEnabled</code>	81
7.32.2.7	<code>frontCCW</code>	81
7.32.2.8	<code>multiSampleEnabled</code>	81
7.32.2.9	<code>polygonMode</code>	81
7.32.2.10	<code>samples</code>	82
7.32.2.11	<code>scissorTestEnabled</code>	82
7.32.2.12	<code>slopeScaledDepthBias</code>	82
7.33	<code>LLGL::RenderContext</code> Class Reference	82
7.33.1	Detailed Description	84
7.33.2	Constructor & Destructor Documentation	84
7.33.2.1	<code>RenderContext(const RenderContext &amp;)=delete</code>	84
7.33.2.2	<code>~RenderContext()</code>	84



7.33.2.3	RenderContext()=default . . . . .	84
7.33.3	Member Function Documentation . . . . .	84
7.33.3.1	BeginQuery(Query &query)=0 . . . . .	84
7.33.3.2	BeginRenderCondition(Query &query, const RenderConditionMode mode)=0 . . . . .	85
7.33.3.3	ClearBuffers(long flags)=0 . . . . .	85
7.33.3.4	DispatchCompute(const Gs::Vector3ui &threadGroupSize)=0 . . . . .	86
7.33.3.5	Draw(unsigned int numVertices, unsigned int firstVertex)=0 . . . . .	86
7.33.3.6	DrawIndexed(unsigned int numVertices, unsigned int firstIndex)=0 . . . . .	86
7.33.3.7	DrawIndexed(unsigned int numVertices, unsigned int firstIndex, int vertexOffset)=0 . . . . .	87
7.33.3.8	DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex)=0 . . . . .	87
7.33.3.9	DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset)=0 . . . . .	87
7.33.3.10	DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset, unsigned int instanceOffset)=0 . . . . .	87
7.33.3.11	DrawInstanced(unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances)=0 . . . . .	87
7.33.3.12	DrawInstanced(unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances, unsigned int instanceOffset)=0 . . . . .	88
7.33.3.13	EndQuery(Query &query)=0 . . . . .	88
7.33.3.14	EndRenderCondition()=0 . . . . .	88
7.33.3.15	GetVideoMode() const . . . . .	88
7.33.3.16	GetWindow() const . . . . .	88
7.33.3.17	MapStorageBuffer(StorageBuffer &storageBuffer, const BufferCPUAccess access)=0 . . . . .	88
7.33.3.18	operator=(const RenderContext &)=delete . . . . .	89
7.33.3.19	Present()=0 . . . . .	89
7.33.3.20	QueryResult(Query &query, std::uint64_t &result)=0 . . . . .	89
7.33.3.21	SetClearColor(const ColorRGBAf &color)=0 . . . . .	89
7.33.3.22	SetClearDepth(float depth)=0 . . . . .	89
7.33.3.23	SetClearStencil(int stencil)=0 . . . . .	90
7.33.3.24	SetComputePipeline(ComputePipeline &computePipeline)=0 . . . . .	90

7.33.3.25 SetConstantBuffer(ConstantBuffer &constantBuffer, unsigned int slot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . . .	90
7.33.3.26 SetGraphicsAPIDependentState(const GraphicsAPIDependentStateDescriptor &state)=0 . . . . .	90
7.33.3.27 SetGraphicsPipeline(GraphicsPipeline &graphicsPipeline)=0 . . . . .	91
7.33.3.28 SetIndexBuffer(IndexBuffer &indexBuffer)=0 . . . . .	91
7.33.3.29 SetRenderTarget(RenderTarget &renderTarget)=0 . . . . .	91
7.33.3.30 SetSampler(Sampler &sampler, unsigned int slot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . . .	92
7.33.3.31 SetScissor(const Scissor &scissor)=0 . . . . .	92
7.33.3.32 SetScissorArray(const std::vector< Scissor > &scissors)=0 . . . . .	92
7.33.3.33 SetStorageBuffer(StorageBuffer &storageBuffer, unsigned int slot)=0 . . . . .	93
7.33.3.34 SetTexture(Texture &texture, unsigned int slot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . . .	93
7.33.3.35 SetVertexBuffer(VertexBuffer &vertexBuffer)=0 . . . . .	93
7.33.3.36 SetVideoMode(const VideoModeDescriptor &videoModeDesc) . . . . .	94
7.33.3.37 SetViewport(const Viewport &viewport)=0 . . . . .	94
7.33.3.38 SetViewportArray(const std::vector< Viewport > &viewports)=0 . . . . .	94
7.33.3.39 SetVsync(const VsyncDescriptor &vsyncDesc)=0 . . . . .	95
7.33.3.40 SetWindow(const std::shared_ptr< Window > &window, VideoModeDescriptor &videoModeDesc, const void *windowContext) . . . . .	95
7.33.3.41 ShareWindowAndVideoMode(RenderContext &other) . . . . .	95
7.33.3.42 SyncGPU()=0 . . . . .	95
7.33.3.43 UnmapStorageBuffer()=0 . . . . .	95
7.33.3.44 UnsetRenderTarget()=0 . . . . .	95
7.34 LLGL::RenderContextDescriptor Struct Reference . . . . .	96
7.34.1 Member Data Documentation . . . . .	96
7.34.1.1 antiAliasing . . . . .	96
7.34.1.2 debugCallback . . . . .	96
7.34.1.3 profileOpenGL . . . . .	96
7.34.1.4 videoMode . . . . .	96
7.34.1.5 vsync . . . . .	96
7.35 LLGL::RenderingCaps Struct Reference . . . . .	97

7.35.1 Detailed Description . . . . .	98
7.35.2 Member Data Documentation . . . . .	98
7.35.2.1 clippingRange . . . . .	98
7.35.2.2 has3DTextures . . . . .	98
7.35.2.3 hasComputeShaders . . . . .	98
7.35.2.4 hasConservativeRasterization . . . . .	98
7.35.2.5 hasConstantBuffers . . . . .	99
7.35.2.6 hasCubeTextureArrays . . . . .	99
7.35.2.7 hasCubeTextures . . . . .	99
7.35.2.8 hasGeometryShaders . . . . .	99
7.35.2.9 hasGLSL . . . . .	99
7.35.2.10 hasHLSL . . . . .	99
7.35.2.11 hasInstancing . . . . .	99
7.35.2.12 hasOffsetInstancing . . . . .	99
7.35.2.13 hasRenderTargetes . . . . .	99
7.35.2.14 hasSamplers . . . . .	100
7.35.2.15 hasStorageBuffers . . . . .	100
7.35.2.16 hasTessellationShaders . . . . .	100
7.35.2.17 hasTextureArrays . . . . .	100
7.35.2.18 hasUniforms . . . . .	100
7.35.2.19 hasViewportArrays . . . . .	100
7.35.2.20 max1DTextureSize . . . . .	100
7.35.2.21 max2DTextureSize . . . . .	100
7.35.2.22 max3DTextureSize . . . . .	100
7.35.2.23 maxAnisotropy . . . . .	100
7.35.2.24 maxComputeShaderWorkGroupSize . . . . .	101
7.35.2.25 maxConstantBufferSize . . . . .	101
7.35.2.26 maxCubeTextureSize . . . . .	101
7.35.2.27 maxNumComputeShaderWorkGroups . . . . .	101
7.35.2.28 maxNumRenderTargetAttachments . . . . .	101

7.35.2.29	maxNumTextureArrayLayers	101
7.35.2.30	maxPatchVertices	101
7.35.2.31	screenOrigin	101
7.36	LLGL::RenderingDebugger Class Reference	102
7.36.1	Detailed Description	102
7.36.2	Constructor & Destructor Documentation	102
7.36.2.1	~RenderingDebugger()	102
7.36.2.2	RenderingDebugger()=default	102
7.36.3	Member Function Documentation	102
7.36.3.1	OnError(ErrorType type, Message &message)	102
7.36.3.2	OnWarning(WarningType type, Message &message)	102
7.36.3.3	PostError(ErrorType type, const std::string &message, const std::string &source)	102
7.36.3.4	PostWarning(WarningType type, const std::string &message, const std::string &source)	103
7.37	LLGL::RenderingProfiler Class Reference	103
7.37.1	Detailed Description	104
7.37.2	Member Function Documentation	104
7.37.2.1	RecordDrawCall(const PrimitiveTopology topology, Counter::ValueType num↔ Vertices)	104
7.37.2.2	RecordDrawCall(const PrimitiveTopology topology, Counter::ValueType num↔ Vertices, Counter::ValueType numInstances)	104
7.37.2.3	ResetCounters()	104
7.37.3	Member Data Documentation	105
7.37.3.1	dispatchComputeCalls	105
7.37.3.2	drawCalls	105
7.37.3.3	mapConstantBuffer	105
7.37.3.4	mapStorageBuffer	105
7.37.3.5	renderedLines	105
7.37.3.6	renderedPatches	105
7.37.3.7	renderedPoints	105
7.37.3.8	renderedTriangles	105
7.37.3.9	setComputePipeline	105

7.37.3.10 setConstantBuffer . . . . .	105
7.37.3.11 setGraphicsPipeline . . . . .	105
7.37.3.12 setIndexBuffer . . . . .	105
7.37.3.13 setRenderTarget . . . . .	105
7.37.3.14 setSampler . . . . .	105
7.37.3.15 setStorageBuffer . . . . .	105
7.37.3.16 setTexture . . . . .	105
7.37.3.17 setVertexBuffer . . . . .	105
7.37.3.18 writeConstantBuffer . . . . .	105
7.37.3.19 writeIndexBuffer . . . . .	105
7.37.3.20 writeStorageBuffer . . . . .	105
7.37.3.21 writeVertexBuffer . . . . .	105
7.38 LLGL::RenderSystem Class Reference . . . . .	106
7.38.1 Detailed Description . . . . .	108
7.38.2 Constructor & Destructor Documentation . . . . .	108
7.38.2.1 RenderSystem(const RenderSystem &)=delete . . . . .	108
7.38.2.2 ~RenderSystem() . . . . .	108
7.38.2.3 RenderSystem()=default . . . . .	108
7.38.3 Member Function Documentation . . . . .	108
7.38.3.1 CreateComputePipeline(const ComputePipelineDescriptor &desc)=0 . . . . .	108
7.38.3.2 CreateConstantBuffer(const ConstantBufferDescriptor &desc, const void *initialData=nullptr)=0 . . . . .	109
7.38.3.3 CreateGraphicsPipeline(const GraphicsPipelineDescriptor &desc)=0 . . . . .	109
7.38.3.4 CreateIndexBuffer(const IndexBufferDescriptor &desc, const void *initial← Data=nullptr)=0 . . . . .	109
7.38.3.5 CreateQuery(const QueryDescriptor &desc)=0 . . . . .	110
7.38.3.6 CreateRenderContext(const RenderContextDescriptor &desc, const std← ::shared_ptr< Window > &window=nullptr)=0 . . . . .	110
7.38.3.7 CreateRenderTarget(unsigned int multiSamples=0)=0 . . . . .	110
7.38.3.8 CreateSampler(const SamplerDescriptor &desc)=0 . . . . .	110
7.38.3.9 CreateShader(const ShaderType type)=0 . . . . .	110
7.38.3.10 CreateShaderProgram()=0 . . . . .	111

7.38.3.11 CreateStorageBuffer(const StorageBufferDescriptor &desc, const void *initial↵ Data=nullptr)=0 . . . . .	111
7.38.3.12 CreateTexture(const TextureDescriptor &textureDesc, const ImageDescriptor *imageDesc=nullptr)=0 . . . . .	111
7.38.3.13 CreateVertexBuffer(const VertexBufferDescriptor &desc, const void *initial↵ Data=nullptr)=0 . . . . .	112
7.38.3.14 FindModules() . . . . .	112
7.38.3.15 GenerateMips(Texture &texture)=0 . . . . .	112
7.38.3.16 GetConfiguration() const . . . . .	112
7.38.3.17 GetCurrentContext() const . . . . .	112
7.38.3.18 GetDefaultTextureImageRGBAub(int numPixels) const . . . . .	113
7.38.3.19 GetName() const . . . . .	113
7.38.3.20 Load(const std::string &moduleName, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr) . . . . .	113
7.38.3.21 MakeCurrent(RenderContext *renderContext) . . . . .	113
7.38.3.22 OnMakeCurrent(RenderContext *renderContext) . . . . .	114
7.38.3.23 operator=(const RenderSystem &)=delete . . . . .	114
7.38.3.24 QueryRendererInfo() const =0 . . . . .	114
7.38.3.25 QueryRenderingCaps() const =0 . . . . .	114
7.38.3.26 QueryShadingLanguage() const =0 . . . . .	114
7.38.3.27 QueryTextureDescriptor(const Texture &texture)=0 . . . . .	114
7.38.3.28 ReadTexture(const Texture &texture, int mipLevel, ImageFormat imageFormat, DataType dataType, void *buffer)=0 . . . . .	114
7.38.3.29 Release(RenderContext &renderContext)=0 . . . . .	115
7.38.3.30 Release(VertexBuffer &vertexBuffer)=0 . . . . .	115
7.38.3.31 Release(IndexBuffer &indexBuffer)=0 . . . . .	115
7.38.3.32 Release(ConstantBuffer &constantBuffer)=0 . . . . .	115
7.38.3.33 Release(StorageBuffer &storageBuffer)=0 . . . . .	115
7.38.3.34 Release(Texture &texture)=0 . . . . .	115
7.38.3.35 Release(Sampler &sampler)=0 . . . . .	115
7.38.3.36 Release(RenderTarget &renderTarget)=0 . . . . .	115
7.38.3.37 Release(Shader &shader)=0 . . . . .	116
7.38.3.38 Release(ShaderProgram &shaderProgram)=0 . . . . .	116

7.38.3.39	<code>Release(GraphicsPipeline &amp;graphicsPipeline)=0</code>	116
7.38.3.40	<code>Release(ComputePipeline &amp;computePipeline)=0</code>	116
7.38.3.41	<code>Release(Query &amp;query)=0</code>	116
7.38.3.42	<code>SetConfiguration(const RenderSystemConfiguration &amp;config)</code>	116
7.38.3.43	<code>WriteConstantBuffer(ConstantBuffer &amp;constantBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0</code>	116
7.38.3.44	<code>WriteIndexBuffer(IndexBuffer &amp;indexBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0</code>	116
7.38.3.45	<code>WriteStorageBuffer(StorageBuffer &amp;storageBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0</code>	116
7.38.3.46	<code>WriteTexture(Texture &amp;texture, const SubTextureDescriptor &amp;subTextureDesc, const ImageDescriptor &amp;imageDesc)=0</code>	116
7.38.3.47	<code>WriteVertexBuffer(VertexBuffer &amp;vertexBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0</code>	117
7.39	<code>LLGL::RenderSystemConfiguration</code> Struct Reference	117
7.39.1	Detailed Description	117
7.39.2	Member Data Documentation	118
7.39.2.1	<code>defaultImageColor</code>	118
7.39.2.2	<code>threadCount</code>	118
7.40	<code>LLGL::RenderTarget</code> Class Reference	118
7.40.1	Detailed Description	119
7.40.2	Constructor & Destructor Documentation	119
7.40.2.1	<code>~RenderTarget()</code>	119
7.40.3	Member Function Documentation	119
7.40.3.1	<code>ApplyMipResolution(Texture &amp;texture, int mipLevel)</code>	119
7.40.3.2	<code>ApplyResolution(const Gs::Vector2i &amp;resolution)</code>	119
7.40.3.3	<code>AttachDepthBuffer(const Gs::Vector2i &amp;size)=0</code>	119
7.40.3.4	<code>AttachDepthStencilBuffer(const Gs::Vector2i &amp;size)=0</code>	119
7.40.3.5	<code>AttachStencilBuffer(const Gs::Vector2i &amp;size)=0</code>	120
7.40.3.6	<code>AttachTexture1D(Texture &amp;texture, int mipLevel=0)=0</code>	120
7.40.3.7	<code>AttachTexture1DArray(Texture &amp;texture, int layer, int mipLevel=0)=0</code>	120
7.40.3.8	<code>AttachTexture2D(Texture &amp;texture, int mipLevel=0)=0</code>	120
7.40.3.9	<code>AttachTexture2DArray(Texture &amp;texture, int layer, int mipLevel=0)=0</code>	120

7.40.3.10	AttachTexture3D(Texture &texture, int layer, int mipLevel=0)=0 . . . . .	120
7.40.3.11	AttachTextureCube(Texture &texture, const AxisDirection cubeFace, int mip↵ Level=0)=0 . . . . .	120
7.40.3.12	AttachTextureCubeArray(Texture &texture, int layer, const AxisDirection cube↵ Face, int mipLevel=0)=0 . . . . .	120
7.40.3.13	DetachTextures()=0 . . . . .	120
7.40.3.14	GetResolution() const . . . . .	121
7.40.3.15	ResetResolution() . . . . .	121
7.41	LLGL::Sampler Class Reference . . . . .	121
7.41.1	Detailed Description . . . . .	121
7.41.2	Constructor & Destructor Documentation . . . . .	121
7.41.2.1	~Sampler() . . . . .	121
7.42	LLGL::SamplerDescriptor Struct Reference . . . . .	121
7.42.1	Detailed Description . . . . .	122
7.42.2	Member Data Documentation . . . . .	122
7.42.2.1	borderColor . . . . .	122
7.42.2.2	compareOp . . . . .	122
7.42.2.3	depthCompare . . . . .	123
7.42.2.4	magFilter . . . . .	123
7.42.2.5	maxAnisotropy . . . . .	123
7.42.2.6	maxLOD . . . . .	123
7.42.2.7	minFilter . . . . .	123
7.42.2.8	minLOD . . . . .	123
7.42.2.9	mipMapFilter . . . . .	123
7.42.2.10	mipMapLODBias . . . . .	123
7.42.2.11	mipMapping . . . . .	123
7.42.2.12	textureWrapU . . . . .	123
7.42.2.13	textureWrapV . . . . .	124
7.42.2.14	textureWrapW . . . . .	124
7.43	LLGL::Scissor Struct Reference . . . . .	124
7.43.1	Detailed Description . . . . .	124



7.43.2	Constructor & Destructor Documentation . . . . .	125
7.43.2.1	Scissor()=default . . . . .	125
7.43.2.2	Scissor(const Scissor &)=default . . . . .	125
7.43.2.3	Scissor(int x, int y, int width, int height) . . . . .	125
7.43.3	Member Data Documentation . . . . .	125
7.43.3.1	height . . . . .	125
7.43.3.2	width . . . . .	125
7.43.3.3	x . . . . .	125
7.43.3.4	y . . . . .	125
7.44	LLGL::Shader Class Reference . . . . .	125
7.44.1	Detailed Description . . . . .	126
7.44.2	Constructor & Destructor Documentation . . . . .	126
7.44.2.1	~Shader() . . . . .	126
7.44.2.2	Shader(const ShaderType type) . . . . .	126
7.44.3	Member Function Documentation . . . . .	126
7.44.3.1	Compile(const ShaderSource &shaderSource)=0 . . . . .	126
7.44.3.2	Disassemble(int flags=0) . . . . .	126
7.44.3.3	GetType() const . . . . .	127
7.44.3.4	QueryInfoLog()=0 . . . . .	127
7.45	LLGL::ShaderCompileFlags Struct Reference . . . . .	127
7.45.1	Detailed Description . . . . .	127
7.45.2	Member Enumeration Documentation . . . . .	127
7.45.2.1	anonymous enum . . . . .	127
7.46	LLGL::ShaderDisassembleFlags Struct Reference . . . . .	128
7.46.1	Detailed Description . . . . .	128
7.46.2	Member Enumeration Documentation . . . . .	128
7.46.2.1	anonymous enum . . . . .	128
7.47	LLGL::ShaderProgram Class Reference . . . . .	128
7.47.1	Detailed Description . . . . .	129
7.47.2	Constructor & Destructor Documentation . . . . .	129

7.47.2.1	<code>~ShaderProgram()</code>	129
7.47.3	Member Function Documentation	129
7.47.3.1	<code>AttachShader(Shader &amp;shader)=0</code>	129
7.47.3.2	<code>BindConstantBuffer(const std::string &amp;name, unsigned int bindingIndex)=0</code>	130
7.47.3.3	<code>BindStorageBuffer(const std::string &amp;name, unsigned int bindingIndex)=0</code>	130
7.47.3.4	<code>BindVertexAttributes(const std::vector&lt; VertexAttribute &gt; &amp;vertexAttribs)=0</code>	131
7.47.3.5	<code>LinkShaders()=0</code>	131
7.47.3.6	<code>LockShaderUniform()=0</code>	132
7.47.3.7	<code>QueryConstantBuffers() const =0</code>	132
7.47.3.8	<code>QueryInfoLog()=0</code>	132
7.47.3.9	<code>QueryStorageBuffers() const =0</code>	132
7.47.3.10	<code>QueryUniforms() const =0</code>	133
7.47.3.11	<code>QueryVertexAttributes() const =0</code>	133
7.47.3.12	<code>UnlockShaderUniform()=0</code>	133
7.48	LLGL::ShaderSource Union Reference	133
7.48.1	Detailed Description	134
7.48.2	Constructor & Destructor Documentation	134
7.48.2.1	<code>ShaderSource(const std::string &amp;sourceCode)</code>	134
7.48.2.2	<code>ShaderSource(const std::string &amp;sourceCode, const std::string &amp;entryPoint, const std::string &amp;target, int flags=0)</code>	134
7.48.2.3	<code>~ShaderSource()</code>	134
7.48.3	Member Data Documentation	134
7.48.3.1	<code>sourceGLSL</code>	135
7.48.3.2	<code>sourceHLSL</code>	135
7.49	LLGL::ShaderStageFlags Struct Reference	135
7.49.1	Detailed Description	135
7.49.2	Member Enumeration Documentation	135
7.49.2.1	anonymous enum	135
7.50	LLGL::ShaderUniform Class Reference	136
7.50.1	Detailed Description	137
7.50.2	Constructor & Destructor Documentation	137

7.50.2.1	<a href="#">~ShaderUniform()</a>	137
7.50.3	<a href="#">Member Function Documentation</a>	137
7.50.3.1	<a href="#">SetUniform(int location, const int value)=0</a>	137
7.50.3.2	<a href="#">SetUniform(int location, const Gs::Vector2i &amp;value)=0</a>	137
7.50.3.3	<a href="#">SetUniform(int location, const Gs::Vector3i &amp;value)=0</a>	137
7.50.3.4	<a href="#">SetUniform(int location, const Gs::Vector4i &amp;value)=0</a>	137
7.50.3.5	<a href="#">SetUniform(int location, const float value)=0</a>	137
7.50.3.6	<a href="#">SetUniform(int location, const Gs::Vector2f &amp;value)=0</a>	137
7.50.3.7	<a href="#">SetUniform(int location, const Gs::Vector3f &amp;value)=0</a>	137
7.50.3.8	<a href="#">SetUniform(int location, const Gs::Vector4f &amp;value)=0</a>	137
7.50.3.9	<a href="#">SetUniform(int location, const Gs::Matrix2f &amp;value)=0</a>	137
7.50.3.10	<a href="#">SetUniform(int location, const Gs::Matrix3f &amp;value)=0</a>	137
7.50.3.11	<a href="#">SetUniform(int location, const Gs::Matrix4f &amp;value)=0</a>	137
7.50.3.12	<a href="#">SetUniform(const std::string &amp;name, const int value)=0</a>	137
7.50.3.13	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector2i &amp;value)=0</a>	137
7.50.3.14	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector3i &amp;value)=0</a>	137
7.50.3.15	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector4i &amp;value)=0</a>	137
7.50.3.16	<a href="#">SetUniform(const std::string &amp;name, const float value)=0</a>	138
7.50.3.17	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector2f &amp;value)=0</a>	138
7.50.3.18	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector3f &amp;value)=0</a>	138
7.50.3.19	<a href="#">SetUniform(const std::string &amp;name, const Gs::Vector4f &amp;value)=0</a>	138
7.50.3.20	<a href="#">SetUniform(const std::string &amp;name, const Gs::Matrix2f &amp;value)=0</a>	138
7.50.3.21	<a href="#">SetUniform(const std::string &amp;name, const Gs::Matrix3f &amp;value)=0</a>	138
7.50.3.22	<a href="#">SetUniform(const std::string &amp;name, const Gs::Matrix4f &amp;value)=0</a>	138
7.50.3.23	<a href="#">SetUniformArray(int location, const int *value, std::size_t count)=0</a>	138
7.50.3.24	<a href="#">SetUniformArray(int location, const Gs::Vector2i *value, std::size_t count)=0</a>	138
7.50.3.25	<a href="#">SetUniformArray(int location, const Gs::Vector3i *value, std::size_t count)=0</a>	138
7.50.3.26	<a href="#">SetUniformArray(int location, const Gs::Vector4i *value, std::size_t count)=0</a>	138
7.50.3.27	<a href="#">SetUniformArray(int location, const float *value, std::size_t count)=0</a>	138
7.50.3.28	<a href="#">SetUniformArray(int location, const Gs::Vector2f *value, std::size_t count)=0</a>	138

7.50.3.29	<a href="#">SetUniformArray(int location, const Gs::Vector3f *value, std::size_t count)=0</a>	138
7.50.3.30	<a href="#">SetUniformArray(int location, const Gs::Vector4f *value, std::size_t count)=0</a>	138
7.50.3.31	<a href="#">SetUniformArray(int location, const Gs::Matrix2f *value, std::size_t count)=0</a>	138
7.50.3.32	<a href="#">SetUniformArray(int location, const Gs::Matrix3f *value, std::size_t count)=0</a>	139
7.50.3.33	<a href="#">SetUniformArray(int location, const Gs::Matrix4f *value, std::size_t count)=0</a>	139
7.50.3.34	<a href="#">SetUniformArray(const std::string &amp;name, const int *value, std::size_t count)=0</a>	139
7.50.3.35	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector2i *value, std::size_t count)=0</a>	139
7.50.3.36	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector3i *value, std::size_t count)=0</a>	139
7.50.3.37	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector4i *value, std::size_t count)=0</a>	139
7.50.3.38	<a href="#">SetUniformArray(const std::string &amp;name, const float *value, std::size_t count)=0</a>	139
7.50.3.39	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector2f *value, std::size_t count)=0</a>	139
7.50.3.40	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector3f *value, std::size_t count)=0</a>	139
7.50.3.41	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Vector4f *value, std::size_t count)=0</a>	139
7.50.3.42	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Matrix2f *value, std::size_t count)=0</a>	139
7.50.3.43	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Matrix3f *value, std::size_t count)=0</a>	139
7.50.3.44	<a href="#">SetUniformArray(const std::string &amp;name, const Gs::Matrix4f *value, std::size_t count)=0</a>	139
7.51	<a href="#">LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor Struct Reference</a>	139
7.51.1	<a href="#">Member Data Documentation</a>	140
7.51.1.1	<a href="#">invertFrontFace</a>	140
7.51.1.2	<a href="#">screenSpaceOriginLowerLeft</a>	140
7.52	<a href="#">LLGL::StencilDescriptor Struct Reference</a>	140
7.52.1	<a href="#">Detailed Description</a>	141
7.52.2	<a href="#">Member Data Documentation</a>	141
7.52.2.1	<a href="#">back</a>	141
7.52.2.2	<a href="#">front</a>	141

7.52.2.3	testEnabled	141
7.53	LLGL::StencilFaceDescriptor Struct Reference	141
7.53.1	Detailed Description	142
7.53.2	Member Data Documentation	142
7.53.2.1	compareMask	142
7.53.2.2	compareOp	142
7.53.2.3	depthFailOp	142
7.53.2.4	depthPassOp	142
7.53.2.5	reference	142
7.53.2.6	stencilFailOp	142
7.53.2.7	writeMask	142
7.54	LLGL::StorageBuffer Class Reference	142
7.54.1	Detailed Description	143
7.54.2	Constructor & Destructor Documentation	143
7.54.2.1	~StorageBuffer()	143
7.55	LLGL::StorageBufferDescriptor Struct Reference	143
7.55.1	Detailed Description	143
7.55.2	Constructor & Destructor Documentation	144
7.55.2.1	StorageBufferDescriptor()=default	144
7.55.2.2	StorageBufferDescriptor(unsigned int size, BufferUsage usage)	144
7.55.3	Member Data Documentation	144
7.55.3.1	size	144
7.55.3.2	type	144
7.55.3.3	usage	144
7.56	LLGL::StorageBufferViewDescriptor Struct Reference	144
7.56.1	Detailed Description	145
7.56.2	Member Data Documentation	145
7.56.2.1	index	145
7.56.2.2	name	145
7.56.2.3	type	145

7.57 LLGL::SubTextureDescriptor Struct Reference . . . . .	145
7.57.1 Detailed Description . . . . .	146
7.57.2 Constructor & Destructor Documentation . . . . .	146
7.57.2.1 SubTextureDescriptor() . . . . .	146
7.57.2.2 ~SubTextureDescriptor() . . . . .	146
7.57.3 Member Data Documentation . . . . .	146
7.57.3.1 "@7 . . . . .	146
7.57.3.2 mipLevel . . . . .	146
7.57.3.3 texture1DDesc . . . . .	147
7.57.3.4 texture2DDesc . . . . .	147
7.57.3.5 texture3DDesc . . . . .	147
7.57.3.6 textureCubeDesc . . . . .	147
7.58 LLGL::Texture Class Reference . . . . .	147
7.58.1 Detailed Description . . . . .	147
7.58.2 Constructor & Destructor Documentation . . . . .	148
7.58.2.1 ~Texture() . . . . .	148
7.58.3 Member Function Documentation . . . . .	148
7.58.3.1 GetType() const . . . . .	148
7.58.3.2 QueryMipLevelSize(int mipLevel) const =0 . . . . .	148
7.58.3.3 SetType(const TextureType type) . . . . .	148
7.59 LLGL::SubTextureDescriptor::Texture1DDescriptor Struct Reference . . . . .	148
7.59.1 Member Data Documentation . . . . .	149
7.59.1.1 layerOffset . . . . .	149
7.59.1.2 layers . . . . .	149
7.59.1.3 width . . . . .	149
7.59.1.4 x . . . . .	149
7.60 LLGL::TextureDescriptor::Texture1DDescriptor Struct Reference . . . . .	149
7.60.1 Member Data Documentation . . . . .	150
7.60.1.1 layers . . . . .	150
7.60.1.2 width . . . . .	150

7.61	<a href="#">LLGL::TextureDescriptor::Texture2DDescriptor Struct Reference</a>	150
7.61.1	<a href="#">Member Data Documentation</a>	150
7.61.1.1	<a href="#">height</a>	150
7.61.1.2	<a href="#">layers</a>	150
7.61.1.3	<a href="#">width</a>	150
7.62	<a href="#">LLGL::SubTextureDescriptor::Texture2DDescriptor Struct Reference</a>	151
7.62.1	<a href="#">Member Data Documentation</a>	151
7.62.1.1	<a href="#">height</a>	151
7.62.1.2	<a href="#">layerOffset</a>	151
7.62.1.3	<a href="#">layers</a>	151
7.62.1.4	<a href="#">width</a>	151
7.62.1.5	<a href="#">x</a>	151
7.62.1.6	<a href="#">y</a>	152
7.63	<a href="#">LLGL::TextureDescriptor::Texture3DDescriptor Struct Reference</a>	152
7.63.1	<a href="#">Member Data Documentation</a>	152
7.63.1.1	<a href="#">depth</a>	152
7.63.1.2	<a href="#">height</a>	152
7.63.1.3	<a href="#">width</a>	152
7.64	<a href="#">LLGL::SubTextureDescriptor::Texture3DDescriptor Struct Reference</a>	152
7.64.1	<a href="#">Member Data Documentation</a>	153
7.64.1.1	<a href="#">depth</a>	153
7.64.1.2	<a href="#">height</a>	153
7.64.1.3	<a href="#">width</a>	153
7.64.1.4	<a href="#">x</a>	153
7.64.1.5	<a href="#">y</a>	153
7.64.1.6	<a href="#">z</a>	153
7.65	<a href="#">LLGL::SubTextureDescriptor::TextureCubeDescriptor Struct Reference</a>	154
7.65.1	<a href="#">Member Data Documentation</a>	154
7.65.1.1	<a href="#">cubeFaceOffset</a>	154
7.65.1.2	<a href="#">cubeFaces</a>	154

7.65.1.3	height	154
7.65.1.4	layerOffset	154
7.65.1.5	width	154
7.65.1.6	x	155
7.65.1.7	y	155
7.66	LLGL::TextureDescriptor::TextureCubeDescriptor Struct Reference	155
7.66.1	Member Data Documentation	155
7.66.1.1	height	155
7.66.1.2	layers	155
7.66.1.3	width	155
7.67	LLGL::TextureDescriptor Struct Reference	156
7.67.1	Detailed Description	156
7.67.2	Constructor & Destructor Documentation	157
7.67.2.1	TextureDescriptor()	157
7.67.2.2	~TextureDescriptor()	157
7.67.3	Member Data Documentation	157
7.67.3.1	"@5	157
7.67.3.2	format	157
7.67.3.3	texture1DDesc	157
7.67.3.4	texture2DDesc	157
7.67.3.5	texture3DDesc	157
7.67.3.6	textureCubeDesc	157
7.67.3.7	type	157
7.68	LLGL::Timer Class Reference	157
7.68.1	Member Typedef Documentation	158
7.68.1.1	FrameCount	158
7.68.2	Constructor & Destructor Documentation	158
7.68.2.1	~Timer()	158
7.68.3	Member Function Documentation	158
7.68.3.1	Create()	158



7.68.3.2	<a href="#">GetDeltaTime() const</a>	159
7.68.3.3	<a href="#">GetFrameCount() const</a>	159
7.68.3.4	<a href="#">GetFrequency() const =0</a>	159
7.68.3.5	<a href="#">MeasureTime()</a>	159
7.68.3.6	<a href="#">ResetFrameCounter()</a>	159
7.68.3.7	<a href="#">Start()=0</a>	160
7.68.3.8	<a href="#">Stop()=0</a>	160
7.69	<a href="#">LLGL::UniformDescriptor Struct Reference</a>	160
7.69.1	<a href="#">Detailed Description</a>	160
7.69.2	<a href="#">Member Data Documentation</a>	160
7.69.2.1	<a href="#">location</a>	160
7.69.2.2	<a href="#">name</a>	160
7.69.2.3	<a href="#">size</a>	160
7.69.2.4	<a href="#">type</a>	160
7.70	<a href="#">LLGL::VertexAttribute Struct Reference</a>	161
7.70.1	<a href="#">Detailed Description</a>	161
7.70.2	<a href="#">Member Data Documentation</a>	161
7.70.2.1	<a href="#">components</a>	161
7.70.2.2	<a href="#">conversion</a>	161
7.70.2.3	<a href="#">dataType</a>	161
7.70.2.4	<a href="#">name</a>	161
7.70.2.5	<a href="#">offset</a>	162
7.70.2.6	<a href="#">perInstance</a>	162
7.70.2.7	<a href="#">semanticIndex</a>	162
7.71	<a href="#">LLGL::VertexBuffer Class Reference</a>	162
7.71.1	<a href="#">Detailed Description</a>	162
7.71.2	<a href="#">Constructor &amp; Destructor Documentation</a>	162
7.71.2.1	<a href="#">~VertexBuffer()</a>	162
7.71.3	<a href="#">Member Function Documentation</a>	162
7.71.3.1	<a href="#">GetVertexFormat() const</a>	162

7.71.3.2	<a href="#">SetVertexFormat(const VertexFormat &amp;vertexFormat)</a>	162
7.72	<a href="#">LLGL::VertexBufferDescriptor Struct Reference</a>	163
7.72.1	<a href="#">Detailed Description</a>	163
7.72.2	<a href="#">Constructor &amp; Destructor Documentation</a>	163
7.72.2.1	<a href="#">VertexBufferDescriptor()=default</a>	163
7.72.2.2	<a href="#">VertexBufferDescriptor(unsigned int size, BufferUsage usage, const VertexFormat &amp;vertexFormat)</a>	163
7.72.3	<a href="#">Member Data Documentation</a>	163
7.72.3.1	<a href="#">size</a>	163
7.72.3.2	<a href="#">usage</a>	163
7.72.3.3	<a href="#">vertexFormat</a>	164
7.73	<a href="#">LLGL::VertexFormat Class Reference</a>	164
7.73.1	<a href="#">Detailed Description</a>	164
7.73.2	<a href="#">Member Function Documentation</a>	164
7.73.2.1	<a href="#">AddAttribute(const std::string &amp;name, const DataType dataType, unsigned int components, bool conversion=false, bool perInstance=false)</a>	164
7.73.2.2	<a href="#">AddAttribute(const std::string &amp;semanticName, unsigned int semanticIndex, const DataType dataType, unsigned int components, bool conversion=false, bool perInstance=false)</a>	165
7.73.2.3	<a href="#">GetAttributes() const</a>	165
7.73.2.4	<a href="#">GetFormatSize() const</a>	166
7.74	<a href="#">LLGL::VideoAdapterDescriptor Struct Reference</a>	166
7.74.1	<a href="#">Detailed Description</a>	166
7.74.2	<a href="#">Member Data Documentation</a>	166
7.74.2.1	<a href="#">name</a>	166
7.74.2.2	<a href="#">outputs</a>	166
7.74.2.3	<a href="#">vendor</a>	167
7.74.2.4	<a href="#">videoMemory</a>	167
7.75	<a href="#">LLGL::VideoDisplayMode Struct Reference</a>	167
7.75.1	<a href="#">Member Data Documentation</a>	167
7.75.1.1	<a href="#">height</a>	167
7.75.1.2	<a href="#">refreshRate</a>	167

7.75.1.3	width	167
7.76	LLGL::VideoModeDescriptor Struct Reference	167
7.76.1	Member Data Documentation	168
7.76.1.1	colorDepth	168
7.76.1.2	fullscreen	168
7.76.1.3	resolution	168
7.76.1.4	swapChainMode	168
7.77	LLGL::VideoOutput Struct Reference	168
7.77.1	Member Data Documentation	169
7.77.1.1	displayModes	169
7.78	LLGL::Viewport Struct Reference	169
7.78.1	Detailed Description	169
7.78.2	Constructor & Destructor Documentation	170
7.78.2.1	Viewport()=default	170
7.78.2.2	Viewport(const Viewport &)=default	170
7.78.2.3	Viewport(float x, float y, float width, float height)	170
7.78.2.4	Viewport(float x, float y, float width, float height, float minDepth, float maxDepth)	170
7.78.3	Member Data Documentation	170
7.78.3.1	height	170
7.78.3.2	maxDepth	170
7.78.3.3	minDepth	170
7.78.3.4	width	170
7.78.3.5	x	170
7.78.3.6	y	170
7.79	LLGL::VsyncDescriptor Struct Reference	170
7.79.1	Member Data Documentation	171
7.79.1.1	enabled	171
7.79.1.2	interval	171
7.79.1.3	refreshRate	171
7.80	LLGL::Window Class Reference	171

7.80.1	Constructor & Destructor Documentation . . . . .	173
7.80.1.1	~Window() . . . . .	173
7.80.2	Member Function Documentation . . . . .	173
7.80.2.1	AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . . .	173
7.80.2.2	Create(const WindowDescriptor &desc) . . . . .	173
7.80.2.3	GetNativeHandle(void *nativeHandle) const =0 . . . . .	173
7.80.2.4	GetPosition() const =0 . . . . .	173
7.80.2.5	GetSize(bool useClientArea=true) const =0 . . . . .	173
7.80.2.6	GetTitle() const =0 . . . . .	173
7.80.2.7	IsShown() const =0 . . . . .	173
7.80.2.8	PostChar(wchar_t chr) . . . . .	173
7.80.2.9	PostDoubleClick(Key keyCode) . . . . .	173
7.80.2.10	PostGlobalMotion(const Point &motion) . . . . .	173
7.80.2.11	PostKeyDown(Key keyCode) . . . . .	173
7.80.2.12	PostKeyUp(Key keyCode) . . . . .	173
7.80.2.13	PostLocalMotion(const Point &position) . . . . .	173
7.80.2.14	PostQuit() . . . . .	173
7.80.2.15	PostResize(const Size &clientAreaSize) . . . . .	174
7.80.2.16	PostWheelMotion(int motion) . . . . .	174
7.80.2.17	ProcessEvents() . . . . .	174
7.80.2.18	ProcessSystemEvents()=0 . . . . .	174
7.80.2.19	QueryDesc() const =0 . . . . .	174
7.80.2.20	Recreate(const WindowDescriptor &desc)=0 . . . . .	174
7.80.2.21	RemoveEventListener(const EventListener *eventListener) . . . . .	174
7.80.2.22	SetDesc(const WindowDescriptor &desc)=0 . . . . .	174
7.80.2.23	SetPosition(const Point &position)=0 . . . . .	174
7.80.2.24	SetSize(const Size &size, bool useClientArea=true)=0 . . . . .	174
7.80.2.25	SetTitle(const std::wstring &title)=0 . . . . .	174
7.80.2.26	Show(bool show=true)=0 . . . . .	174
7.81	LLGL::WindowDescriptor Struct Reference . . . . .	175
7.81.1	Detailed Description . . . . .	175
7.81.2	Member Data Documentation . . . . .	175
7.81.2.1	acceptDropFiles . . . . .	175
7.81.2.2	borderless . . . . .	175
7.81.2.3	centered . . . . .	175
7.81.2.4	position . . . . .	175
7.81.2.5	preventForPowerSafe . . . . .	175
7.81.2.6	resizable . . . . .	175
7.81.2.7	size . . . . .	175
7.81.2.8	title . . . . .	176
7.81.2.9	visible . . . . .	176
7.81.2.10	windowContext . . . . .	176

<b>8 File Documentation</b>	<b>177</b>
8.1 Color.h File Reference	177
8.2 ColorRGB.h File Reference	178
8.3 ColorRGBA.h File Reference	178
8.4 ComputePipeline.h File Reference	179
8.5 ConstantBuffer.h File Reference	179
8.6 Desktop.h File Reference	180
8.7 Export.h File Reference	180
8.7.1 Macro Definition Documentation	180
8.7.1.1 LLGL_EXPORT	180
8.8 GraphicsPipeline.h File Reference	180
8.9 GraphicsPipelineFlags.h File Reference	181
8.10 Image.h File Reference	183
8.11 IndexBuffer.h File Reference	184
8.12 IndexFormat.h File Reference	184
8.13 Input.h File Reference	184
8.14 Key.h File Reference	185
8.15 LinuxNativeHandle.h File Reference	186
8.16 LLGL.h File Reference	186
8.17 Log.h File Reference	186
8.18 MacOSNativeHandle.h File Reference	187
8.19 NativeHandle.h File Reference	187
8.20 Query.h File Reference	187
8.21 RenderContext.h File Reference	188
8.22 RenderContextDescriptor.h File Reference	188
8.23 RenderContextFlags.h File Reference	189
8.24 RenderingDebugger.h File Reference	190
8.25 RenderingProfiler.h File Reference	190
8.26 RenderSystem.h File Reference	191
8.27 RenderSystemFlags.h File Reference	191

8.28	RenderTarget.h File Reference . . . . .	192
8.29	Sampler.h File Reference . . . . .	193
8.30	SamplerFlags.h File Reference . . . . .	193
8.31	Shader.h File Reference . . . . .	194
8.32	ShaderProgram.h File Reference . . . . .	194
8.33	ShaderUniform.h File Reference . . . . .	195
8.34	StorageBuffer.h File Reference . . . . .	196
8.35	Texture.h File Reference . . . . .	196
8.36	TextureFlags.h File Reference . . . . .	197
8.37	Timer.h File Reference . . . . .	198
8.38	Types.h File Reference . . . . .	198
8.39	VertexAttribute.h File Reference . . . . .	199
8.40	VertexBuffer.h File Reference . . . . .	199
8.41	VertexFormat.h File Reference . . . . .	200
8.42	VideoAdapter.h File Reference . . . . .	200
8.43	Win32NativeHandle.h File Reference . . . . .	200
8.44	Window.h File Reference . . . . .	201
	<b>Index</b>	<b>203</b>

# Chapter 1

## LLGL 1.00 Alpha Documentation

### LLGL (Low Level Graphics Library)

#### Overview

- **Version:** 1.00 Alpha
- **License:** 3-Clause BSD License

#### Progress

- **OpenGL Renderer:** ~70% done
- **Direct3D 12 Renderer:** ~5% done
- **Direct3D 11 Renderer:** not started yet
- **Vulkan Renderer:** not started yet

#### Getting Started

```
#include <LLGL/LLGL.h>

int main()
{
    // Create a window to render into
    LLGL::WindowDescriptor windowDesc;

    windowDesc.title    = L"LLGL Example";
    windowDesc.visible  = true;
    windowDesc.centered = true;
    windowDesc.width    = 640;
    windowDesc.height   = 480;

    auto window = LLGL::Window::Create(windowDesc);

    // Add keyboard/mouse event listener
    auto input = std::make_shared<LLGL::Input>();
    window->AddEventListener(input);

    //TO BE CONTINUED ...

    // Main loop
    while (window->ProcessEvents() && !input->KeyPressed(LLGL::Key::Escape))
    {
        // Draw with OpenGL, or Direct3D, or Vulkan, or whatever ...
    }

    return 0;
}
```

## Thin Abstraction Layer

```
// Interface:
RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex);

// OpenGL Implementation:
void GLRenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    glDrawElements(
        renderState_.drawMode,
        static_cast<GLsizei>(numVertices),
        renderState_.indexBufferDataType,
        (reinterpret_cast<const GLvoid*>(firstIndex * renderState_.indexBufferStride))
    );
}

// Direct3D 11 Implementation
void D3D11RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    context_->DrawIndexed(numVertices, 0, firstIndex);
}

// Direct3D 12 Implementation
void D3D12RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    commandList_->DrawIndexedInstanced(numVertices, 1, firstIndex, 0, 0);
}

// Vulkan Implementation
void VKRenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    vkCmdDrawIndexed(commandBuffer_, numVertices, 1, firstIndex, 0, 0);
}
```



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">LLGL</a> . . . . .	<a href="#">13</a>
<a href="#">LLGL::Desktop</a> . . . . .	<a href="#">41</a>
<a href="#">LLGL::Log</a> . . . . .	<a href="#">42</a>



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

LLGL::AntiAliasingDescriptor . . . . .	45
LLGL::BlendDescriptor . . . . .	45
LLGL::BlendTargetDescriptor . . . . .	46
LLGL::ClearBuffersFlags . . . . .	48
LLGL::Color< T, N > . . . . .	48
LLGL::Color< bool > . . . . .	48
LLGL::Color< float > . . . . .	48
LLGL::Color< T, 3u > . . . . .	51
LLGL::Color< T, 4u > . . . . .	54
LLGL::Color< unsigned char > . . . . .	48
LLGL::ComputePipeline . . . . .	57
LLGL::ComputePipelineDescriptor . . . . .	58
LLGL::ConstantBuffer . . . . .	59
LLGL::ConstantBufferDescriptor . . . . .	59
LLGL::ConstantBufferViewDescriptor . . . . .	60
LLGL::RenderingProfiler::Counter . . . . .	61
LLGL::DepthDescriptor . . . . .	62
LLGL::Window::EventListener . . . . .	62
LLGL::Input . . . . .	73
LLGL::ShaderSource::GLSL . . . . .	64
LLGL::GraphicsAPIDependentStateDescriptor . . . . .	64
LLGL::GraphicsPipeline . . . . .	65
LLGL::GraphicsPipelineDescriptor . . . . .	66
LLGL::ShaderSource::HLSL . . . . .	67
LLGL::ImageDescriptor . . . . .	68
LLGL::IndexBuffer . . . . .	70
LLGL::IndexBufferDescriptor . . . . .	71
LLGL::IndexFormat . . . . .	72
LLGL::RenderingDebugger::Message . . . . .	74
LLGL::NativeContextHandle . . . . .	76
LLGL::NativeHandle . . . . .	77
LLGL::ProfileOpenGLDescriptor . . . . .	77
LLGL::Query . . . . .	78
LLGL::QueryDescriptor . . . . .	79
LLGL::RasterizerDescriptor . . . . .	80

LLGL::RenderContext . . . . .	82
LLGL::RenderContextDescriptor . . . . .	96
LLGL::RenderingCaps . . . . .	97
LLGL::RenderingDebugger . . . . .	102
LLGL::RenderingProfiler . . . . .	103
LLGL::RenderSystem . . . . .	106
LLGL::RenderSystemConfiguration . . . . .	117
LLGL::RenderTarget . . . . .	118
LLGL::Sampler . . . . .	121
LLGL::SamplerDescriptor . . . . .	121
LLGL::Scissor . . . . .	124
LLGL::Shader . . . . .	125
LLGL::ShaderCompileFlags . . . . .	127
LLGL::ShaderDisassembleFlags . . . . .	128
LLGL::ShaderProgram . . . . .	128
LLGL::ShaderSource . . . . .	133
LLGL::ShaderStageFlags . . . . .	135
LLGL::ShaderUniform . . . . .	136
LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor . . . . .	139
LLGL::StencilDescriptor . . . . .	140
LLGL::StencilFaceDescriptor . . . . .	141
LLGL::StorageBuffer . . . . .	142
LLGL::StorageBufferDescriptor . . . . .	143
LLGL::StorageBufferViewDescriptor . . . . .	144
LLGL::SubTextureDescriptor . . . . .	145
LLGL::Texture . . . . .	147
LLGL::SubTextureDescriptor::Texture1DDescriptor . . . . .	148
LLGL::TextureDescriptor::Texture1DDescriptor . . . . .	149
LLGL::TextureDescriptor::Texture2DDescriptor . . . . .	150
LLGL::SubTextureDescriptor::Texture2DDescriptor . . . . .	151
LLGL::TextureDescriptor::Texture3DDescriptor . . . . .	152
LLGL::SubTextureDescriptor::Texture3DDescriptor . . . . .	152
LLGL::SubTextureDescriptor::TextureCubeDescriptor . . . . .	154
LLGL::TextureDescriptor::TextureCubeDescriptor . . . . .	155
LLGL::TextureDescriptor . . . . .	156
LLGL::Timer . . . . .	157
LLGL::UniformDescriptor . . . . .	160
LLGL::VertexAttribute . . . . .	161
LLGL::VertexBuffer . . . . .	162
LLGL::VertexBufferDescriptor . . . . .	163
LLGL::VertexFormat . . . . .	164
LLGL::VideoAdapterDescriptor . . . . .	166
LLGL::VideoDisplayMode . . . . .	167
LLGL::VideoModeDescriptor . . . . .	167
LLGL::VideoOutput . . . . .	168
LLGL::Viewport . . . . .	169
LLGL::VsyncDescriptor . . . . .	170
LLGL::Window . . . . .	171
LLGL::WindowDescriptor . . . . .	175

## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">LLGL::AntiAliasingDescriptor</a>	45
<a href="#">LLGL::BlendDescriptor</a>	
Blending state descriptor structure	45
<a href="#">LLGL::BlendTargetDescriptor</a>	
Blend target state descriptor structure	46
<a href="#">LLGL::ClearBuffersFlags</a>	
Render context clear buffer flags	48
<a href="#">LLGL::Color&lt; T, N &gt;</a>	
Base color class with N components	48
<a href="#">LLGL::Color&lt; T, 3u &gt;</a>	
RGB color class with components: r, g, and b	51
<a href="#">LLGL::Color&lt; T, 4u &gt;</a>	
RGBA color class with components: r, g, b, and a	54
<a href="#">LLGL::ComputePipeline</a>	
Compute pipeline interface	57
<a href="#">LLGL::ComputePipelineDescriptor</a>	
Compute pipeline descriptor structure	58
<a href="#">LLGL::ConstantBuffer</a>	
Constant buffer (also "Uniform Buffer Object") interface	59
<a href="#">LLGL::ConstantBufferDescriptor</a>	
Constant buffer descriptor structure	59
<a href="#">LLGL::ConstantBufferViewDescriptor</a>	
Constant buffer shader-view descriptor structure	60
<a href="#">LLGL::RenderingProfiler::Counter</a>	61
<a href="#">LLGL::DepthDescriptor</a>	
Depth state descriptor structure	62
<a href="#">LLGL::Window::EventListener</a>	62
<a href="#">LLGL::ShaderSource::GLSL</a>	
Shader source descriptor for <a href="#">GLSL</a>	64
<a href="#">LLGL::GraphicsAPIDependentStateDescriptor</a>	
Low-level graphics API dependent state descriptor union	64
<a href="#">LLGL::GraphicsPipeline</a>	
Graphics pipeline interface	65
<a href="#">LLGL::GraphicsPipelineDescriptor</a>	
Graphics pipeline descriptor structure	66

LLGL::ShaderSource::HLSL	
Shader source descriptor for HLSL	67
LLGL::ImageDescriptor	
Image descriptor structure	68
LLGL::IndexBuffer	
Index buffer interface	70
LLGL::IndexBufferDescriptor	
Index buffer descriptor structure	71
LLGL::IndexFormat	72
LLGL::Input	73
LLGL::RenderingDebugger::Message	
Rendering debugger message class	74
LLGL::NativeContextHandle	
Linux native context handle structure	76
LLGL::NativeHandle	
Linux native handle structure	77
LLGL::ProfileOpenGLDescriptor	77
LLGL::Query	
Query interface	78
LLGL::QueryDescriptor	
Query descriptor structure	79
LLGL::RasterizerDescriptor	
Rasterizer state descriptor structure	80
LLGL::RenderContext	
Render context interface	82
LLGL::RenderContextDescriptor	96
LLGL::RenderingCaps	
Rendering capabilities structure	97
LLGL::RenderingDebugger	
Rendering debugger interface	102
LLGL::RenderingProfiler	
Rendering profiler model class	103
LLGL::RenderSystem	
Render system interface	106
LLGL::RenderSystemConfiguration	
Render system configuration structure	117
LLGL::RenderTarget	
Render target interface	118
LLGL::Sampler	
Sampler interface	121
LLGL::SamplerDescriptor	
Texture sampler descriptor structure	121
LLGL::Scissor	
Scissor dimensions	124
LLGL::Shader	
Shader interface	125
LLGL::ShaderCompileFlags	
Shader compilation flags enumeration	127
LLGL::ShaderDisassembleFlags	
Shader disassemble flags enumeration	128
LLGL::ShaderProgram	
Shader program interface	128
LLGL::ShaderSource	
Shader source code union	133
LLGL::ShaderStageFlags	
Shader stage flags	135
LLGL::ShaderUniform	
Shader uniform setter interface	136

LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor	139
LLGL::StencilDescriptor	
Stencil state descriptor structure	140
LLGL::StencilFaceDescriptor	
Stencil face descriptor structure	141
LLGL::StorageBuffer	
Storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer") interface	142
LLGL::StorageBufferDescriptor	
Storage buffer descriptor structure	143
LLGL::StorageBufferViewDescriptor	
Storage buffer shader-view descriptor structure	144
LLGL::SubTextureDescriptor	
Sub-texture descriptor structure	145
LLGL::Texture	
Texture interface	147
LLGL::SubTextureDescriptor::Texture1DDescriptor	148
LLGL::TextureDescriptor::Texture1DDescriptor	149
LLGL::TextureDescriptor::Texture2DDescriptor	150
LLGL::SubTextureDescriptor::Texture2DDescriptor	151
LLGL::TextureDescriptor::Texture3DDescriptor	152
LLGL::SubTextureDescriptor::Texture3DDescriptor	152
LLGL::SubTextureDescriptor::TextureCubeDescriptor	154
LLGL::TextureDescriptor::TextureCubeDescriptor	155
LLGL::TextureDescriptor	
Texture descriptor structure	156
LLGL::Timer	157
LLGL::UniformDescriptor	
Shader uniform descriptor structure	160
LLGL::VertexAttribute	
Vertex attribute class	161
LLGL::VertexBuffer	
Vertex buffer interface	162
LLGL::VertexBufferDescriptor	
Vertex buffer descriptor structure	163
LLGL::VertexFormat	
Vertex format descriptor class	164
LLGL::VideoAdapterDescriptor	
Video adapter descriptor structure	166
LLGL::VideoDisplayMode	167
LLGL::VideoModeDescriptor	167
LLGL::VideoOutput	168
LLGL::Viewport	
Viewport dimensions	169
LLGL::VsyncDescriptor	170
LLGL::Window	171
LLGL::WindowDescriptor	
Window descriptor structure	175





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

Color.h	177
ColorRGB.h	178
ColorRGBA.h	178
ComputePipeline.h	179
ConstantBuffer.h	179
Desktop.h	180
Export.h	180
GraphicsPipeline.h	180
GraphicsPipelineFlags.h	181
Image.h	183
IndexBuffer.h	184
IndexFormat.h	184
Input.h	184
Key.h	185
LinuxNativeHandle.h	186
LLGL.h	186
Log.h	186
MacOSNativeHandle.h	187
NativeHandle.h	187
Query.h	187
RenderContext.h	188
RenderContextDescriptor.h	188
RenderContextFlags.h	189
RenderingDebugger.h	190
RenderingProfiler.h	190
RenderSystem.h	191
RenderSystemFlags.h	191
RenderTarget.h	192
Sampler.h	193
SamplerFlags.h	193
Shader.h	194
ShaderProgram.h	194
ShaderUniform.h	195
StorageBuffer.h	196
Texture.h	196

TextureFlags.h	197
Timer.h	198
Types.h	198
VertexAttribute.h	199
VertexBuffer.h	199
VertexFormat.h	200
VideoAdapter.h	200
Win32NativeHandle.h	200
Window.h	201

## Chapter 6

# Namespace Documentation

### 6.1 LLGL Namespace Reference

#### Namespaces

- [Desktop](#)
- [Log](#)

#### Classes

- struct [AntiAliasingDescriptor](#)
- struct [BlendDescriptor](#)  
*Blending state descriptor structure.*
- struct [BlendTargetDescriptor](#)  
*Blend target state descriptor structure.*
- struct [ClearBuffersFlags](#)  
*Render context clear buffer flags.*
- class [Color](#)  
*Base color class with N components.*
- class [Color< T, 3u >](#)  
*RGB color class with components: r, g, and b.*
- class [Color< T, 4u >](#)  
*RGBA color class with components: r, g, b, and a.*
- class [ComputePipeline](#)  
*Compute pipeline interface.*
- struct [ComputePipelineDescriptor](#)  
*Compute pipeline descriptor structure.*
- class [ConstantBuffer](#)  
*Constant buffer (also "Uniform Buffer Object") interface.*
- struct [ConstantBufferDescriptor](#)  
*Constant buffer descriptor structure.*
- struct [ConstantBufferViewDescriptor](#)  
*Constant buffer shader-view descriptor structure.*
- struct [DepthDescriptor](#)  
*Depth state descriptor structure.*

- union [GraphicsAPIDependentStateDescriptor](#)  
*Low-level graphics API dependent state descriptor union.*
- class [GraphicsPipeline](#)  
*Graphics pipeline interface.*
- struct [GraphicsPipelineDescriptor](#)  
*Graphics pipeline descriptor structure.*
- struct [ImageDescriptor](#)  
*Image descriptor structure.*
- class [IndexBuffer](#)  
*Index buffer interface.*
- struct [IndexBufferDescriptor](#)  
*Index buffer descriptor structure.*
- class [IndexFormat](#)
- class [Input](#)
- struct [NativeContextHandle](#)  
*Linux native context handle structure.*
- struct [NativeHandle](#)  
*Linux native handle structure.*
- struct [ProfileOpenGLDescriptor](#)
- class [Query](#)  
*Query interface.*
- struct [QueryDescriptor](#)  
*Query descriptor structure.*
- struct [RasterizerDescriptor](#)  
*Rasterizer state descriptor structure.*
- class [RenderContext](#)  
*Render context interface.*
- struct [RenderContextDescriptor](#)
- struct [RenderingCaps](#)  
*Rendering capabilities structure.*
- class [RenderingDebugger](#)  
*Rendering debugger interface.*
- class [RenderingProfiler](#)  
*Rendering profiler model class.*
- class [RenderSystem](#)  
*Render system interface.*
- struct [RenderSystemConfiguration](#)  
*Render system configuration structure.*
- class [RenderTarget](#)  
*Render target interface.*
- class [Sampler](#)  
*Sampler interface.*
- struct [SamplerDescriptor](#)  
*Texture sampler descriptor structure.*
- struct [Scissor](#)  
*Scissor dimensions.*
- class [Shader](#)  
*Shader interface.*
- struct [ShaderCompileFlags](#)  
*Shader compilation flags enumeration.*
- struct [ShaderDisassembleFlags](#)

- *Shader* disassemble flags enumeration.
- class [ShaderProgram](#)  
*Shader* program interface.
- union [ShaderSource](#)  
*Shader* source code union.
- struct [ShaderStageFlags](#)  
*Shader* stage flags.
- class [ShaderUniform](#)  
*Shader* uniform setter interface.
- struct [StencilDescriptor](#)  
*Stencil* state descriptor structure.
- struct [StencilFaceDescriptor](#)  
*Stencil* face descriptor structure.
- class [StorageBuffer](#)  
*Storage* buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer") interface.
- struct [StorageBufferDescriptor](#)  
*Storage* buffer descriptor structure.
- struct [StorageBufferViewDescriptor](#)  
*Storage* buffer shader-view descriptor structure.
- struct [SubTextureDescriptor](#)  
*Sub-texture* descriptor structure.
- class [Texture](#)  
*Texture* interface.
- struct [TextureDescriptor](#)  
*Texture* descriptor structure.
- class [Timer](#)
- struct [UniformDescriptor](#)  
*Shader* uniform descriptor structure.
- struct [VertexAttribute](#)  
*Vertex* attribute class.
- class [VertexBuffer](#)  
*Vertex* buffer interface.
- struct [VertexBufferDescriptor](#)  
*Vertex* buffer descriptor structure.
- class [VertexFormat](#)  
*Vertex* format descriptor class.
- struct [VideoAdapterDescriptor](#)  
*Video* adapter descriptor structure.
- struct [VideoDisplayMode](#)
- struct [VideoModeDescriptor](#)
- struct [VideoOutput](#)
- struct [Viewport](#)  
*Viewport* dimensions.
- struct [VsyncDescriptor](#)
- class [Window](#)
- struct [WindowDescriptor](#)  
*Window* descriptor structure.

## Typedefs

- `template<typename T >`  
`using ColorRGBT = Color< T, 3 >`
- `using ColorRGB = ColorRGBT< Gs::Real >`
- `using ColorRGBb = ColorRGBT< bool >`
- `using ColorRGBf = ColorRGBT< float >`
- `using ColorRGBd = ColorRGBT< double >`
- `using ColorRGBub = ColorRGBT< unsigned char >`
- `template<typename T >`  
`using ColorRGBAT = Color< T, 4 >`
- `using ColorRGBA = ColorRGBAT< Gs::Real >`
- `using ColorRGBAb = ColorRGBAT< bool >`
- `using ColorRGBAf = ColorRGBAT< float >`
- `using ColorRGBAd = ColorRGBAT< double >`
- `using ColorRGBAub = ColorRGBAT< unsigned char >`
- `using ByteBuffer = std::unique_ptr< char[ ]>`  
*Common byte buffer type.*
- `using DebugCallback = std::function< void(const std::string &type, const std::string &message)>`  
*Debug callback function interface.*
- `using Point = Gs::Vector2i`  
*2D point (integer)*
- `using Size = Gs::Vector2i`  
*2D size (integer)*

## Enumerations

- `enum PrimitiveTopology {`  
`PrimitiveTopology::PointList, PrimitiveTopology::LineList, PrimitiveTopology::LineStrip, PrimitiveTopology::↵`  
`LineLoop,`  
`PrimitiveTopology::LineListAdjacency, PrimitiveTopology::LineStripAdjacency, PrimitiveTopology::Triangle↵`  
`List, PrimitiveTopology::TriangleStrip,`  
`PrimitiveTopology::TriangleFan, PrimitiveTopology::TriangleListAdjacency, PrimitiveTopology::TriangleStrip↵`  
`Adjacency, PrimitiveTopology::Patches1,`  
`PrimitiveTopology::Patches2, PrimitiveTopology::Patches3, PrimitiveTopology::Patches4, Primitive↵`  
`Topology::Patches5,`  
`PrimitiveTopology::Patches6, PrimitiveTopology::Patches7, PrimitiveTopology::Patches8, Primitive↵`  
`Topology::Patches9,`  
`PrimitiveTopology::Patches10, PrimitiveTopology::Patches11, PrimitiveTopology::Patches12, Primitive↵`  
`Topology::Patches13,`  
`PrimitiveTopology::Patches14, PrimitiveTopology::Patches15, PrimitiveTopology::Patches16, Primitive↵`  
`Topology::Patches17,`  
`PrimitiveTopology::Patches18, PrimitiveTopology::Patches19, PrimitiveTopology::Patches20, Primitive↵`  
`Topology::Patches21,`  
`PrimitiveTopology::Patches22, PrimitiveTopology::Patches23, PrimitiveTopology::Patches24, Primitive↵`  
`Topology::Patches25,`  
`PrimitiveTopology::Patches26, PrimitiveTopology::Patches27, PrimitiveTopology::Patches28, Primitive↵`  
`Topology::Patches29,`  
`PrimitiveTopology::Patches30, PrimitiveTopology::Patches31, PrimitiveTopology::Patches32 }`  
*Primitive topology enumeration.*
- `enum CompareOp {`  
`CompareOp::Never, CompareOp::Less, CompareOp::Equal, CompareOp::LessEqual,`  
`CompareOp::Greater, CompareOp::NotEqual, CompareOp::GreaterEqual, CompareOp::Ever }`  
*Compare operations enumeration.*

- enum [StencilOp](#) {  
    [StencilOp::Keep](#), [StencilOp::Zero](#), [StencilOp::Replace](#), [StencilOp::IncClamp](#),  
    [StencilOp::DecClamp](#), [StencilOp::Invert](#), [StencilOp::IncWrap](#), [StencilOp::DecWrap](#) }

*Stencil operations enumeration.*

- enum [BlendOp](#) {  
    [BlendOp::Zero](#), [BlendOp::One](#), [BlendOp::SrcColor](#), [BlendOp::InvSrcColor](#),  
    [BlendOp::SrcAlpha](#), [BlendOp::InvSrcAlpha](#), [BlendOp::DestColor](#), [BlendOp::InvDestColor](#),  
    [BlendOp::DestAlpha](#), [BlendOp::InvDestAlpha](#) }

*Blending operations enumeration.*

- enum [BlendArithmetic](#) {  
    [BlendArithmetic::Add](#), [BlendArithmetic::Subtract](#), [BlendArithmetic::RevSubtract](#), [BlendArithmetic::Min](#),  
    [BlendArithmetic::Max](#) }

*Blending arithmetic operations enumeration.*

- enum [PolygonMode](#) { [PolygonMode::Fill](#), [PolygonMode::Wireframe](#), [PolygonMode::Points](#) }

*Polygon filling modes enumeration.*

- enum [CullMode](#) { [CullMode::Disabled](#), [CullMode::Front](#), [CullMode::Back](#) }

*Polygon culling modes enumeration.*

- enum [DataType](#) {  
    [DataType::Int8](#), [DataType::UInt8](#), [DataType::Int16](#), [DataType::UInt16](#),  
    [DataType::Int32](#), [DataType::UInt32](#), [DataType::Float](#), [DataType::Double](#) }

*Renderer data types enumeration.*

- enum [ImageFormat](#) {  
    [ImageFormat::R](#), [ImageFormat::RG](#), [ImageFormat::RGB](#), [ImageFormat::BGR](#),  
    [ImageFormat::RGBA](#), [ImageFormat::BGRA](#), [ImageFormat::Depth](#), [ImageFormat::DepthStencil](#),  
    [ImageFormat::CompressedRGB](#), [ImageFormat::CompressedRGBA](#) }

*Image format used to write texture data.*

- enum [Key](#) {

```

Key::LButton, Key::RButton, Key::Cancel, Key::MButton,
Key::XButton1, Key::XButton2, Key::Back, Key::Tab,
Key::Clear, Key::Return, Key::Shift, Key::Control,
Key::Menu, Key::Pause, Key::Capital, Key::Escape,
Key::Space, Key::PageUp, Key::PageDown, Key::End,
Key::Home, Key::Left, Key::Up, Key::Right,
Key::Down, Key::Select, Key::Print, Key::Exe,
Key::Snapshot, Key::Insert, Key::Delete, Key::Help,
Key::D0, Key::D1, Key::D2, Key::D3,
Key::D4, Key::D5, Key::D6, Key::D7,
Key::D8, Key::D9, Key::A, Key::B,
Key::C, Key::D, Key::E, Key::F,
Key::G, Key::H, Key::I, Key::J,
Key::K, Key::L, Key::M, Key::N,
Key::O, Key::P, Key::Q, Key::R,
Key::S, Key::T, Key::U, Key::V,
Key::W, Key::X, Key::Y, Key::Z,
Key::LWin, Key::RWin, Key::Apps, Key::Sleep,
Key::Keypad0, Key::Keypad1, Key::Keypad2, Key::Keypad3,
Key::Keypad4, Key::Keypad5, Key::Keypad6, Key::Keypad7,
Key::Keypad8, Key::Keypad9, Key::KeypadMultiply, Key::KeypadPlus,
Key::KeypadSeparator, Key::KeypadMinus, Key::KeypadDecimal, Key::KeypadDivide,
Key::F1, Key::F2, Key::F3, Key::F4,
Key::F5, Key::F6, Key::F7, Key::F8,
Key::F9, Key::F10, Key::F11, Key::F12,
Key::F13, Key::F14, Key::F15, Key::F16,
Key::F17, Key::F18, Key::F19, Key::F20,
Key::F21, Key::F22, Key::F23, Key::F24,
Key::NumLock, Key::ScrollLock, Key::LShift, Key::RShift,
Key::LControl, Key::RControl, Key::LMenu, Key::RMenu,
Key::BrowserBack, Key::BrowserForward, Key::BrowserRefresh, Key::BrowserStop,
Key::BrowserSearch, Key::BrowserFavorites, Key::BrowserHome, Key::VolumeMute,
Key::VolumeDown, Key::VolumeUp, Key::MediaNextTrack, Key::MediaPrevTrack,
Key::MediaStop, Key::MediaPlayPause, Key::LaunchMail, Key::LaunchMediaSelect,
Key::LaunchApp1, Key::LaunchApp2, Key::Plus, Key::Comma,
Key::Minus, Key::Period, Key::Exponent, Key::Attn,
Key::CrSel, Key::ExSel, Key::ErEOF, Key::Play,
Key::Zoom, Key::NoName, Key::PA1, Key::OEMClear }

```

*Input key codes.*

- enum QueryType {
 QueryType::SamplesPassed, QueryType::AnySamplesPassed, QueryType::AnySamplesPassedConservative,
 QueryType::PrimitivesGenerated,
 QueryType::TimeElapsed, QueryType::StreamOutPrimitivesWritten, QueryType::StreamOutOverflow,
 QueryType::VerticesSubmitted,
 QueryType::PrimitivesSubmitted, QueryType::VertexShaderInvocations, QueryType::TessControlShader↵
 Invocations, QueryType::TessEvaluationShaderInvocations,
 QueryType::GeometryShaderInvocations, QueryType::FragmentShaderInvocations, QueryType::Compute↵
 ShaderInvocations, QueryType::GeometryPrimitivesGenerated,
 QueryType::ClippingInputPrimitives, QueryType::ClippingOutputPrimitives }

*Query type enumeration.*

- enum OpenGLVersion {
 OpenGLVersion::OpenGL\_Latest = 0, OpenGLVersion::OpenGL\_1\_0 = 100, OpenGLVersion::OpenGL\_1\_1
 = 110, OpenGLVersion::OpenGL\_1\_2 = 120,
 OpenGLVersion::OpenGL\_1\_3 = 130, OpenGLVersion::OpenGL\_1\_4 = 140, OpenGLVersion::OpenGL\_1\_5
 = 150, OpenGLVersion::OpenGL\_2\_0 = 200,
 OpenGLVersion::OpenGL\_2\_1 = 210, OpenGLVersion::OpenGL\_3\_0 = 300, OpenGLVersion::OpenGL\_3\_1
 = 310, OpenGLVersion::OpenGL\_3\_2 = 320,
 OpenGLVersion::OpenGL\_3\_3 = 330, OpenGLVersion::OpenGL\_4\_0 = 400, OpenGLVersion::OpenGL\_4\_1



- = 410, [OpenGLVersion::OpenGL\\_4\\_2](#) = 420,  
[OpenGLVersion::OpenGL\\_4\\_3](#) = 430, [OpenGLVersion::OpenGL\\_4\\_4](#) = 440, [OpenGLVersion::OpenGL\\_4\\_5](#) = 450 }
- enum [SwapChainMode](#) { [SwapChainMode::SingleBuffering](#) = 1, [SwapChainMode::DoubleBuffering](#) = 2, [SwapChainMode::TripleBuffering](#) = 3 }
- Swap chain mode enumeration.*
- enum [RendererInfo](#) { [RendererInfo::Version](#), [RendererInfo::Vendor](#), [RendererInfo::Hardware](#), [RendererInfo::ShadingLanguageVersion](#) }
- Renderer info enumeration.*
- enum [RenderConditionMode](#) { [RenderConditionMode::Wait](#), [RenderConditionMode::NoWait](#), [RenderConditionMode::ByRegionWait](#), [RenderConditionMode::ByRegionNoWait](#), [RenderConditionMode::WaitInverted](#), [RenderConditionMode::NoWaitInverted](#), [RenderConditionMode::ByRegionWaitInverted](#), [RenderConditionMode::ByRegionNoWaitInverted](#) }
- Render condition mode enumeration.*
- enum [ErrorType](#) { [ErrorType::InvalidArgument](#), [ErrorType::InvalidState](#), [ErrorType::UnsupportedFeature](#) }
- Rendering debugger error types enumeration.*
- enum [WarningType](#) { [WarningType::ImproperArgument](#), [WarningType::ImproperState](#), [WarningType::PointlessOperation](#) }
- enum [BufferUsage](#) { [BufferUsage::Static](#), [BufferUsage::Dynamic](#) }
- Hardware buffer usage enumeration.*
- enum [BufferCPUAccess](#) { [BufferCPUAccess::ReadOnly](#), [BufferCPUAccess::WriteOnly](#), [BufferCPUAccess::ReadWrite](#) }
- Hardware buffer CPU access enumeration.*
- enum [ShadingLanguage](#) { [ShadingLanguage::GLSL\\_110](#), [ShadingLanguage::GLSL\\_120](#), [ShadingLanguage::GLSL\\_130](#), [ShadingLanguage::GLSL\\_140](#), [ShadingLanguage::GLSL\\_150](#), [ShadingLanguage::GLSL\\_330](#), [ShadingLanguage::GLSL\\_400](#), [ShadingLanguage::GLSL\\_410](#), [ShadingLanguage::GLSL\\_420](#), [ShadingLanguage::GLSL\\_430](#), [ShadingLanguage::GLSL\\_440](#), [ShadingLanguage::GLSL\\_450](#), [ShadingLanguage::HLSL\\_2\\_0](#), [ShadingLanguage::HLSL\\_2\\_0a](#), [ShadingLanguage::HLSL\\_2\\_0b](#), [ShadingLanguage::HLSL\\_3\\_0](#), [ShadingLanguage::HLSL\\_4\\_0](#), [ShadingLanguage::HLSL\\_4\\_1](#), [ShadingLanguage::HLSL\\_5\\_0](#) }
- Shading language version enumeration.*
- enum [ScreenOrigin](#) { [ScreenOrigin::LowerLeft](#), [ScreenOrigin::UpperLeft](#) }
- Screen coordinate system origin enumeration.*
- enum [ClippingRange](#) { [ClippingRange::MinusOneToOne](#), [ClippingRange::ZeroToOne](#) }
- Clipping depth range enumeration.*
- enum [TextureWrap](#) { [TextureWrap::Repeat](#), [TextureWrap::Mirror](#), [TextureWrap::Clamp](#), [TextureWrap::Border](#), [TextureWrap::MirrorOnce](#) }
- Texture coordinate wrap enumeration.*
- enum [TextureFilter](#) { [TextureFilter::Nearest](#), [TextureFilter::Linear](#) }
- Texture sampling filter enumeration.*
- enum [ShaderType](#) { [ShaderType::Vertex](#), [ShaderType::TessControl](#), [ShaderType::TessEvaluation](#), [ShaderType::Geometry](#), [ShaderType::Fragment](#), [ShaderType::Compute](#) }
- Shader type enumeration.*
- enum [UniformType](#) { [UniformType::Float](#), [UniformType::Float2](#), [UniformType::Float3](#), [UniformType::Float4](#), [UniformType::Double](#), [UniformType::Double2](#), [UniformType::Double3](#), [UniformType::Double4](#), [UniformType::Int](#), [UniformType::Int2](#), [UniformType::Int3](#), [UniformType::Int4](#), [UniformType::Float2x2](#), [UniformType::Float3x3](#), [UniformType::Float4x4](#), [UniformType::Double2x2](#), [UniformType::Double3x3](#), [UniformType::Double4x4](#), [UniformType::Sampler1D](#), [UniformType::Sampler2D](#), [UniformType::Sampler3D](#), [UniformType::SamplerCube](#) }

*Shader uniform type enumeration.*

- enum [StorageBufferType](#) {  
[StorageBufferType::Buffer](#), [StorageBufferType::StructuredBuffer](#), [StorageBufferType::ByteAddressBuffer](#),  
[StorageBufferType::RWBuffer](#),  
[StorageBufferType::RWStructuredBuffer](#), [StorageBufferType::RWByteAddressBuffer](#), [StorageBufferType::↵  
AppendStructuredBuffer](#), [StorageBufferType::ConsumeStructuredBuffer](#) }

*Storage buffer type enumeration.*

- enum [TextureType](#) {  
[TextureType::Undefined](#), [TextureType::Texture1D](#), [TextureType::Texture2D](#), [TextureType::Texture3D](#),  
[TextureType::TextureCube](#), [TextureType::Texture1DArray](#), [TextureType::Texture2DArray](#), [TextureType::↵  
TextureCubeArray](#) }

*Texture type enumeration.*

- enum [TextureFormat](#) {  
[TextureFormat::Unknown](#), [TextureFormat::DepthComponent](#), [TextureFormat::DepthStencil](#), [TextureFormat::↵  
::R](#),  
[TextureFormat::RG](#), [TextureFormat::RGB](#), [TextureFormat::RGBA](#), [TextureFormat::R8](#),  
[TextureFormat::R8Sgn](#), [TextureFormat::R16](#), [TextureFormat::R16Sgn](#), [TextureFormat::R16Float](#),  
[TextureFormat::R32UInt](#), [TextureFormat::R32SInt](#), [TextureFormat::R32Float](#), [TextureFormat::RG8](#),  
[TextureFormat::RG8Sgn](#), [TextureFormat::RG16](#), [TextureFormat::RG16Sgn](#), [TextureFormat::RG16Float](#),  
[TextureFormat::RG32UInt](#), [TextureFormat::RG32SInt](#), [TextureFormat::RG32Float](#), [TextureFormat::RGB8](#),  
[TextureFormat::RGB8Sgn](#), [TextureFormat::RGB16](#), [TextureFormat::RGB16Sgn](#), [TextureFormat::RGB16↵  
Float](#),  
[TextureFormat::RGB32UInt](#), [TextureFormat::RGB32SInt](#), [TextureFormat::RGB32Float](#), [TextureFormat::RG↵  
BA8](#),  
[TextureFormat::RGBA8Sgn](#), [TextureFormat::RGBA16](#), [TextureFormat::RGBA16Sgn](#), [TextureFormat::RGB↵  
A16Float](#),  
[TextureFormat::RGBA32UInt](#), [TextureFormat::RGBA32SInt](#), [TextureFormat::RGBA32Float](#), [TextureFormat::↵  
::RGB\\_DXT1](#),  
[TextureFormat::RGBA\\_DXT1](#), [TextureFormat::RGBA\\_DXT3](#), [TextureFormat::RGBA\\_DXT5](#) }

*Hardware texture format enumeration.*

- enum [AxisDirection](#) {  
[AxisDirection::XPos](#) = 0, [AxisDirection::XNeg](#), [AxisDirection::YPos](#), [AxisDirection::YNeg](#),  
[AxisDirection::ZPos](#), [AxisDirection::ZNeg](#) }

*Axis direction (also used for texture cube face).*

## Functions

- template<typename T >  
T [MaxColorValue](#) ()  
*Returns the maximal color value for the data type T. By default 1.*
- template<>  
unsigned char [MaxColorValue](#)< unsigned char > ()  
*Specialized version. For unsigned 8-bit integers, the return value is 255.*
- template<>  
bool [MaxColorValue](#)< bool > ()  
*Specialized version. For booleans, the return value is true.*
- template<typename T, std::size\_t N>  
[Color](#)< T, N > [operator+](#) (const [Color](#)< T, N > &lhs, const [Color](#)< T, N > &rhs)
- template<typename T, std::size\_t N>  
[Color](#)< T, N > [operator-](#) (const [Color](#)< T, N > &lhs, const [Color](#)< T, N > &rhs)
- template<typename T, std::size\_t N>  
[Color](#)< T, N > [operator\\*](#) (const [Color](#)< T, N > &lhs, const [Color](#)< T, N > &rhs)
- template<typename T, std::size\_t N>  
[Color](#)< T, N > [operator/](#) (const [Color](#)< T, N > &lhs, const [Color](#)< T, N > &rhs)

- `template<typename T, std::size_t N>`  
`Color< T, N > operator* (const Color< T, N > &lhs, const T &rhs)`
- `template<typename T, std::size_t N>`  
`Color< T, N > operator* (const T &lhs, const Color< T, N > &rhs)`
- `template<typename T, std::size_t N>`  
`Color< T, N > operator/ (const Color< T, N > &lhs, const T &rhs)`
- `template<typename T, std::size_t N>`  
`bool operator== (const Color< T, N > &lhs, const Color< T, N > &rhs)`
- `template<typename T, std::size_t N>`  
`bool operator!= (const Color< T, N > &lhs, const Color< T, N > &rhs)`
- `LLGL_EXPORT std::size_t DataTypeSize (const DataType dataType)`  
*Returns the size (in bytes) of the specified data type.*
- `LLGL_EXPORT std::size_t ImageFormatSize (const ImageFormat imageFormat)`  
*Returns the size (in number of components) of the specified image format.*
- `LLGL_EXPORT bool IsCompressedFormat (const ImageFormat format)`  
*Returns true if the specified color format is a compressed format, i.e. either `ImageFormat::CompressedRGB`, or `ImageFormat::CompressedRGBA`.*
- `LLGL_EXPORT bool IsDepthStencilFormat (const ImageFormat format)`  
*Returns true if the specified color format is a depth-stencil format, i.e. either `ImageFormat::Depth` or `ImageFormat::DepthStencil`.*
- `LLGL_EXPORT ByteBuffer ConvertImageBuffer (ImageFormat srcFormat, DataType srcDataType, const void *srcBuffer, std::size_t srcBufferSize, ImageFormat dstFormat, DataType dstDataType, std::size_t threadCount=0)`  
*Converts the image format and data type of the source image (only uncompressed color formats).*
- `LLGL_EXPORT bool operator== (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)`
- `LLGL_EXPORT bool operator!= (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)`
- `LLGL_EXPORT bool operator== (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)`
- `LLGL_EXPORT bool operator!= (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)`
- `LLGL_EXPORT int NumMipLevels (const Gs::Vector3i &textureSize)`  
*Returns the number of MIP-map levels for a texture with the specified size.*
- `LLGL_EXPORT bool IsCompressedFormat (const TextureFormat format)`  
*Returns true if the specified texture format is a compressed format, i.e. either `TextureFormat::RGB_DXT1`, `TextureFormat::RGBA_DXT1`, `TextureFormat::RGBA_DXT3`, or `TextureFormat::RGBA_DXT5`.*
- `LLGL_EXPORT bool operator== (const VertexAttribute &lhs, const VertexAttribute &rhs)`
- `LLGL_EXPORT bool operator!= (const VertexAttribute &lhs, const VertexAttribute &rhs)`
- `LLGL_EXPORT bool operator== (const VideoDisplayMode &lhs, const VideoDisplayMode &rhs)`
- `LLGL_EXPORT bool CompareSWO (const VideoDisplayMode &lhs, const VideoDisplayMode &rhs)`  
*Compares the two video display modes in a strict-weak-order (SWO) fashion.*

### 6.1.1 Typedef Documentation

#### 6.1.1.1 `using LLGL::ByteBuffer = typedef std::unique_ptr<char[]>`

Common byte buffer type.

#### Remarks

Commonly this would be an `std::vector<char>`, but the buffer conversion is an optimized process, where the default initialization of an `std::vector` is undesired. Therefore, the byte buffer type is an `std::unique_ptr<char[]>`.

#### See also

[ConvertImageBuffer](#)

6.1.1.2 using LLGL::ColorRGB = typedef ColorRGBT<Gs::Real>

6.1.1.3 using LLGL::ColorRGBA = typedef ColorRGBAT<Gs::Real>

6.1.1.4 using LLGL::ColorRGBAb = typedef ColorRGBAT<bool>

6.1.1.5 using LLGL::ColorRGBAd = typedef ColorRGBAT<double>

6.1.1.6 using LLGL::ColorRGBAf = typedef ColorRGBAT<float>

6.1.1.7 template<typename T > using LLGL::ColorRGBAT = typedef Color<T, 4>

6.1.1.8 using LLGL::ColorRGBAub = typedef ColorRGBAT<unsigned char>

6.1.1.9 using LLGL::ColorRGBb = typedef ColorRGBT<bool>

6.1.1.10 using LLGL::ColorRGBd = typedef ColorRGBT<double>

6.1.1.11 using LLGL::ColorRGBf = typedef ColorRGBT<float>

6.1.1.12 template<typename T > using LLGL::ColorRGBT = typedef Color<T, 3>

6.1.1.13 using LLGL::ColorRGBub = typedef ColorRGBT<unsigned char>

6.1.1.14 using LLGL::DebugCallback = typedef std::function<void(const std::string& type, const std::string& message)>

Debug callback function interface.

#### Parameters

in	<i>type</i>	Descriptive type of the message.
in	<i>message</i>	Specifies the debug output message.

#### Remarks

This output is renderer dependent.

6.1.1.15 using LLGL::Point = typedef Gs::Vector2i

2D point (integer)

6.1.1.16 using LLGL::Size = typedef Gs::Vector2i

2D size (integer)

## 6.1.2 Enumeration Type Documentation

### 6.1.2.1 enum LLGL::AxisDirection [strong]

Axis direction (also used for texture cube face).

Enumerator

**XPos** X+ direction.

**XNeg** X- direction.

**YPos** Y+ direction.

**YNeg** Y- direction.

**ZPos** Z+ direction.

**ZNeg** Z- direction.

### 6.1.2.2 enum LLGL::BlendArithmetic [strong]

Blending arithmetic operations enumeration.

Enumerator

**Add** Add source 1 and source 2. This is the default for all renderers.

**Subtract** Subtract source 1 from source 2.

**RevSubtract** Subtract source 2 from source 1.

**Min** Find the minimum of source 1 and source 2.

**Max** Find the maximum of source 1 and source 2.

### 6.1.2.3 enum LLGL::BlendOp [strong]

Blending operations enumeration.

Enumerator

**Zero**

**One**

**SrcColor**

**InvSrcColor**

**SrcAlpha**

**InvSrcAlpha**

**DestColor**

**InvDestColor**

**DestAlpha**

**InvDestAlpha**

#### 6.1.2.4 enum LLGL::BufferCPUAccess [strong]

Hardware buffer CPU access enumeration.

See also

`RenderSystem::MapBuffer`

Enumerator

**ReadOnly** CPU read access only.

**WriteOnly** CPU write access only.

**ReadWrite** CPU read and write access.

#### 6.1.2.5 enum LLGL::BufferUsage [strong]

Hardware buffer usage enumeration.

Remarks

For OpenGL, the buffer usage is just a hint to the GL server. For Direct3D, the buffer usage is crucial during buffer creation.

See also

[RenderSystem::CreateVertexBuffer](#)

[RenderSystem::CreateIndexBuffer](#)

[RenderSystem::CreateConstantBuffer](#)

[RenderSystem::CreateStorageBuffer](#)

Enumerator

**Static** The hardware buffer will be rarely changed by the client but often used by the hardware.

Remarks

For Direct3D 11, a buffer can use the static buffer usage, if always the entire buffer will be updated. Otherwise, the dynamic buffer usage must be used.

**Dynamic** The hardware buffer will be often changed by the client (e.g. almost every frame).

Remarks

For Direct3D 11, a buffer must use the dynamic buffer usage, if it will only partially be updated at any time.

#### 6.1.2.6 enum LLGL::ClippingRange [strong]

Clipping depth range enumeration.

Enumerator

**MinusOneToOne** Clipping depth is in the range [-1, 1] (default in OpenGL).

**ZeroToOne** Clipping depth is in the range [0, 1] (default in Direct3D).

### 6.1.2.7 enum LLGL::CompareOp [strong]

Compare operations enumeration.

#### Remarks

This operation is used for depth-test and stencil-test.

#### Enumerator

**Never** Compare test never succeeds.

**Less** Compare test succeeds if the left-hand-side is less than the right-hand-side.

**Equal** Compare test succeeds if the left-hand-side is equal to the right-hand-side.

**LessEqual** Compare test succeeds if the left-hand-side is less than or equal to the right-hand-side.

**Greater** Compare test succeeds if the left-hand-side is greater than the right-hand-side.

**NotEqual** Compare test succeeds if the left-hand-side is not equal to the right-hand-side.

**GreaterEqual** Compare test succeeds if the left-hand-side is greater than or equal to the right-hand-side.

**Ever** Compare test always succeeds. (Can not be called "Always" due to conflict with X11 lib on Linux).

### 6.1.2.8 enum LLGL::CullMode [strong]

Polygon culling modes enumeration.

#### Enumerator

**Disabled** No culling.

**Front** Front face culling.

**Back** Back face culling.

### 6.1.2.9 enum LLGL::DataType [strong]

Renderer data types enumeration.

#### Enumerator

**Int8** 8-bit signed integer (char).

**UInt8** 8-bit unsigned integer (unsigned char).

**Int16** 16-bit signed integer (short).

**UInt16** 16-bit unsigned integer (unsigned short).

**Int32** 32-bit signed integer (int).

**UInt32** 32-bit unsigned integer (unsigned int).

**Float** 32-bit floating-point (float).

**Double** 64-bit real type (double).

#### 6.1.2.10 enum LLGL::ErrorType [strong]

Rendering debugger error types enumeration.

Enumerator

**InvalidArgument** Error due to invalid argument (e.g. creating a graphics pipeline without a valid shader program being specified).

**InvalidState** Error due to invalid render state (e.g. rendering without a valid graphics pipeline).

**UnsupportedFeature** Error due to use of unsupported feature (e.g. drawing with hardware instancing when the renderer hardware does not support it).

#### 6.1.2.11 enum LLGL::ImageFormat [strong]

Image format used to write texture data.

Enumerator

**R** Single color component: Red.

**RG** Two color components: Red, Green.

**RGB** Three color components: Red, Green, Blue.

**BGR** Three color components: Blue, Green, Red.

**RGBA** Four color components: Red, Green, Blue, Alpha.

**BGRA** Four color components: Blue, Green, Red, Alpha.

**Depth** 32-bit depth component.

**DepthStencil** 24-bit depth- and 8-bit stencil component.

**CompressedRGB** Generic compressed format with three color components: Red, Green, Blue.

**CompressedRGBA** Generic compressed format with four color components: Red, Green, Blue, Alpha.

#### 6.1.2.12 enum LLGL::Key [strong]

Input key codes.

Enumerator

**LButton** Left mouse button.

**RButton** Right mouse button.

**Cancel** Control-break processing.

**MButton** Middle mouse button (three-button mouse).

**XButton1** Windows 2000/XP: X1 mouse button.

**XButton2** Windows 2000/XP: X2 mouse button.

**Back** BACKSPACE key.

**Tab** TAB key.

**Clear** CLEAR key.

**Return** RETURN (or ENTER) key.

**Shift** SHIFT key.



**Control** CTRL key.

**Menu** ALT key.

**Pause** PAUSE key.

**Capital** CAPS LOCK key.

**Escape** Escape (ESC) key.

**Space** Space key.

**PageUp** Page up key.

**PageDown** Page down key.

**End** END key.

**Home** HOME (or POS1) key.

**Left** Left arrow key.

**Up** Up arrow key.

**Right** Right arrow key.

**Down** Down arrow key.

**Select** Select key.

**Print** Print key.

**Exe** Execute key.

**Snapshot** Snapshot key.

**Insert** Insert key.

**Delete** Delete key.

**Help** Help key.

**D0** Digit 0.

**D1** Digit 1.

**D2** Digit 2.

**D3** Digit 3.

**D4** Digit 4.

**D5** Digit 5.

**D6** Digit 6.

**D7** Digit 7.

**D8** Digit 8.

**D9** Digit 9.

**A** Letter A.

**B** Letter B.

**C** Letter C.

**D** Letter D.

**E** Letter E.

**F** Letter F.

**G** Letter G.

**H** Letter H.

**I** Letter I.

**J** Letter J.

**K** Letter K.

**L** Letter L.

**M** Letter M.

**N** Letter N.

**O** Letter O.

**P** Letter P.

**Q** Letter Q.

**R** Letter R.

**S** Letter S.

**T** Letter T.

**U** Letter U.

**V** Letter V.

**W** Letter W.

**X** Letter X.

**Y** Letter Y.

**Z** Letter Z.

**LWin** Left Windows key.

**RWin** Righth Windows key.

**Apps** Application key.

**Sleep** Sleep key.

**Keypad0** Keypad 0 key.

**Keypad1** Keypad 1 key.

**Keypad2** Keypad 2 key.

**Keypad3** Keypad 3 key.

**Keypad4** Keypad 4 key.

**Keypad5** Keypad 5 key.

**Keypad6** Keypad 6 key.

**Keypad7** Keypad 7 key.

**Keypad8** Keypad 8 key.

**Keypad9** Keypad 9 key.

**KeypadMultiply** Keypad multiply '\*'.

**KeypadPlus** Keypad plus '+'.

**KeypadSeparator** Keypad separator.

**KeypadMinus** Keypad minus '-'.

**KeypadDecimal** Keypad decimal ',' or '.' (depends on language).

**KeypadDivide** Keypad divide '/'.

**F1** F1 function key.

**F2** F2 function key.

**F3** F3 function key.

**F4** F4 function key.

**F5** F5 function key.

**F6** F6 function key.

**F7** F7 function key.

**F8** F8 function key.

**F9** F9 function key.

**F10** F10 function key.

**F11** F11 function key.

**F12** F12 function key.

**F13** F13 function key.

**F14** F14 function key.  
**F15** F15 function key.  
**F16** F16 function key.  
**F17** F17 function key.  
**F18** F18 function key.  
**F19** F19 function key.  
**F20** F20 function key.  
**F21** F21 function key.  
**F22** F22 function key.  
**F23** F23 function key.  
**F24** F24 function key.  
**NumLock** Num lock key.  
**ScrollLock** Scroll lock key.  
**LShift** Left shift key.  
**RShift** Right shift key.  
**LControl** Left control (CTRL) key.  
**RControl** Right control (CTRL) key.  
**LMenu** Left menu key.  
**RMenu** Right menu key.  
**BrowserBack**  
**BrowserForward**  
**BrowserRefresh**  
**BrowserStop**  
**BrowserSearch**  
**BrowserFavorites**  
**BrowserHome**  
**VolumeMute**  
**VolumeDown**  
**VolumeUp**  
**MediaNextTrack**  
**MediaPrevTrack**  
**MediaStop**  
**MediaPlayPause**  
**LaunchMail**  
**LaunchMediaSelect**  
**LaunchApp1**  
**LaunchApp2**  
**Plus** '+'  
**Comma** ','  
**Minus** '-'  
**Period** '.'  
**Exponent** '^'  
**Attn**  
**CrSel**  
**ExSel**

***ErEOF***  
***Play***  
***Zoom***  
***NoName***  
***PA1***  
***OEMClear***

#### 6.1.2.13 enum LLGL::OpenGLVersion [strong]

##### Enumerator

***OpenGL\_Latest*** Latest available OpenGL version (on the host platform).  
***OpenGL\_1\_0*** OpenGL 1.0, released in Jan, 1992.  
***OpenGL\_1\_1*** OpenGL 1.1, released in Mar, 1997.  
***OpenGL\_1\_2*** OpenGL 1.2, released in Mar, 1998.  
***OpenGL\_1\_3*** OpenGL 1.3, released in Aug, 2001.  
***OpenGL\_1\_4*** OpenGL 1.4, released in Jul, 2002.  
***OpenGL\_1\_5*** OpenGL 1.5, released in Jul, 2003.  
***OpenGL\_2\_0*** OpenGL 2.0, released in Sep, 2004.  
***OpenGL\_2\_1*** OpenGL 2.1, released in Jul, 2006.  
***OpenGL\_3\_0*** OpenGL 3.0, released in Aug, 2008 (known as "Longs Peak").  
***OpenGL\_3\_1*** OpenGL 3.1, released in Mar, 2009 (known as "Longs Peak Reloaded").  
***OpenGL\_3\_2*** OpenGL 3.2, released in Aug, 2009.  
***OpenGL\_3\_3*** OpenGL 3.3, released in Mar, 2010.  
***OpenGL\_4\_0*** OpenGL 4.0, released in Mar, 2010 (alongside with OpenGL 3.3).  
***OpenGL\_4\_1*** OpenGL 4.1, released in Jul, 2010.  
***OpenGL\_4\_2*** OpenGL 4.2, released in Aug, 2011.  
***OpenGL\_4\_3*** OpenGL 4.3, released in Aug, 2012.  
***OpenGL\_4\_4*** OpenGL 4.4, released in Jul, 2013.  
***OpenGL\_4\_5*** OpenGL 4.5, released in Aug, 2014.

#### 6.1.2.14 enum LLGL::PolygonMode [strong]

Polygon filling modes enumeration.

##### Enumerator

***Fill*** Draw filled polygon.  
***Wireframe*** Draw triangle edges only.  
***Points*** Draw vertex points only.

##### Note

Only supported with: OpenGL.

## 6.1.2.15 enum LLGL::PrimitiveTopology [strong]

Primitive topology enumeration.

## Enumerator

**PointList** Point list.

**LineList** Line list where each line has its own two vertices.

**LineStrip** Line strip where each line after the first one begins with the previous vertex.

**LineLoop** Line loop which is similiar to line strip but the last line ends with the first vertex.

## Note

Only supported with: OpenGL.

**LineListAdjacency** Adjacency line list.

**LineStripAdjacency** Adjacency line strips.

**TriangleList** Triangle list where each triangle has its own three vertices.

**TriangleStrip** Triangle strip where each triangle after the first one begins with the previous vertex.

**TriangleFan** Triangle fan where each triangle uses the first vertex, the previous vertex, and a new vertex.

## Note

Only supported with: OpenGL.

**TriangleListAdjacency** Adjacency triangle list.

**TriangleStripAdjacency** Adjacency triangle strips.

**Patches1** Patches with 1 control point.

**Patches2** Patches with 2 control points.

**Patches3** Patches with 3 control points.

**Patches4** Patches with 4 control points.

**Patches5** Patches with 5 control points.

**Patches6** Patches with 6 control points.

**Patches7** Patches with 7 control points.

**Patches8** Patches with 8 control points.

**Patches9** Patches with 9 control points.

**Patches10** Patches with 10 control points.

**Patches11** Patches with 11 control points.

**Patches12** Patches with 12 control points.

**Patches13** Patches with 13 control points.

**Patches14** Patches with 14 control points.

**Patches15** Patches with 15 control points.

**Patches16** Patches with 16 control points.

**Patches17** Patches with 17 control points.

**Patches18** Patches with 18 control points.

**Patches19** Patches with 19 control points.

**Patches20** Patches with 20 control points.

**Patches21** Patches with 21 control points.

**Patches22** Patches with 22 control points.

**Patches23** Patches with 23 control points.

**Patches24** Patches with 24 control points.

**Patches25** Patches with 25 control points.

**Patches26** Patches with 26 control points.

**Patches27** Patches with 27 control points.

**Patches28** Patches with 28 control points.

**Patches29** Patches with 29 control points.

**Patches30** Patches with 30 control points.

**Patches31** Patches with 31 control points.

**Patches32** Patches with 32 control points.

#### 6.1.2.16 enum LLGL::QueryType [strong]

Query type enumeration.

#### Enumerator

**SamplesPassed** Number of samples that passed the depth test. This can be used as render condition.

**AnySamplesPassed** Non-zero if any samples passed the depth test. This can be used as render condition.

**AnySamplesPassedConservative** Non-zero if any samples passed the depth test within a conservative rasterization. This can be used as render condition.

**PrimitivesGenerated** Number of generated primitives which are send to the rasterizer (either emitted from the geometry or vertex shader).

**TimeElapsed** Elapsed time (in nanoseconds) between the begin- and end query command.

**StreamOutPrimitivesWritten** Number of vertices that have been written into a stream output (also called "Transform Feedback").

**StreamOutOverflow** Non-zero if any of the streaming output buffers (also called "Transform Feedback Buffers") has an overflow.

**VerticesSubmitted** Number of vertices submitted to the input-assembly.

**PrimitivesSubmitted** Number of primitives submitted to the input-assembly.

**VertexShaderInvocations** Number of vertex shader invocations.

**TessControlShaderInvocations** Number of tessellation-control shader invocations.

**TessEvaluationShaderInvocations** Number of tessellation-evaluation shader invocations.

**GeometryShaderInvocations** Number of geometry shader invocations.

**FragmentShaderInvocations** Number of fragment shader invocations.

**ComputeShaderInvocations** Number of compute shader invocations.

**GeometryPrimitivesGenerated** Number of primitives generated by the geometry shader.

**ClippingInputPrimitives** Number of primitives that reached the primitive clipping stage.

**ClippingOutputPrimitives** Number of primitives that passed the primitive clipping stage.

#### 6.1.2.17 enum LLGL::RenderConditionMode [strong]

Render condition mode enumeration.

##### Remarks

The condition is determined by the type of the [Query](#) object.

##### See also

[RenderContext::BeginRenderCondition](#)

##### Enumerator

**Wait** Wait until the occlusion query result is available, before conditional rendering begins.

**NoWait** Do not wait until the occlusion query result is available, before conditional rendering begins.

**ByRegionWait** Similar to Wait, but the renderer may discard the results of commands for any framebuffer region that did not contribute to the occlusion query.

**ByRegionNoWait** Similar to NoWait, but the renderer may discard the results of commands for any framebuffer region that did not contribute to the occlusion query.

**WaitInverted** Same as Wait, but the condition is inverted.

**NoWaitInverted** Same as NoWait, but the condition is inverted.

**ByRegionWaitInverted** Same as ByRegionWait, but the condition is inverted.

**ByRegionNoWaitInverted** Same as ByRegionNoWait, but the condition is inverted.

#### 6.1.2.18 enum LLGL::RendererInfo [strong]

Renderer info enumeration.

##### See also

[RenderContext::QueryRendererInfo](#)

##### Enumerator

**Version**

**Vendor**

**Hardware**

**ShadingLanguageVersion**

#### 6.1.2.19 enum LLGL::ScreenOrigin [strong]

Screen coordinate system origin enumeration.

##### Enumerator

**LowerLeft** Screen origin is in the lower-left (default in OpenGL).

**UpperLeft** Screen origin is in the upper-left (default in Direct3D).

#### 6.1.2.20 enum LLGL::ShaderType [strong]

Shader type enumeration.

Enumerator

**Vertex** Vertex shader type.

**TessControl** Tessellation control shader type (also "Hull Shader").

**TessEvaluation** Tessellation evaluation shader type (also "Domain Shader").

**Geometry** Geometry shader type.

**Fragment** Fragment shader type (also "Pixel Shader").

**Compute** Compute shader type.

#### 6.1.2.21 enum LLGL::ShadingLanguage [strong]

Shading language version enumeration.

Enumerator

**GLSL\_110** GLSL 1.10 (since OpenGL 2.0).

**GLSL\_120** GLSL 1.20 (since OpenGL 2.1).

**GLSL\_130** GLSL 1.30 (since OpenGL 3.0).

**GLSL\_140** GLSL 1.40 (since OpenGL 3.1).

**GLSL\_150** GLSL 1.50 (since OpenGL 3.2).

**GLSL\_330** GLSL 3.30 (since OpenGL 3.3).

**GLSL\_400** GLSL 4.00 (since OpenGL 4.0).

**GLSL\_410** GLSL 4.10 (since OpenGL 4.1).

**GLSL\_420** GLSL 4.20 (since OpenGL 4.2).

**GLSL\_430** GLSL 4.30 (since OpenGL 4.3).

**GLSL\_440** GLSL 4.40 (since OpenGL 4.4).

**GLSL\_450** GLSL 4.50 (since OpenGL 4.5).

**HLSL\_2\_0** HLSL 2.0 (since Direct3D 9).

**HLSL\_2\_0a** HLSL 2.0a (since Direct3D 9a).

**HLSL\_2\_0b** HLSL 2.0b (since Direct3D 9b).

**HLSL\_3\_0** HLSL 3.0 (since Direct3D 9c).

**HLSL\_4\_0** HLSL 4.0 (since Direct3D 10).

**HLSL\_4\_1** HLSL 4.1 (since Direct3D 10.1).

**HLSL\_5\_0** HLSL 5.0 (since Direct3D 11).

#### 6.1.2.22 enum LLGL::StencilOp [strong]

Stencil operations enumeration.

Enumerator

**Keep**

**Zero**

**Replace**

**IncClamp**

**DecClamp**

**Invert**

**IncWrap**

**DecWrap**



#### 6.1.2.23 enum LLGL::StorageBufferType [strong]

Storage buffer type enumeration.

##### Note

Only supported with: Direct3D 11, Direct3D 12.

##### Enumerator

- Buffer** Typed buffer.
- StructuredBuffer** Structured buffer.
- ByteAddressBuffer** Byte-address buffer.
- RWBuffer** Typed read/write buffer.
- RWStructuredBuffer** Structured read/write buffer.
- RWByteAddressBuffer** Byte-address read/write buffer.
- AppendStructuredBuffer** Append structured buffer.
- ConsumeStructuredBuffer** Consume structured buffer.

#### 6.1.2.24 enum LLGL::SwapChainMode [strong]

Swap chain mode enumeration.

##### Enumerator

- SingleBuffering** Single buffering. This is almost no longer used.
- DoubleBuffering** Double buffering. This is the default for most renderers.
- TripleBuffering** Triple buffering. Triple buffering can only be used for Direct3D renderers.

#### 6.1.2.25 enum LLGL::TextureFilter [strong]

[Texture](#) sampling filter enumeration.

##### Enumerator

- Nearest** Take the nearest sample.
- Linear** Interpolate between two samples.

### 6.1.2.26 enum LLGL::TextureFormat [strong]

Hardware texture format enumeration.

#### Note

All integral 32-bit formats are un-normalized!

#### Enumerator

**Unknown** Unknown texture format.

**DepthComponent** Base format: depth component.

**DepthStencil** Base format: depth- and stencil components.

**R** Base format: red component.

**RG** Base format: red and green components.

**RGB** Base format: red, green, and blue components.

#### Note

Only supported with: OpenGL.

**RGBA** Base format: red, green, blue, and alpha components.

**R8** Sized format: red 8-bit normalized unsigned integer component.

**R8Sgn** Sized format: red 8-bit normalized signed integer component.

**R16** Sized format: red 16-bit normalized unsigned integer component.

**R16Sgn** Sized format: red 16-bit normalized signed integer component.

**R16Float** Sized format: red 16-bit floating point component.

**R32UInt** Sized format: red 32-bit un-normalized unsigned integer component.

**R32SInt** Sized format: red 32-bit un-normalized signed integer component.

**R32Float** Sized format: red 32-bit floating point component.

**RG8** Sized format: red, green 8-bit normalized unsigned integer components.

**RG8Sgn** Sized format: red, green 8-bit normalized signed integer components.

**RG16** Sized format: red, green 16-bit normalized unsigned integer components.

**RG16Sgn** Sized format: red, green 16-bit normalized signed integer components.

**RG16Float** Sized format: red, green 16-bit floating point components.

**RG32UInt** Sized format: red, green 32-bit un-normalized unsigned integer components.

**RG32SInt** Sized format: red, green 32-bit un-normalized signed integer components.

**RG32Float** Sized format: red, green 32-bit floating point components.

**RGB8** Sized format: red, green, blue 8-bit normalized unsigned integer components.

#### Note

Only supported with: OpenGL.

**RGB8Sgn** Sized format: red, green, blue 8-bit normalized signed integer components.

#### Note

Only supported with: OpenGL.

**RGB16** Sized format: red, green, blue 16-bit normalized unsigned integer components.

#### Note

Only supported with: OpenGL.

**RGB16Sgn** Sized format: red, green, blue 16-bit normalized signed integer components.

## Note

Only supported with: OpenGL.

**RGB16Float** Sized format: red, green, blue 16-bit floating point components.

## Note

Only supported with: OpenGL.

**RGB32UInt** Sized format: red, green, blue 32-bit un-normalized unsigned interger components.

**RGB32SInt** Sized format: red, green, blue 32-bit un-normalized signed interger components.

**RGB32Float** Sized format: red, green, blue 32-bit floating point components.

**RGBA8** Sized format: red, green, blue, alpha 8-bit normalized unsigned integer components.

**RGBA8Sgn** Sized format: red, green, blue, alpha 8-bit normalized signed integer components.

**RGBA16** Sized format: red, green, blue, alpha 16-bit normalized unsigned interger components.

**RGBA16Sgn** Sized format: red, green, blue, alpha 16-bit normalized signed interger components.

**RGBA16Float** Sized format: red, green, blue, alpha 16-bit floating point components.

**RGBA32UInt** Sized format: red, green, blue, alpha 32-bit un-normalized unsigned interger components.

**RGBA32SInt** Sized format: red, green, blue, alpha 32-bit un-normalized signed interger components.

**RGBA32Float** Sized format: red, green, blue, alpha 32-bit floating point components.

**RGB\_DXT1** Compressed format: RGB S3TC DXT1.

**RGBA\_DXT1** Compressed format: RGBA S3TC DXT1.

**RGBA\_DXT3** Compressed format: RGBA S3TC DXT3.

**RGBA\_DXT5** Compressed format: RGBA S3TC DXT5.

## 6.1.2.27 enum LLGL::TextureType [strong]

[Texture](#) type enumeration.

## Enumerator

**Undefined** Initial value of a [Texture](#) object.

**Texture1D** 1-Dimensional texture.

**Texture2D** 2-Dimensional texture.

**Texture3D** 3-Dimensional texture.

**TextureCube** Cube texture.

**Texture1DArray** 1-Dimensional array texture.

**Texture2DArray** 2-Dimensional array texture.

**TextureCubeArray** Cube array texture.

## 6.1.2.28 enum LLGL::TextureWrap [strong]

[Texture](#) coordinate wrap enumeration.

## Enumerator

**Repeat** Repeat texture coordinates within the interval [0, 1).

**Mirror** Flip texture coordinates at ever integer junction.

**Clamp** Clamp texture coordinates to the interval [0, 1].

**Border** Clamp texture coordinates to their border.

**MirrorOnce** Takes the absolute value of the texture coordinates and then clamps it to the interval [0, 1], i.e. mirror around 0.

## 6.1.2.29 enum LLGL::UniformType [strong]

Shader uniform type enumeration.

Enumerator

**Float** float uniform.  
**Float2** float2/ vec2 uniform.  
**Float3** float3/ vec3 uniform.  
**Float4** float4/ vec4 uniform.  
**Double** double uniform.  
**Double2** double2/ dvec2 uniform.  
**Double3** double3/ dvec3 uniform.  
**Double4** double4/ dvec4 uniform.  
**Int** int uniform.  
**Int2** int2/ ivec2 uniform.  
**Int3** int3/ ivec3 uniform.  
**Int4** int4/ ivec4 uniform.  
**Float2x2** float2x2/ mat2 uniform.  
**Float3x3** float3x3/ mat3 uniform.  
**Float4x4** float4x4/ mat4 uniform.  
**Double2x2** double2x2/ dmat2 uniform.  
**Double3x3** double3x3/ dmat3 uniform.  
**Double4x4** double4x4/ dmat4 uniform.  
**Sampler1D** sampler1D uniform.  
**Sampler2D** sampler2D uniform.  
**Sampler3D** sampler3D uniform.  
**SamplerCube** samplerCube uniform.

## 6.1.2.30 enum LLGL::WarningType [strong]

Enumerator

**ImproperArgument** Warning due to improper argument (e.g. generating 4 vertices while having triangle list as primitive topology).  
**ImproperState** Warning due to improper state (e.g. rendering while viewport is not visible).  
**PointlessOperation** Warning due to a operation without any effect (e.g. drawing with 0 vertices).

## 6.1.3 Function Documentation

## 6.1.3.1 LLGL\_EXPORT bool LLGL::CompareSWO ( const VideoDisplayMode &amp; lhs, const VideoDisplayMode &amp; rhs )

Compares the two video display modes in a strict-weak-order (SWO) fashion.

## 6.1.3.2 LLGL\_EXPORT ByteBuffer LLGL::ConvertImageBuffer ( ImageFormat srcFormat, DataType srcDataType, const void \* srcBuffer, std::size\_t srcBufferSize, ImageFormat dstFormat, DataType dstDataType, std::size\_t threadCount = 0 )

Converts the image format and data type of the source image (only uncompressed color formats).

## Parameters

in	<i>srcFormat</i>	Specifies the source image format.
in	<i>srcDataType</i>	Specifies the source data type.
in	<i>srcBuffer</i>	Pointer to the source image buffer which is to be converted.
in	<i>srcBufferSize</i>	Specifies the size (in bytes) of the source image buffer.
in	<i>dstFormat</i>	Specifies the destination image format.
in	<i>dstDataType</i>	Specifies the destination data type.
in	<i>threadCount</i>	Specifies the number of threads to use for conversion. If this is less than 2, no multi-threading is used. If this is 'maxThreadCount', the maximal count of threads the system supports will be used (e.g. 4 on a quad-core processor). By default 0.

## Returns

Byte buffer with the converted image data or null if no conversion is necessary. This can be casted to the respective target data type (e.g. "unsigned char", "int", "float" etc.).

## Remarks

Compressed images and depth-stencil images can not be converted.

## Exceptions

<i>std::invalid_argument</i>	If a compressed image format is specified either as source or destination, if a depth-stencil format is specified either as source or destination, if the source buffer size is not a multiple of the source data type size times the image format size, or if 'srcBuffer' is a null pointer.
------------------------------	---

## See also

maxThreadCount

[ByteBuffer](#)

[DataTypeSize](#)

### 6.1.3.3 LLGL\_EXPORT std::size\_t LLGL::DataTypeSize ( const DataType *dataType* )

Returns the size (in bytes) of the specified data type.

### 6.1.3.4 LLGL\_EXPORT std::size\_t LLGL::ImageFormatSize ( const ImageFormat *imageFormat* )

Returns the size (in number of components) of the specified image format.

## Parameters

in	<i>imageFormat</i>	Specifies the image format.
----	--------------------	-----------------------------

**Returns**

Number of components of the specified image format, or 0 if 'imageFormat' specifies a compressed color format.

**See also**

[IsCompressedFormat\(const ImageFormat\)](#)

#### 6.1.3.5 **LLGL\_EXPORT** bool LLGL::IsCompressedFormat ( const ImageFormat *format* )

Returns true if the specified color format is a compressed format, i.e. either [ImageFormat::CompressedRGB](#), or [ImageFormat::CompressedRGBA](#).

**See also**

[ImageFormat](#)

#### 6.1.3.6 **LLGL\_EXPORT** bool LLGL::IsCompressedFormat ( const TextureFormat *format* )

Returns true if the specified texture format is a compressed format, i.e. either [TextureFormat::RGB\\_DXT1](#), [TextureFormat::RGBA\\_DXT1](#), [TextureFormat::RGBA\\_DXT3](#), or [TextureFormat::RGBA\\_DXT5](#).

**See also**

[TextureFormat](#)

#### 6.1.3.7 **LLGL\_EXPORT** bool LLGL::IsDepthStencilFormat ( const ImageFormat *format* )

Returns true if the specified color format is a depth-stencil format, i.e. either [ImageFormat::Depth](#) or [ImageFormat::DepthStencil](#).

#### 6.1.3.8 **template<typename T > T LLGL::MaxColorValue ( )** `[inline]`

Returns the maximal color value for the data type T. By default 1.

#### 6.1.3.9 **template<> bool LLGL::MaxColorValue< bool > ( )** `[inline]`

Specialized version. For booleans, the return value is true.

#### 6.1.3.10 **template<> unsigned char LLGL::MaxColorValue< unsigned char > ( )** `[inline]`

Specialized version. For unsigned 8-bit integers, the return value is 255.

6.1.3.11 **LLGL\_EXPORT** int LLGL::NumMipLevels ( const Gs::Vector3i & *textureSize* )

Returns the number of MIP-map levels for a texture with the specified size.

Returns

$1 + \text{floor}(\log_2(\max\{x, y, z\}))$ .

6.1.3.12 **LLGL\_EXPORT** bool LLGL::operator!= ( const VertexAttribute & *lhs*, const VertexAttribute & *rhs* )

6.1.3.13 **LLGL\_EXPORT** bool LLGL::operator!= ( const VsyncDescriptor & *lhs*, const VsyncDescriptor & *rhs* )

6.1.3.14 **LLGL\_EXPORT** bool LLGL::operator!= ( const VideoModeDescriptor & *lhs*, const VideoModeDescriptor & *rhs* )

6.1.3.15 **template**<typename T, std::size\_t N> bool LLGL::operator!= ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

6.1.3.16 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator\* ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

6.1.3.17 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator\* ( const Color< T, N > & *lhs*, const T & *rhs* )

6.1.3.18 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator\* ( const T & *lhs*, const Color< T, N > & *rhs* )

6.1.3.19 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator+ ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

6.1.3.20 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator- ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

6.1.3.21 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator/ ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

6.1.3.22 **template**<typename T, std::size\_t N> Color<T, N> LLGL::operator/ ( const Color< T, N > & *lhs*, const T & *rhs* )

6.1.3.23 **LLGL\_EXPORT** bool LLGL::operator== ( const VertexAttribute & *lhs*, const VertexAttribute & *rhs* )

6.1.3.24 **LLGL\_EXPORT** bool LLGL::operator== ( const VideoDisplayMode & *lhs*, const VideoDisplayMode & *rhs* )

6.1.3.25 **LLGL\_EXPORT** bool LLGL::operator== ( const VsyncDescriptor & *lhs*, const VsyncDescriptor & *rhs* )

6.1.3.26 **LLGL\_EXPORT** bool LLGL::operator== ( const VideoModeDescriptor & *lhs*, const VideoModeDescriptor & *rhs* )

6.1.3.27 **template**<typename T, std::size\_t N> bool LLGL::operator== ( const Color< T, N > & *lhs*, const Color< T, N > & *rhs* )

## 6.2 LLGL::Desktop Namespace Reference

### Functions

- [LLGL\\_EXPORT Size GetResolution \(\)](#)

- Returns the desktop resolution.*
- `LLGL_EXPORT` `int` `GetColorDepth` ( )  
*Returns the desktop color depth (bits per pixel).*
- `LLGL_EXPORT` `bool` `SetVideoMode` (const `VideoModeDescriptor` &`videoMode`)  
*Sets the new specified video mode for the desktop (resolution and fullscreen mode).*
- `LLGL_EXPORT` `bool` `ResetVideoMode` ( )  
*Restes the standard video mode for the desktop.*

### 6.2.1 Function Documentation

#### 6.2.1.1 `LLGL_EXPORT int LLGL::Desktop::GetColorDepth ( )`

Returns the desktop color depth (bits per pixel).

#### 6.2.1.2 `LLGL_EXPORT Size LLGL::Desktop::GetResolution ( )`

Returns the desktop resolution.

#### 6.2.1.3 `LLGL_EXPORT bool LLGL::Desktop::ResetVideoMode ( )`

Restes the standard video mode for the desktop.

#### 6.2.1.4 `LLGL_EXPORT bool LLGL::Desktop::SetVideoMode ( const VideoModeDescriptor & videoMode )`

Sets the new specified video mode for the desktop (resolution and fullscreen mode).

## 6.3 LLGL::Log Namespace Reference

### Functions

- `LLGL_EXPORT` `void` `SetStdOut` (std::ostream &`stream`)  
*Sets the standard output stream. By default std::cout.*
- `LLGL_EXPORT` `void` `SetStdErr` (std::ostream &`stream`)  
*Sets the standard output stream for error and warning messages. By default std::cerr.*
- `LLGL_EXPORT` `std::ostream` & `StdOut` ( )  
*Returns the standard output stream.*
- `LLGL_EXPORT` `std::ostream` & `StdErr` ( )  
*Returns the standard output stream for error and warning messages.*

### 6.3.1 Function Documentation

#### 6.3.1.1 `LLGL_EXPORT void LLGL::Log::SetStdErr ( std::ostream & stream )`

Sets the standard output stream for error and warning messages. By default std::cerr.



6.3.1.2 LLGL\_EXPORT void LLGL::Log::SetStdOut ( std::ostream & *stream* )

Sets the standard output stream. By default std::cout.

6.3.1.3 LLGL\_EXPORT std::ostream& LLGL::Log::StdErr ( )

Returns the standard output stream for error and warning messages.

6.3.1.4 LLGL\_EXPORT std::ostream& LLGL::Log::StdOut ( )

Returns the standard output stream.



## Chapter 7

# Class Documentation

### 7.1 LLGL::AntiAliasingDescriptor Struct Reference

```
#include <RenderContextDescriptor.h>
```

#### Public Attributes

- bool [enabled](#) = false  
*Specifies whether multi-sampling is enabled or disabled. By default disabled.*
- unsigned int [samples](#) = 1  
*Number of samples used for multi-sampling. By default 1.*

#### 7.1.1 Member Data Documentation

##### 7.1.1.1 bool LLGL::AntiAliasingDescriptor::enabled = false

Specifies whether multi-sampling is enabled or disabled. By default disabled.

##### 7.1.1.2 unsigned int LLGL::AntiAliasingDescriptor::samples = 1

Number of samples used for multi-sampling. By default 1.

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

- [RenderContextDescriptor.h](#)

### 7.2 LLGL::BlendDescriptor Struct Reference

Blending state descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

## Public Attributes

- bool [blendEnabled](#) = false  
*Specifies whether blending is enabled or disabled. This applies to all blending targets.*
- `std::vector< BlendTargetDescriptor >` [targets](#)  
*Render-target blend states. A maximum of 8 targets is supported. Further targets will be ignored.*

### 7.2.1 Detailed Description

Blending state descriptor structure.

### 7.2.2 Member Data Documentation

#### 7.2.2.1 bool LLGL::BlendDescriptor::blendEnabled = false

Specifies whether blending is enabled or disabled. This applies to all blending targets.

#### 7.2.2.2 `std::vector<BlendTargetDescriptor>` LLGL::BlendDescriptor::targets

Render-target blend states. A maximum of 8 targets is supported. Further targets will be ignored.

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

- [GraphicsPipelineFlags.h](#)

## 7.3 LLGL::BlendTargetDescriptor Struct Reference

Blend target state descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

## Public Attributes

- [BlendOp](#) [srcColor](#) = [BlendOp::SrcAlpha](#)  
*Source color blending operation.*
- [BlendOp](#) [destColor](#) = [BlendOp::InvSrcAlpha](#)  
*Destination color blending operation.*
- [BlendArithmetic](#) [colorArithmetic](#) = [BlendArithmetic::Add](#)  
*Color blending arithmetic.*
- [BlendOp](#) [srcAlpha](#) = [BlendOp::SrcAlpha](#)  
*Source alpha blending operation.*
- [BlendOp](#) [destAlpha](#) = [BlendOp::InvSrcAlpha](#)  
*Destination alpha blending operation.*
- [BlendArithmetic](#) [alphaArithmetic](#) = [BlendArithmetic::Add](#)  
*Alpha blending arithmetic.*
- [ColorRGBAb](#) [colorMask](#)  
*Specifies which color components are enabled for writing. By default (true, true, true, true).*

### 7.3.1 Detailed Description

Blend target state descriptor structure.

### 7.3.2 Member Data Documentation

#### 7.3.2.1 BlendArithmetic LLGL::BlendTargetDescriptor::alphaArithmetic = BlendArithmetic::Add

Alpha blending arithmetic.

##### Note

Only supported with: Direct3D 11, Direct3D 12.

#### 7.3.2.2 BlendArithmetic LLGL::BlendTargetDescriptor::colorArithmetic = BlendArithmetic::Add

Color blending arithmetic.

##### Note

Only supported with: Direct3D 11, Direct3D 12.

#### 7.3.2.3 ColorRGBAb LLGL::BlendTargetDescriptor::colorMask

Specifies which color components are enabled for writing. By default (true, true, true, true).

#### 7.3.2.4 BlendOp LLGL::BlendTargetDescriptor::destAlpha = BlendOp::InvSrcAlpha

Destination alpha blending operation.

#### 7.3.2.5 BlendOp LLGL::BlendTargetDescriptor::destColor = BlendOp::InvSrcAlpha

Destination color blending operation.

#### 7.3.2.6 BlendOp LLGL::BlendTargetDescriptor::srcAlpha = BlendOp::SrcAlpha

Source alpha blending operation.

#### 7.3.2.7 BlendOp LLGL::BlendTargetDescriptor::srcColor = BlendOp::SrcAlpha

Source color blending operation.

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

- [GraphicsPipelineFlags.h](#)

## 7.4 LLGL::ClearBuffersFlags Struct Reference

Render context clear buffer flags.

```
#include <RenderContextFlags.h>
```

### Public Types

- enum { [Color](#) = (1 << 0), [Depth](#) = (1 << 1), [Stencil](#) = (1 << 2) }

#### 7.4.1 Detailed Description

Render context clear buffer flags.

See also

[RenderContext::ClearBuffers](#)

#### 7.4.2 Member Enumeration Documentation

##### 7.4.2.1 anonymous enum

Enumerator

***Color***

***Depth***

***Stencil***

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

- [RenderContextFlags.h](#)

## 7.5 LLGL::Color< T, N > Class Template Reference

Base color class with N components.

```
#include <Color.h>
```

## Public Member Functions

- [Color](#) ()
- [Color](#) (const [Color](#)< T, N > &rhs)
- [Color](#) (Gs::UninitializeTag)
- [Color](#)< T, N > & [operator+=](#) (const [Color](#)< T, N > &rhs)
- [Color](#)< T, N > & [operator-=](#) (const [Color](#)< T, N > &rhs)
- [Color](#)< T, N > & [operator\\*=](#) (const [Color](#)< T, N > &rhs)
- [Color](#)< T, N > & [operator/=](#) (const [Color](#)< T, N > &rhs)
- [Color](#)< T, N > & [operator\\*=](#) (const T &rhs)
- [Color](#)< T, N > & [operator/=](#) (const T &rhs)
- T & [operator\[\]](#) (std::size\_t component)  
*Returns the specified vector component.*
- const T & [operator\[\]](#) (std::size\_t component) const  
*Returns the specified vector component.*
- [Color](#)< T, N > [operator-](#) () const
- template<typename C >  
[Color](#)< C, N > [Cast](#) () const
- T \* [Ptr](#) ()  
*Returns a pointer to the first element of this vector.*
- const T \* [Ptr](#) () const  
*Returns a constant pointer to the first element of this vector.*

## Static Public Attributes

- static const std::size\_t [components](#) = N  
*Specifies the number of vector components.*

### 7.5.1 Detailed Description

```
template<typename T, std::size_t N>
class LLGL::Color< T, N >
```

Base color class with N components.

#### Template Parameters

<i>T</i>	Specifies the data type of the vector components. This should be a primitive data type such as float, double, int etc.
<i>N</i>	Specifies the number of components. There are specialized templates for N = 3, and 4.

### 7.5.2 Constructor & Destructor Documentation

7.5.2.1 `template<typename T, std::size_t N> LLGL::Color< T, N >::Color ( ) [inline]`

7.5.2.2 `template<typename T, std::size_t N> LLGL::Color< T, N >::Color ( const Color< T, N > & rhs ) [inline]`

7.5.2.3 `template<typename T, std::size_t N> LLGL::Color< T, N >::Color ( Gs::UninitializeTag ) [inline]`

### 7.5.3 Member Function Documentation

7.5.3.1 `template<typename T, std::size_t N> template<typename C > Color<C, N> LLGL::Color< T, N >::Cast ( ) const [inline]`

Returns a type casted instance of this vector.

#### Template Parameters

<i>C</i>	Specifies the static cast type.
----------	---------------------------------

7.5.3.2 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator*= ( const Color< T, N > & rhs ) [inline]`

7.5.3.3 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator*= ( const T & rhs ) [inline]`

7.5.3.4 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator+= ( const Color< T, N > & rhs ) [inline]`

7.5.3.5 `template<typename T, std::size_t N> Color<T, N> LLGL::Color< T, N >::operator- ( ) const [inline]`

7.5.3.6 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator-= ( const Color< T, N > & rhs ) [inline]`

7.5.3.7 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator/= ( const Color< T, N > & rhs ) [inline]`

7.5.3.8 `template<typename T, std::size_t N> Color<T, N>& LLGL::Color< T, N >::operator/= ( const T & rhs ) [inline]`

7.5.3.9 `template<typename T, std::size_t N> T& LLGL::Color< T, N >::operator[] ( std::size_t component ) [inline]`

Returns the specified vector component.

#### Parameters

<i>in</i>	<i>component</i>	Specifies the vector component index. This must be in the range [0, N).
-----------	------------------	---

7.5.3.10 `template<typename T, std::size_t N> const T& LLGL::Color< T, N >::operator[] ( std::size_t component ) const [inline]`

Returns the specified vector component.



## Parameters

in	<i>component</i>	Specifies the vector component index. This must be in the range [0, N).
----	------------------	---

7.5.3.11 `template<typename T, std::size_t N> T* LLGL::Color< T, N >::Ptr ( ) [inline]`

Returns a pointer to the first element of this vector.

7.5.3.12 `template<typename T, std::size_t N> const T* LLGL::Color< T, N >::Ptr ( ) const [inline]`

Returns a constant pointer to the first element of this vector.

## 7.5.4 Member Data Documentation

7.5.4.1 `template<typename T, std::size_t N> const std::size_t LLGL::Color< T, N >::components = N [static]`

Specifies the number of vector components.

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

- [Color.h](#)

## 7.6 LLGL::Color< T, 3u > Class Template Reference

RGB color class with components: r, g, and b.

```
#include <ColorRGB.h>
```

### Public Member Functions

- [Color](#) ()
- [Color](#) (const [Color](#)< T, 3 > &rhs)
- [Color](#) (const T &scalar)
- [Color](#) (const T &r, const T &g, const T &b)
- [Color](#) (Gs::UninitializeTag)
- [Color](#)< T, 3 > & [operator+=](#) (const [Color](#)< T, 3 > &rhs)
- [Color](#)< T, 3 > & [operator-=](#) (const [Color](#)< T, 3 > &rhs)
- [Color](#)< T, 3 > & [operator\\*=](#) (const [Color](#)< T, 3 > &rhs)
- [Color](#)< T, 3 > & [operator/=](#) (const [Color](#)< T, 3 > &rhs)
- [Color](#)< T, 3 > & [operator\\*=](#) (const T &rhs)
- [Color](#)< T, 3 > & [operator/=](#) (const T &rhs)
- [Color](#)< T, 3 > [operator-](#) () const
- T & [operator\[\]](#) (std::size\_t component)  
*Returns the specified color component.*
- const T & [operator\[\]](#) (std::size\_t component) const  
*Returns the specified color component.*
- template<typename C >  
[Color](#)< C, 3 > [Cast](#) () const  
*Returns a type casted instance of this color.*
- T \* [Ptr](#) ()  
*Returns a pointer to the first element of this color.*
- const T \* [Ptr](#) () const  
*Returns a constant pointer to the first element of this color.*

## Public Attributes

- [T r](#)
- [T g](#)
- [T b](#)

## Static Public Attributes

- static const std::size\_t [components](#) = 3  
*Specifies the number of color components.*

### 7.6.1 Detailed Description

```
template<typename T>
class LLGL::Color< T, 3u >
```

RGB color class with components: r, g, and b.

#### Remarks

[Color](#) components are default initialized with their maximal value, i.e. for floating-points, the initial value is 1.0, because this its maximal color value, but for unsigned-bytes, the initial value is 255.

### 7.6.2 Constructor & Destructor Documentation

7.6.2.1 `template<typename T > LLGL::Color< T, 3u >::Color ( )` `[inline]`

7.6.2.2 `template<typename T > LLGL::Color< T, 3u >::Color ( const Color< T, 3 > & rhs )` `[inline]`

7.6.2.3 `template<typename T > LLGL::Color< T, 3u >::Color ( const T & scalar )` `[inline]`, `[explicit]`

7.6.2.4 `template<typename T > LLGL::Color< T, 3u >::Color ( const T & r, const T & g, const T & b )` `[inline]`

7.6.2.5 `template<typename T > LLGL::Color< T, 3u >::Color ( Gs::UninitializeTag )` `[inline]`

### 7.6.3 Member Function Documentation

7.6.3.1 `template<typename T > template<typename C > Color<C, 3> LLGL::Color< T, 3u >::Cast ( ) const`  
`[inline]`

Returns a type casted instance of this color.

#### Remarks

All color components will be scaled to the range of the new color type.

## Template Parameters

<i>C</i>	Specifies the static cast type.
----------	---------------------------------

7.6.3.2 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator*=( const Color< T, 3 > & rhs )`  
[inline]

7.6.3.3 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator*=( const T & rhs )` [inline]

7.6.3.4 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator+=( const Color< T, 3 > & rhs )`  
[inline]

7.6.3.5 `template<typename T > Color<T, 3> LLGL::Color< T, 3u >::operator-( ) const` [inline]

7.6.3.6 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator-=( const Color< T, 3 > & rhs )`  
[inline]

7.6.3.7 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator/=( const Color< T, 3 > & rhs )`  
[inline]

7.6.3.8 `template<typename T > Color<T, 3>& LLGL::Color< T, 3u >::operator/=( const T & rhs )` [inline]

7.6.3.9 `template<typename T > T& LLGL::Color< T, 3u >::operator[] ( std::size_t component )` [inline]

Returns the specified color component.

## Parameters

in	<i>component</i>	Specifies the color component index. This must be 0, 1, or 2.
----	------------------	---

7.6.3.10 `template<typename T > const T& LLGL::Color< T, 3u >::operator[] ( std::size_t component ) const`  
[inline]

Returns the specified color component.

## Parameters

in	<i>component</i>	Specifies the color component index. This must be 0, 1, or 2.
----	------------------	---

7.6.3.11 `template<typename T > T* LLGL::Color< T, 3u >::Ptr ( )` [inline]

Returns a pointer to the first element of this color.

7.6.3.12 `template<typename T> const T* LLGL::Color< T, 3u >::Ptr ( ) const` `[inline]`

Returns a constant pointer to the first element of this color.

## 7.6.4 Member Data Documentation

7.6.4.1 `template<typename T> T LLGL::Color< T, 3u >::b`

7.6.4.2 `template<typename T> const std::size_t LLGL::Color< T, 3u >::components = 3` `[static]`

Specifies the number of color components.

7.6.4.3 `template<typename T> T LLGL::Color< T, 3u >::g`

7.6.4.4 `template<typename T> T LLGL::Color< T, 3u >::r`

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

- [ColorRGB.h](#)

## 7.7 LLGL::Color< T, 4u > Class Template Reference

RGBA color class with components: r, g, b, and a.

```
#include <ColorRGBA.h>
```

### Public Member Functions

- [Color](#) ()
- [Color](#) (const [Color](#)< T, 4 > &rhs)
- [Color](#) (const T &brightness)
- [Color](#) (const T &r, const T &g, const T &b)
- [Color](#) (const T &r, const T &g, const T &b, const T &a)
- [Color](#) (Gs::UninitializeTag)
- [Color](#)< T, 4 > & [operator+=](#) (const [Color](#)< T, 4 > &rhs)
- [Color](#)< T, 4 > & [operator-=](#) (const [Color](#)< T, 4 > &rhs)
- [Color](#)< T, 4 > & [operator\\*=](#) (const [Color](#)< T, 4 > &rhs)
- [Color](#)< T, 4 > & [operator/=](#) (const [Color](#)< T, 4 > &rhs)
- [Color](#)< T, 4 > & [operator\\*=](#) (const T &rhs)
- [Color](#)< T, 4 > & [operator/=](#) (const T &rhs)
- [Color](#)< T, 4 > [operator-](#) () const
- T & [operator\[\]](#) (std::size\_t component)  
*Returns the specified color component.*
- const T & [operator\[\]](#) (std::size\_t component) const  
*Returns the specified color component.*
- template<typename C >  
[Color](#)< C, 4 > [Cast](#) () const  
*Returns a type casted instance of this color.*
- T \* [Ptr](#) ()  
*Returns a pointer to the first element of this color.*
- const T \* [Ptr](#) () const  
*Returns a constant pointer to the first element of this color.*

## Public Attributes

- [T r](#)
- [T g](#)
- [T b](#)
- [T a](#)

## Static Public Attributes

- static const std::size\_t [components](#) = 4  
*Specifies the number of color components.*

### 7.7.1 Detailed Description

```
template<typename T>
class LLGL::Color< T, 4u >
```

RGBA color class with components: r, g, b, and a.

#### Remarks

[Color](#) components are default initialized with their maximal value, i.e. for floating-points, the initial value is 1.0, because this its maximal color value, but for unsigned-bytes, the initial value is 255.

### 7.7.2 Constructor & Destructor Documentation

7.7.2.1 `template<typename T > LLGL::Color< T, 4u >::Color ( )` `[inline]`

7.7.2.2 `template<typename T > LLGL::Color< T, 4u >::Color ( const Color< T, 4 > & rhs )` `[inline]`

7.7.2.3 `template<typename T > LLGL::Color< T, 4u >::Color ( const T & brightness )` `[inline]`, `[explicit]`

7.7.2.4 `template<typename T > LLGL::Color< T, 4u >::Color ( const T & r, const T & g, const T & b )` `[inline]`

7.7.2.5 `template<typename T > LLGL::Color< T, 4u >::Color ( const T & r, const T & g, const T & b, const T & a )`  
`[inline]`

7.7.2.6 `template<typename T > LLGL::Color< T, 4u >::Color ( Gs::UninitializeTag )` `[inline]`

### 7.7.3 Member Function Documentation

7.7.3.1 `template<typename T > template<typename C > Color<C, 4> LLGL::Color< T, 4u >::Cast ( )` const  
`[inline]`

Returns a type casted instance of this color.

#### Remarks

All color components will be scaled to the range of the new color type.

## Template Parameters

<i>C</i>	Specifies the static cast type.
----------	---------------------------------

7.7.3.2 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator*=( const Color< T, 4> & rhs )`  
[inline]

7.7.3.3 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator*=( const T & rhs )` [inline]

7.7.3.4 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator+=( const Color< T, 4> & rhs )`  
[inline]

7.7.3.5 `template<typename T> Color<T, 4> LLGL::Color< T, 4u>::operator-( ) const` [inline]

7.7.3.6 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator-=( const Color< T, 4> & rhs )`  
[inline]

7.7.3.7 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator/=( const Color< T, 4> & rhs )`  
[inline]

7.7.3.8 `template<typename T> Color<T, 4>& LLGL::Color< T, 4u>::operator/=( const T & rhs )` [inline]

7.7.3.9 `template<typename T> T& LLGL::Color< T, 4u>::operator[] ( std::size_t component )` [inline]

Returns the specified color component.

## Parameters

in	<i>component</i>	Specifies the color component index. This must be 0, 1, 2, or 3.
----	------------------	--

7.7.3.10 `template<typename T> const T& LLGL::Color< T, 4u>::operator[] ( std::size_t component ) const`  
[inline]

Returns the specified color component.

## Parameters

in	<i>component</i>	Specifies the color component index. This must be 0, 1, 2, or 3.
----	------------------	--

7.7.3.11 `template<typename T> T* LLGL::Color< T, 4u>::Ptr ( )` [inline]

Returns a pointer to the first element of this color.

7.7.3.12 `template<typename T> const T* LLGL::Color< T, 4u >::Ptr ( ) const [inline]`

Returns a constant pointer to the first element of this color.

## 7.7.4 Member Data Documentation

7.7.4.1 `template<typename T> T LLGL::Color< T, 4u >::a`

7.7.4.2 `template<typename T> T LLGL::Color< T, 4u >::b`

7.7.4.3 `template<typename T> const std::size_t LLGL::Color< T, 4u >::components = 4 [static]`

Specifies the number of color components.

7.7.4.4 `template<typename T> T LLGL::Color< T, 4u >::g`

7.7.4.5 `template<typename T> T LLGL::Color< T, 4u >::r`

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

- [ColorRGBA.h](#)

## 7.8 LLGL::ComputePipeline Class Reference

Compute pipeline interface.

```
#include <ComputePipeline.h>
```

### Public Member Functions

- virtual [~ComputePipeline](#) ()

### 7.8.1 Detailed Description

Compute pipeline interface.

### 7.8.2 Constructor & Destructor Documentation

7.8.2.1 `virtual LLGL::ComputePipeline::~~ComputePipeline ( ) [inline], [virtual]`

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

- [ComputePipeline.h](#)

## 7.9 LLGL::ComputePipelineDescriptor Struct Reference

Compute pipeline descriptor structure.

```
#include <ComputePipeline.h>
```

### Public Member Functions

- [ComputePipelineDescriptor](#) ()=default
- [ComputePipelineDescriptor](#) ([ShaderProgram](#) \**shaderProgram*)

### Public Attributes

- [ShaderProgram](#) \* *shaderProgram* = nullptr  
*Pointer to the shader program for the compute pipeline.*

#### 7.9.1 Detailed Description

Compute pipeline descriptor structure.

#### 7.9.2 Constructor & Destructor Documentation

7.9.2.1 [LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor \( \)](#) [default]

7.9.2.2 [LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor \( \[ShaderProgram\]\(#\) \\* \*shaderProgram\* \)](#)  
[inline]

#### 7.9.3 Member Data Documentation

7.9.3.1 [ShaderProgram](#)\* [LLGL::ComputePipelineDescriptor::shaderProgram](#) = nullptr

Pointer to the shader program for the compute pipeline.

#### Remarks

This must never be null when "RenderSystem::CreateComputePipeline" is called with this structure.

#### See also

[RenderSystem::CreateComputePipeline](#)  
[RenderSystem::CreateShaderProgram](#)

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

- [ComputePipeline.h](#)



## 7.10 LLGL::ConstantBuffer Class Reference

Constant buffer (also "Uniform Buffer Object") interface.

```
#include <ConstantBuffer.h>
```

### Public Member Functions

- virtual [~ConstantBuffer](#) ()

#### 7.10.1 Detailed Description

Constant buffer (also "Uniform Buffer Object") interface.

#### 7.10.2 Constructor & Destructor Documentation

7.10.2.1 virtual LLGL::ConstantBuffer::~~ConstantBuffer ( ) `[inline], [virtual]`

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

- [ConstantBuffer.h](#)

## 7.11 LLGL::ConstantBufferDescriptor Struct Reference

Constant buffer descriptor structure.

```
#include <ConstantBuffer.h>
```

### Public Member Functions

- [ConstantBufferDescriptor](#) ()=default
- [ConstantBufferDescriptor](#) (unsigned int [size](#), [BufferUsage](#) [usage](#))

### Public Attributes

- unsigned int [size](#) = 0  
*Buffer size (in bytes).*
- [BufferUsage](#) [usage](#) = [BufferUsage::Dynamic](#)  
*Buffer usage (typically "BufferUsage::Dynamic", since a constant buffer is commonly frequently changed).*

#### 7.11.1 Detailed Description

Constant buffer descriptor structure.

### 7.11.2 Constructor & Destructor Documentation

7.11.2.1 `LLGL::ConstantBufferDescriptor::ConstantBufferDescriptor ( ) [default]`

7.11.2.2 `LLGL::ConstantBufferDescriptor::ConstantBufferDescriptor ( unsigned int size, BufferUsage usage ) [inline]`

### 7.11.3 Member Data Documentation

7.11.3.1 `unsigned int LLGL::ConstantBufferDescriptor::size = 0`

Buffer size (in bytes).

7.11.3.2 `BufferUsage LLGL::ConstantBufferDescriptor::usage = BufferUsage::Dynamic`

Buffer usage (typically "BufferUsage::Dynamic", since a constant buffer is commonly frequently changed).

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

- [ConstantBuffer.h](#)

## 7.12 LLGL::ConstantBufferViewDescriptor Struct Reference

Constant buffer shader-view descriptor structure.

```
#include <ConstantBuffer.h>
```

### Public Attributes

- `std::string name`  
*Constant buffer name.*
- `unsigned int index = 0`  
*Index of the constant buffer within the respective shader.*
- `unsigned int size = 0`  
*Buffer size (in bytes).*

### 7.12.1 Detailed Description

Constant buffer shader-view descriptor structure.

#### Remarks

This structure is used to describe the view of a constant buffer within a shader.

## 7.12.2 Member Data Documentation

### 7.12.2.1 unsigned int LLGL::ConstantBufferViewDescriptor::index = 0

Index of the constant buffer within the respective shader.

### 7.12.2.2 std::string LLGL::ConstantBufferViewDescriptor::name

Constant buffer name.

### 7.12.2.3 unsigned int LLGL::ConstantBufferViewDescriptor::size = 0

Buffer size (in bytes).

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

- [ConstantBuffer.h](#)

## 7.13 LLGL::RenderingProfiler::Counter Class Reference

```
#include <RenderingProfiler.h>
```

### Public Types

- using [ValueType](#) = unsigned int

### Public Member Functions

- void [Inc](#) ()
- void [Inc](#) ([ValueType](#) value)
- void [Reset](#) ()
- [ValueType](#) [Count](#) () const
- [operator unsigned int](#) () const

## 7.13.1 Member Typedef Documentation

### 7.13.1.1 using LLGL::RenderingProfiler::Counter::ValueType = unsigned int

## 7.13.2 Member Function Documentation

### 7.13.2.1 ValueType LLGL::RenderingProfiler::Counter::Count ( ) const [inline]

### 7.13.2.2 void LLGL::RenderingProfiler::Counter::Inc ( ) [inline]

### 7.13.2.3 void LLGL::RenderingProfiler::Counter::Inc ( ValueType value ) [inline]

### 7.13.2.4 LLGL::RenderingProfiler::Counter::operator unsigned int ( ) const [inline]

### 7.13.2.5 void LLGL::RenderingProfiler::Counter::Reset ( ) [inline]

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

- [RenderingProfiler.h](#)

## 7.14 LLGL::DepthDescriptor Struct Reference

Depth state descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

### Public Attributes

- bool `testEnabled` = false  
*Specifies whether the depth test is enabled or disabled. By default disabled.*
- bool `writeEnabled` = false  
*Specifies whether writing to the depth buffer is enabled or disabled. By default disabled.*
- `CompareOp compareOp` = `CompareOp::Less`  
*Specifies the depth test comparison function. By default `CompareOp::Less`.*

### 7.14.1 Detailed Description

Depth state descriptor structure.

### 7.14.2 Member Data Documentation

#### 7.14.2.1 `CompareOp LLGL::DepthDescriptor::compareOp = CompareOp::Less`

Specifies the depth test comparison function. By default `CompareOp::Less`.

#### 7.14.2.2 `bool LLGL::DepthDescriptor::testEnabled = false`

Specifies whether the depth test is enabled or disabled. By default disabled.

#### 7.14.2.3 `bool LLGL::DepthDescriptor::writeEnabled = false`

Specifies whether writing to the depth buffer is enabled or disabled. By default disabled.

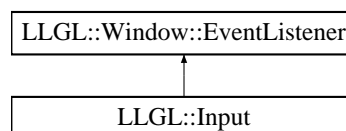
The documentation for this struct was generated from the following file:

- [GraphicsPipelineFlags.h](#)

## 7.15 LLGL::Window::EventListener Class Reference

```
#include <Window.h>
```

Inheritance diagram for `LLGL::Window::EventListener`:



## Public Member Functions

- virtual [~EventListener](#) ()

## Protected Member Functions

- virtual void [OnProcessEvents](#) ([Window](#) &sender)
- virtual void [OnKeyDown](#) ([Window](#) &sender, [Key](#) keyCode)
- virtual void [OnKeyUp](#) ([Window](#) &sender, [Key](#) keyCode)
- virtual void [OnDoubleClick](#) ([Window](#) &sender, [Key](#) keyCode)
- virtual void [OnChar](#) ([Window](#) &sender, [wchar\\_t](#) chr)
- virtual void [OnWheelMotion](#) ([Window](#) &sender, int motion)
- virtual void [OnLocalMotion](#) ([Window](#) &sender, const [Point](#) &position)
- virtual void [OnGlobalMotion](#) ([Window](#) &sender, const [Point](#) &motion)
- virtual void [OnResize](#) ([Window](#) &sender, const [Size](#) &clientAreaSize)
- virtual bool [OnQuit](#) ([Window](#) &sender)

*Returns true if the specified window can quit, i.e. "ProcessEvents" returns false from now on.*

## Friends

- class [Window](#)

## 7.15.1 Constructor & Destructor Documentation

7.15.1.1 virtual LLGL::Window::EventListener::~~EventListener ( ) [virtual]

## 7.15.2 Member Function Documentation

7.15.2.1 virtual void LLGL::Window::EventListener::OnChar ( [Window](#) & *sender*, [wchar\\_t](#) *chr* ) [protected], [virtual]

7.15.2.2 virtual void LLGL::Window::EventListener::OnDoubleClick ( [Window](#) & *sender*, [Key](#) *keyCode* ) [protected], [virtual]

7.15.2.3 virtual void LLGL::Window::EventListener::OnGlobalMotion ( [Window](#) & *sender*, const [Point](#) & *motion* ) [protected], [virtual]

7.15.2.4 virtual void LLGL::Window::EventListener::OnKeyDown ( [Window](#) & *sender*, [Key](#) *keyCode* ) [protected], [virtual]

7.15.2.5 virtual void LLGL::Window::EventListener::OnKeyUp ( [Window](#) & *sender*, [Key](#) *keyCode* ) [protected], [virtual]

7.15.2.6 virtual void LLGL::Window::EventListener::OnLocalMotion ( [Window](#) & *sender*, const [Point](#) & *position* ) [protected], [virtual]

7.15.2.7 virtual void LLGL::Window::EventListener::OnProcessEvents ( [Window](#) & *sender* ) [protected], [virtual]

7.15.2.8 virtual bool LLGL::Window::EventListener::OnQuit ( [Window](#) & *sender* ) [protected], [virtual]

Returns true if the specified window can quit, i.e. "ProcessEvents" returns false from now on.

7.15.2.9 `virtual void LLGL::Window::EventListener::OnResize ( Window & sender, const Size & clientAreaSize )`  
`[protected], [virtual]`

7.15.2.10 `virtual void LLGL::Window::EventListener::OnWheelMotion ( Window & sender, int motion )` `[protected],`  
`[virtual]`

### 7.15.3 Friends And Related Function Documentation

7.15.3.1 `friend class Window` `[friend]`

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

- [Window.h](#)

## 7.16 LLGL::ShaderSource::GLSL Struct Reference

[Shader](#) source descriptor for [GLSL](#).

```
#include <Shader.h>
```

### Public Attributes

- `const std::string & sourceCode`  
[Shader](#) source code string.

### 7.16.1 Detailed Description

[Shader](#) source descriptor for [GLSL](#).

### 7.16.2 Member Data Documentation

7.16.2.1 `const std::string& LLGL::ShaderSource::GLSL::sourceCode`

[Shader](#) source code string.

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

- [Shader.h](#)

## 7.17 LLGL::GraphicsAPIDependentStateDescriptor Union Reference

Low-level graphics API dependent state descriptor union.

```
#include <RenderContextFlags.h>
```

## Classes

- struct [StateOpenGLDescriptor](#)

## Public Member Functions

- [GraphicsAPIDependentStateDescriptor](#) ()

## Public Attributes

- struct [LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor](#) stateOpenGL

### 7.17.1 Detailed Description

Low-level graphics API dependent state descriptor union.

See also

[RenderContext::SetGraphicsAPIDependentState](#)

### 7.17.2 Constructor & Destructor Documentation

7.17.2.1 [LLGL::GraphicsAPIDependentStateDescriptor::GraphicsAPIDependentStateDescriptor \( \)](#) `[inline]`

### 7.17.3 Member Data Documentation

7.17.3.1 [struct LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor](#)  
[LLGL::GraphicsAPIDependentStateDescriptor::stateOpenGL](#)

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

- [RenderContextFlags.h](#)

## 7.18 LLGL::GraphicsPipeline Class Reference

Graphics pipeline interface.

```
#include <GraphicsPipeline.h>
```

## Public Member Functions

- virtual [~GraphicsPipeline](#) ()

### 7.18.1 Detailed Description

Graphics pipeline interface.

### 7.18.2 Constructor & Destructor Documentation

7.18.2.1 `virtual LLGL::GraphicsPipeline::~GraphicsPipeline ( ) [inline], [virtual]`

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

- [GraphicsPipeline.h](#)

## 7.19 LLGL::GraphicsPipelineDescriptor Struct Reference

Graphics pipeline descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

### Public Attributes

- `ShaderProgram * shaderProgram = nullptr`  
*Pointer to the shader program for the graphics pipeline.*
- `PrimitiveTopology primitiveTopology = PrimitiveTopology::TriangleList`  
*Specifies the primitive topology and ordering of the primitive data. By default [PrimitiveTopology::TriangleList](#).*
- `DepthDescriptor depth`  
*Specifies the depth state descriptor.*
- `StencilDescriptor stencil`  
*Specifies the stencil state descriptor.*
- `RasterizerDescriptor rasterizer`  
*Specifies the rasterizer state descriptor.*
- `BlendDescriptor blend`  
*Specifies the blending state descriptor.*

### 7.19.1 Detailed Description

Graphics pipeline descriptor structure.

#### Remarks

This structure describes the entire graphics pipeline: viewports, depth-/ stencil-/ rasterizer-/ blend states, shader stages etc.

### 7.19.2 Member Data Documentation

7.19.2.1 **BlendDescriptor** `LLGL::GraphicsPipelineDescriptor::blend`

Specifies the blending state descriptor.



### 7.19.2.2 DepthDescriptor LLGL::GraphicsPipelineDescriptor::depth

Specifies the depth state descriptor.

### 7.19.2.3 PrimitiveTopology LLGL::GraphicsPipelineDescriptor::primitiveTopology = PrimitiveTopology::TriangleList

Specifies the primitive topology and ordering of the primitive data. By default [PrimitiveTopology::TriangleList](#).

See also

[PrimitiveTopology](#)

### 7.19.2.4 RasterizerDescriptor LLGL::GraphicsPipelineDescriptor::rasterizer

Specifies the rasterizer state descriptor.

### 7.19.2.5 ShaderProgram\* LLGL::GraphicsPipelineDescriptor::shaderProgram = nullptr

Pointer to the shader program for the graphics pipeline.

Remarks

This must never be null when "RenderSystem::CreateGraphicsPipeline" is called with this structure.

See also

[RenderSystem::CreateGraphicsPipeline](#)  
[RenderSystem::CreateShaderProgram](#)

### 7.19.2.6 StencilDescriptor LLGL::GraphicsPipelineDescriptor::stencil

Specifies the stencil state descriptor.

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

- [GraphicsPipelineFlags.h](#)

## 7.20 LLGL::ShaderSource::HLSL Struct Reference

[Shader](#) source descriptor for [HLSL](#).

```
#include <Shader.h>
```

## Public Attributes

- `const std::string & sourceCode`  
*Shader source code string.*
- `std::string entryPoint`  
*Shader entry point (this is the name of the shader main function).*
- `std::string target`  
*Shader version target (see [https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx)).*
- `int flags`  
*Optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries.*

### 7.20.1 Detailed Description

Shader source descriptor for HLSL.

### 7.20.2 Member Data Documentation

#### 7.20.2.1 `std::string LLGL::ShaderSource::HLSL::entryPoint`

Shader entry point (this is the name of the shader main function).

#### 7.20.2.2 `int LLGL::ShaderSource::HLSL::flags`

Optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries.

#### 7.20.2.3 `const std::string& LLGL::ShaderSource::HLSL::sourceCode`

Shader source code string.

#### 7.20.2.4 `std::string LLGL::ShaderSource::HLSL::target`

Shader version target (see [https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx)).

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

- [Shader.h](#)

## 7.21 LLGL::ImageDescriptor Struct Reference

Image descriptor structure.

```
#include <Image.h>
```

## Public Member Functions

- [ImageDescriptor](#) ()=default
- [ImageDescriptor](#) ([ImageFormat](#) format, [DataType](#) dataType, const void \*buffer)
- [ImageDescriptor](#) ([ImageFormat](#) format, const void \*buffer, unsigned int compressedSize)

*Constructor for compressed image data.*

## Public Attributes

- [ImageFormat](#) format = [ImageFormat::RGBA](#)  
*Specifies the image format. By default [ImageFormat::RGBA](#).*
- [DataType](#) dataType = [DataType::UInt8](#)  
*Specifies the image data type. This must be [DataType::UInt8](#) for compressed images.*
- const void \* [buffer](#) = nullptr  
*Pointer to the image buffer.*
- unsigned int [compressedSize](#) = 0  
*Specifies the size (in bytes) of a compressed image. This must be 0 for uncompressed images.*

### 7.21.1 Detailed Description

Image descriptor structure.

#### Remarks

This kind of 'Image' is mainly used to fill the image data of a hardware texture.

### 7.21.2 Constructor & Destructor Documentation

7.21.2.1 `LLGL::ImageDescriptor::ImageDescriptor ( ) [default]`

7.21.2.2 `LLGL::ImageDescriptor::ImageDescriptor ( ImageFormat format, DataType dataType, const void * buffer ) [inline]`

7.21.2.3 `LLGL::ImageDescriptor::ImageDescriptor ( ImageFormat format, const void * buffer, unsigned int compressedSize ) [inline]`

Constructor for compressed image data.

### 7.21.3 Member Data Documentation

7.21.3.1 `const void* LLGL::ImageDescriptor::buffer = nullptr`

Pointer to the image buffer.

7.21.3.2 `unsigned int LLGL::ImageDescriptor::compressedSize = 0`

Specifies the size (in bytes) of a compressed image. This must be 0 for uncompressed images.

### 7.21.3.3 `DataType` `LLGL::ImageDescriptor::dataType = DataType::UInt8`

Specifies the image data type. This must be `DataType::UInt8` for compressed images.

### 7.21.3.4 `ImageFormat` `LLGL::ImageDescriptor::format = ImageFormat::RGBA`

Specifies the image format. By default `ImageFormat::RGBA`.

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

- [Image.h](#)

## 7.22 `LLGL::IndexBuffer` Class Reference

Index buffer interface.

```
#include <IndexBuffer.h>
```

### Public Member Functions

- virtual `~IndexBuffer()`
- const `IndexFormat & GetIndexFormat()` const

### Protected Member Functions

- void `SetIndexFormat` (const `IndexFormat` &indexFormat)

### 7.22.1 Detailed Description

Index buffer interface.

### 7.22.2 Constructor & Destructor Documentation

7.22.2.1 virtual `LLGL::IndexBuffer::~IndexBuffer()` `[inline]`, `[virtual]`

### 7.22.3 Member Function Documentation

7.22.3.1 const `IndexFormat& LLGL::IndexBuffer::GetIndexFormat()` const `[inline]`

7.22.3.2 void `LLGL::IndexBuffer::SetIndexFormat` ( const `IndexFormat` & *indexFormat* ) `[inline]`, `[protected]`

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

- [IndexBuffer.h](#)

## 7.23 LLGL::IndexBufferDescriptor Struct Reference

Index buffer descriptor structure.

```
#include <IndexBuffer.h>
```

### Public Member Functions

- [IndexBufferDescriptor](#) ()=default
- [IndexBufferDescriptor](#) (unsigned int [size](#), [BufferUsage](#) [usage](#), const [IndexFormat](#) &[indexFormat](#))

### Public Attributes

- unsigned int [size](#) = 0  
*Buffer size (in bytes).*
- [BufferUsage](#) [usage](#) = [BufferUsage::Static](#)  
*Buffer usage (typically "BufferUsage::Static", since an index buffer is rarely changed).*
- [IndexFormat](#) [indexFormat](#)  
*Specifies the index format layout, which is basically only the data type of each index.*

#### 7.23.1 Detailed Description

Index buffer descriptor structure.

#### 7.23.2 Constructor & Destructor Documentation

7.23.2.1 LLGL::IndexBufferDescriptor::IndexBufferDescriptor ( ) [default]

7.23.2.2 LLGL::IndexBufferDescriptor::IndexBufferDescriptor ( unsigned int [size](#), [BufferUsage](#) [usage](#), const [IndexFormat](#) & [indexFormat](#) ) [inline]

#### 7.23.3 Member Data Documentation

7.23.3.1 [IndexFormat](#) LLGL::IndexBufferDescriptor::indexFormat

Specifies the index format layout, which is basically only the data type of each index.

#### Remarks

The only valid format types for an index buffer are: [DataType::UByte](#), [DataType::UShort](#), and [DataType::UInt](#).

#### See also

[DataType](#)

### 7.23.3.2 unsigned int LLGL::IndexBufferDescriptor::size = 0

Buffer size (in bytes).

### 7.23.3.3 BufferUsage LLGL::IndexBufferDescriptor::usage = BufferUsage::Static

Buffer usage (typically "BufferUsage::Static", since an index buffer is rarely changed).

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

- [IndexBuffer.h](#)

## 7.24 LLGL::IndexFormat Class Reference

```
#include <IndexFormat.h>
```

### Public Member Functions

- [IndexFormat](#) ()=default
- [IndexFormat](#) (const [DataType](#) dataType)
- [DataType](#) [GetDataType](#) () const  
*Returns the data type of this index format.*
- unsigned int [GetFormatSize](#) () const  
*Returns the size of this vertex format (in bytes).*

### 7.24.1 Constructor & Destructor Documentation

7.24.1.1 `LLGL::IndexFormat::IndexFormat ( ) [default]`

7.24.1.2 `LLGL::IndexFormat::IndexFormat ( const DataType dataType )`

### 7.24.2 Member Function Documentation

7.24.2.1 `DataType LLGL::IndexFormat::GetDataType ( ) const [inline]`

Returns the data type of this index format.

7.24.2.2 `unsigned int LLGL::IndexFormat::GetFormatSize ( ) const [inline]`

Returns the size of this vertex format (in bytes).

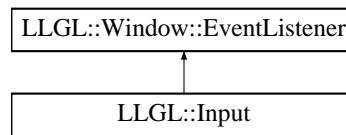
The documentation for this class was generated from the following file:

- [IndexFormat.h](#)

## 7.25 LLGL::Input Class Reference

```
#include <Input.h>
```

Inheritance diagram for LLGL::Input:



### Public Member Functions

- [Input](#) ()
- bool [KeyPressed](#) ([Key](#) keyCode) const  
*Returns true if the specified key is currently being pressed down.*
- bool [KeyDown](#) ([Key](#) keyCode) const  
*Returns true if the specified key was pressed down in the previous event processing.*
- bool [KeyUp](#) ([Key](#) keyCode) const  
*Returns true if the specified key was released in the previous event processing.*
- bool [KeyDoubleClick](#) ([Key](#) keyCode) const  
*Returns true if the specified key was double clicked.*
- const [Point](#) & [GetMousePosition](#) () const  
*Returns the local mouse position.*
- const [Point](#) & [GetMouseMotion](#) () const  
*Returns the global mouse motion.*
- int [GetWheelMotion](#) () const  
*Returns the mouse wheel motion.*
- const std::wstring & [GetEnteredChars](#) () const  
*Returns the entered characters.*

### Additional Inherited Members

#### 7.25.1 Constructor & Destructor Documentation

##### 7.25.1.1 LLGL::Input::Input ( )

#### 7.25.2 Member Function Documentation

##### 7.25.2.1 const std::wstring& LLGL::Input::GetEnteredChars ( ) const [inline]

Returns the entered characters.

##### 7.25.2.2 const Point& LLGL::Input::GetMouseMotion ( ) const [inline]

Returns the global mouse motion.

**7.25.2.3** `const Point& LLGL::Input::GetMousePosition ( ) const` `[inline]`

Returns the local mouse position.

**7.25.2.4** `int LLGL::Input::GetWheelMotion ( ) const` `[inline]`

Returns the mouse wheel motion.

**7.25.2.5** `bool LLGL::Input::KeyDoubleClick ( Key keyCode ) const`

Returns true if the specified key was double clicked.

#### Remarks

This can only be true for the key codes: [Key::LButton](#), [Key::RButton](#), and [Key::MButton](#).

**7.25.2.6** `bool LLGL::Input::KeyDown ( Key keyCode ) const`

Returns true if the specified key was pressed down in the previous event processing.

**7.25.2.7** `bool LLGL::Input::KeyPressed ( Key keyCode ) const`

Returns true if the specified key is currently being pressed down.

**7.25.2.8** `bool LLGL::Input::KeyUp ( Key keyCode ) const`

Returns true if the specified key was released in the previous event processing.

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

- [Input.h](#)

## 7.26 LLGL::RenderingDebugger::Message Class Reference

Rendering debugger message class.

```
#include <RenderingDebugger.h>
```



## Public Member Functions

- [Message](#) ()=default
- [Message](#) (const [Message](#) &)=default
- [Message](#) & [operator=](#) (const [Message](#) &)=default
- [Message](#) (const std::string &text, const std::string &source)
- void [Block](#) ()
- void [BlockAfter](#) (std::size\_t occurrences)
- const std::string & [GetText](#) () const
- const std::string & [GetSource](#) () const
- std::size\_t [GetOccurrences](#) () const
- bool [IsBlocked](#) () const

## Protected Member Functions

- void [IncOccurrence](#) ()

## Friends

- class [RenderingDebugger](#)

### 7.26.1 Detailed Description

Rendering debugger message class.

### 7.26.2 Constructor & Destructor Documentation

7.26.2.1 LLGL::RenderingDebugger::Message::Message ( ) [default]

7.26.2.2 LLGL::RenderingDebugger::Message::Message ( const [Message](#) & ) [default]

7.26.2.3 LLGL::RenderingDebugger::Message::Message ( const std::string & *text*, const std::string & *source* )

### 7.26.3 Member Function Documentation

7.26.3.1 void LLGL::RenderingDebugger::Message::Block ( )

7.26.3.2 void LLGL::RenderingDebugger::Message::BlockAfter ( std::size\_t *occurrences* )

7.26.3.3 std::size\_t LLGL::RenderingDebugger::Message::GetOccurrences ( ) const [inline]

7.26.3.4 const std::string& LLGL::RenderingDebugger::Message::GetSource ( ) const [inline]

7.26.3.5 const std::string& LLGL::RenderingDebugger::Message::GetText ( ) const [inline]

7.26.3.6 void LLGL::RenderingDebugger::Message::IncOccurrence ( ) [protected]

7.26.3.7 bool LLGL::RenderingDebugger::Message::IsBlocked ( ) const [inline]

7.26.3.8 [Message](#)& LLGL::RenderingDebugger::Message::operator= ( const [Message](#) & ) [default]

### 7.26.4 Friends And Related Function Documentation

7.26.4.1 friend class [RenderingDebugger](#) [friend]

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

- [RenderingDebugger.h](#)

## 7.27 LLGL::NativeContextHandle Struct Reference

Linux native context handle structure.

```
#include <LinuxNativeHandle.h>
```

### Public Attributes

- `::Display *` [display](#)
- `::Window` [parentWindow](#)
- `::XVisualInfo *` [visual](#)
- `::Colormap` [colorMap](#)
- `int` [screen](#)
- `NSWindow *` [parentWindow](#)
- `HWND` [parentWindow](#)

### 7.27.1 Detailed Description

Linux native context handle structure.

Win32 native context handle structure.

MacOS native context handle structure.

### 7.27.2 Member Data Documentation

7.27.2.1 `::Colormap` `LLGL::NativeContextHandle::colorMap`

7.27.2.2 `::Display*` `LLGL::NativeContextHandle::display`

7.27.2.3 `HWND` `LLGL::NativeContextHandle::parentWindow`

7.27.2.4 `NSWindow*` `LLGL::NativeContextHandle::parentWindow`

7.27.2.5 `::Window` `LLGL::NativeContextHandle::parentWindow`

7.27.2.6 `int` `LLGL::NativeContextHandle::screen`

7.27.2.7 `::XVisualInfo*` `LLGL::NativeContextHandle::visual`

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

- [LinuxNativeHandle.h](#)
- [MacOSNativeHandle.h](#)
- [Win32NativeHandle.h](#)

## 7.28 LLGL::NativeHandle Struct Reference

Linux native handle structure.

```
#include <LinuxNativeHandle.h>
```

### Public Attributes

- `::Display *` [display](#)
- `::Window` [window](#)
- `::XVisualInfo *` [visual](#)
- `NSWindow *` [window](#)
- `HWND` [window](#)

### 7.28.1 Detailed Description

Linux native handle structure.

Win32 native handle structure.

MacOS native handle structure.

### 7.28.2 Member Data Documentation

**7.28.2.1** `::Display*` `LLGL::NativeHandle::display`

**7.28.2.2** `::XVisualInfo*` `LLGL::NativeHandle::visual`

**7.28.2.3** `NSWindow*` `LLGL::NativeHandle::window`

**7.28.2.4** `HWND` `LLGL::NativeHandle::window`

**7.28.2.5** `::Window` `LLGL::NativeHandle::window`

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

- [LinuxNativeHandle.h](#)
- [MacOSNativeHandle.h](#)
- [Win32NativeHandle.h](#)

## 7.29 LLGL::ProfileOpenGLDescriptor Struct Reference

```
#include <RenderContextDescriptor.h>
```

## Public Attributes

- bool `extProfile` = false  
*Specifies whether an extended renderer profile is to be used. By default false.*
- bool `coreProfile` = false  
*Specifies whether to use 'OpenGL Core Profile', instead of 'OpenGL Compatibility Profile'. By default disabled.*
- bool `debugDump` = false  
*Specifies whether the hardware renderer will produce debug dump. By default disabled.*
- `OpenGLVersion` `version` = `OpenGLVersion::OpenGL_Latest`  
*OpenGL version to create the render context with.*

## 7.29.1 Member Data Documentation

### 7.29.1.1 bool LLGL::ProfileOpenGLDescriptor::coreProfile = false

Specifies whether to use 'OpenGL Core Profile', instead of 'OpenGL Compatibility Profile'. By default disabled.

#### Remarks

This requires 'extProfile' to be enabled.

### 7.29.1.2 bool LLGL::ProfileOpenGLDescriptor::debugDump = false

Specifies whether the hardware renderer will produce debug dump. By default disabled.

### 7.29.1.3 bool LLGL::ProfileOpenGLDescriptor::extProfile = false

Specifies whether an extended renderer profile is to be used. By default false.

### 7.29.1.4 OpenGLVersion LLGL::ProfileOpenGLDescriptor::version = OpenGLVersion::OpenGL\_Latest

OpenGL version to create the render context with.

#### Remarks

This required 'coreProfile' to be enabled.

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

- [RenderContextDescriptor.h](#)

## 7.30 LLGL::Query Class Reference

[Query](#) interface.

```
#include <Query.h>
```

## Public Member Functions

- [Query](#) (const [Query](#) &)=delete
- [Query](#) & [operator=](#) (const [Query](#) &)=delete
- virtual [~Query](#) ()
- [QueryType](#) [GetType](#) () const

*Returns the type of the query.*

## Protected Member Functions

- [Query](#) ([QueryType](#) type)

### 7.30.1 Detailed Description

[Query](#) interface.

### 7.30.2 Constructor & Destructor Documentation

7.30.2.1 `LLGL::Query::Query ( const Query & )` `[delete]`

7.30.2.2 `virtual LLGL::Query::~~Query ( )` `[inline], [virtual]`

7.30.2.3 `LLGL::Query::Query ( QueryType type )` `[inline], [protected]`

### 7.30.3 Member Function Documentation

7.30.3.1 `QueryType LLGL::Query::GetType ( ) const` `[inline]`

Returns the type of the query.

7.30.3.2 `Query& LLGL::Query::operator= ( const Query & )` `[delete]`

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

- [Query.h](#)

## 7.31 LLGL::QueryDescriptor Struct Reference

[Query](#) descriptor structure.

```
#include <Query.h>
```

## Public Attributes

- [QueryType type = QueryType::SamplesPassed](#)  
*Specifies the type of the query. By default [QueryType::SamplesPassed](#) (occlusion query).*
- bool [renderCondition](#) = false  
*Specifies whether the query is to be used as a render condition. By default false.*

### 7.31.1 Detailed Description

[Query](#) descriptor structure.

### 7.31.2 Member Data Documentation

#### 7.31.2.1 bool LLGL::QueryDescriptor::renderCondition = false

Specifies whether the query is to be used as a render condition. By default false.

#### Remarks

If this is true, 'type' can only have one of the following values: [QueryType::SamplesPassed](#), [QueryType::AnySamplesPassed](#), [QueryType::AnySamplesPassedConservative](#), or [QueryType::StreamOutOverflow](#).

#### 7.31.2.2 QueryType LLGL::QueryDescriptor::type = QueryType::SamplesPassed

Specifies the type of the query. By default [QueryType::SamplesPassed](#) (occlusion query).

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

- [Query.h](#)

## 7.32 LLGL::RasterizerDescriptor Struct Reference

Rasterizer state descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

## Public Attributes

- [PolygonMode polygonMode = PolygonMode::Fill](#)  
*Polygon render mode. By default [PolygonMode::Fill](#).*
- [CullMode cullMode = CullMode::Disabled](#)
- int [depthBias](#) = 0
- float [depthBiasClamp](#) = 0.0f
- float [slopeScaledDepthBias](#) = 0.0f
- unsigned int [samples](#) = 1  
*Number of samples for multi-sample anti-aliasing (MSAA).*
- bool [frontCCW](#) = false  
*If true, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding.*
- bool [depthClampEnabled](#) = false
- bool [scissorTestEnabled](#) = false
- bool [multiSampleEnabled](#) = false
- bool [antiAliasedLineEnabled](#) = false
- bool [conservativeRasterization](#) = false  
*If true, conservative rasterization is enabled.*

### 7.32.1 Detailed Description

Rasterizer state descriptor structure.

### 7.32.2 Member Data Documentation

7.32.2.1 `bool LLGL::RasterizerDescriptor::antiAliasedLineEnabled = false`

7.32.2.2 `bool LLGL::RasterizerDescriptor::conservativeRasterization = false`

If true, conservative rasterization is enabled.

#### Note

Only supported with: Direct3D 12 (or OpenGL if the extension "GL\_NV\_conservative\_raster" or "GL\_INTEL\_conservative\_rasterization" is supported).

#### See also

[https://www.opengl.org/registry/specs/NV/conservative\\_raster.txt](https://www.opengl.org/registry/specs/NV/conservative_raster.txt)  
[https://www.opengl.org/registry/specs/INTEL/conservative\\_rasterization.txt](https://www.opengl.org/registry/specs/INTEL/conservative_rasterization.txt)

7.32.2.3 `CullMode LLGL::RasterizerDescriptor::cullMode = CullMode::Disabled`

7.32.2.4 `int LLGL::RasterizerDescriptor::depthBias = 0`

7.32.2.5 `float LLGL::RasterizerDescriptor::depthBiasClamp = 0.0f`

7.32.2.6 `bool LLGL::RasterizerDescriptor::depthClampEnabled = false`

7.32.2.7 `bool LLGL::RasterizerDescriptor::frontCCW = false`

If true, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding.

7.32.2.8 `bool LLGL::RasterizerDescriptor::multiSampleEnabled = false`

7.32.2.9 `PolygonMode LLGL::RasterizerDescriptor::polygonMode = PolygonMode::Fill`

Polygon render mode. By default [PolygonMode::Fill](#).

7.32.2.10 unsigned int LLGL::RasterizerDescriptor::samples = 1

Number of samples for multi-sample anti-aliasing (MSAA).

See also

[multiSampleEnabled](#)

Note

Only supported with: Direct3D 11, Direct3D 12.

7.32.2.11 bool LLGL::RasterizerDescriptor::scissorTestEnabled = false

7.32.2.12 float LLGL::RasterizerDescriptor::slopeScaledDepthBias = 0.0f

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

- [GraphicsPipelineFlags.h](#)

## 7.33 LLGL::RenderContext Class Reference

Render context interface.

```
#include <RenderContext.h>
```

### Public Member Functions

- [RenderContext](#) (const [RenderContext](#) &)=delete
- [RenderContext](#) & operator= (const [RenderContext](#) &)=delete
- virtual [~RenderContext](#) ()
- virtual void [Present](#) ()=0  
*Presents the current frame on the screen.*
- [Window](#) & [GetWindow](#) () const  
*Returns the window which is used to draw all content.*
- virtual void [SetGraphicsAPIDependentState](#) (const [GraphicsAPIDependentStateDescriptor](#) &state)=0  
*Sets a few low-level graphics API dependent states.*
- virtual void [SetVideoMode](#) (const [VideoModeDescriptor](#) &videoModeDesc)  
*Sets the new video mode for this render context.*
- virtual void [SetVsync](#) (const [VsyncDescriptor](#) &vsyncDesc)=0  
*Sets the new vertical-synchronization (Vsync) configuration for this render context.*
- const [VideoModeDescriptor](#) & [GetVideoMode](#) () const  
*Returns the video mode for this render context.*
- virtual void [SetViewport](#) (const [Viewport](#) &viewport)=0  
*Sets a single viewport.*
- virtual void [SetViewportArray](#) (const std::vector< [Viewport](#) > &viewports)=0  
*Sets the array of viewports.*



- virtual void [SetScissor](#) (const [Scissor](#) &scissor)=0  
*Sets a single scissor rectangle.*
- virtual void [SetScissorArray](#) (const std::vector< [Scissor](#) > &scissors)=0  
*Sets the specified scissor rectangles.*
- virtual void [SetClearColor](#) (const [ColorRGBAf](#) &color)=0  
*Sets the new value to clear the color buffer. By default black (0, 0, 0, 0).*
- virtual void [SetClearDepth](#) (float depth)=0  
*Sets the new value to clear the depth buffer with. By default 1.0.*
- virtual void [SetClearStencil](#) (int stencil)=0  
*Sets the new value to clear the stencil buffer. By default 0.*
- virtual void [ClearBuffers](#) (long flags)=0  
*Clears the specified frame buffers.*
- virtual void [SetVertexBuffer](#) ([VertexBuffer](#) &vertexBuffer)=0  
*Sets the active vertex buffer for subsequent drawing operations.*
- virtual void [SetIndexBuffer](#) ([IndexBuffer](#) &indexBuffer)=0  
*Sets the active index buffer for subsequent drawing operations.*
- virtual void [SetConstantBuffer](#) ([ConstantBuffer](#) &constantBuffer, unsigned int slot, long shaderStageFlags=[ShaderStageFlags::AllStages](#))=0  
*Sets the active constant buffer of the specified slot index for subsequent drawing and compute operations.*
- virtual void [SetStorageBuffer](#) ([StorageBuffer](#) &storageBuffer, unsigned int slot)=0  
*Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.*
- virtual void \* [MapStorageBuffer](#) ([StorageBuffer](#) &storageBuffer, const [BufferCPUAccess](#) access)=0  
*Maps the specified storage buffer from GPU to CPU memory space.*
- virtual void [UnmapStorageBuffer](#) ()=0  
*Unmaps the previously mapped storage buffer.*
- virtual void [SetTexture](#) ([Texture](#) &texture, unsigned int slot, long shaderStageFlags=[ShaderStageFlags::AllStages](#))=0  
*Sets the active texture of the specified slot index for subsequent drawing and compute operations.*
- virtual void [SetSampler](#) ([Sampler](#) &sampler, unsigned int slot, long shaderStageFlags=[ShaderStageFlags::AllStages](#))=0  
*Sets the active sampler of the specified slot index for subsequent drawing and compute operations.*
- virtual void [SetRenderTarget](#) ([RenderTarget](#) &renderTarget)=0  
*Sets the active render target.*
- virtual void [UnsetRenderTarget](#) ()=0  
*Unsets the previously set render target.*
- virtual void [SetGraphicsPipeline](#) ([GraphicsPipeline](#) &graphicsPipeline)=0  
*Sets the active graphics pipeline state.*
- virtual void [SetComputePipeline](#) ([ComputePipeline](#) &computePipeline)=0  
*Sets the active compute pipeline state.*
- virtual void [BeginQuery](#) ([Query](#) &query)=0  
*Begins the specified query.*
- virtual void [EndQuery](#) ([Query](#) &query)=0  
*Ends the specified query.*
- virtual bool [QueryResult](#) ([Query](#) &query, std::uint64\_t &result)=0  
*Queries the result of the specified [Query](#) object.*
- virtual void [BeginRenderCondition](#) ([Query](#) &query, const [RenderConditionMode](#) mode)=0  
*Begins conditional rendering with the specified query object.*
- virtual void [EndRenderCondition](#) ()=0  
*Ends the current render condition.*
- virtual void [Draw](#) (unsigned int numVertices, unsigned int firstVertex)=0  
*Draws the specified amount of primitives from the currently set vertex buffer.*

- virtual void [DrawIndexed](#) (unsigned int numVertices, unsigned int firstIndex)=0
- virtual void [DrawIndexed](#) (unsigned int numVertices, unsigned int firstIndex, int vertexOffset)=0  
*Draws the specified amount of primitives from the currently set vertex- and index buffers.*
- virtual void [DrawInstanced](#) (unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances)=0
- virtual void [DrawInstanced](#) (unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances, unsigned int instanceOffset)=0  
*Draws the specified amount of instances of primitives from the currently set vertex buffer.*
- virtual void [DrawIndexedInstanced](#) (unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex)=0
- virtual void [DrawIndexedInstanced](#) (unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset)=0
- virtual void [DrawIndexedInstanced](#) (unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset, unsigned int instanceOffset)=0  
*Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.*
- virtual void [DispatchCompute](#) (const Gs::Vector3ui &threadGroupSize)=0  
*Dispatches a compute command.*
- virtual void [SyncGPU](#) ()=0  
*Synchronizes the GPU, i.e. waits until the GPU has completed all pending commands.*

## Protected Member Functions

- [RenderContext](#) ()=default
- void [SetWindow](#) (const std::shared\_ptr< [Window](#) > &window, [VideoModeDescriptor](#) &videoModeDesc, const void \*windowContext)
- void [ShareWindowAndVideoMode](#) ([RenderContext](#) &other)  
*Shares the window and video mode with another render context.*

### 7.33.1 Detailed Description

Render context interface.

#### Remarks

The render context is the main interface for drawing and compute operations.

### 7.33.2 Constructor & Destructor Documentation

7.33.2.1 `LLGL::RenderContext::RenderContext ( const RenderContext & )` `[delete]`

7.33.2.2 `virtual LLGL::RenderContext::~~RenderContext ( )` `[virtual]`

7.33.2.3 `LLGL::RenderContext::RenderContext ( )` `[protected], [default]`

### 7.33.3 Member Function Documentation

7.33.3.1 `virtual void LLGL::RenderContext::BeginQuery ( Query & query )` `[pure virtual]`

Begins the specified query.

## Parameters

in	<i>query</i>	Specifies the query to begin with. This must be same query object as in the subsequent "EndQuery" function call, to end the query operation.
----	--------------	--

## Remarks

The "BeginQuery" and "EndQuery" functions can be wrapped around any drawing and/or compute operation. This can an occlusion query for instance, which determines how many fragments have passed the depth test.

## See also

[RenderSystem::CreateQuery](#)  
[EndQuery](#)  
[QueryResult](#)

**7.33.3.2** `virtual void LLGL::RenderContext::BeginRenderCondition ( Query & query, const RenderConditionMode mode )`  
 [pure virtual]

Begins conditional rendering with the specified query object.

## Parameters

in	<i>query</i>	Specifies the query object which is to be used as render condition. This must be an occlusion query, i.e. it's type must be either <a href="#">QueryType::SamplesPassed</a> , <a href="#">QueryType::AnySamplesPassed</a> , or <a href="#">QueryType::AnySamplesPassedConservative</a> .
in	<i>mode</i>	Specifies the mode of the render condition.

## Remarks

Here is a usage example:

```
context->BeginQuery(*occlusionQuery);
// draw bounding box ...
context->EndQuery(*occlusionQuery);
context->BeginRenderCondition(*occlusionQuery, LLGL::RenderConditionMode::Wait
);
// draw actual object ...
context->EndRenderCondition();
```

## See also

[QueryType](#)  
[RenderConditionMode](#)

**7.33.3.3** `virtual void LLGL::RenderContext::ClearBuffers ( long flags )` [pure virtual]

Clears the specified frame buffers.

## Parameters

in	<i>flags</i>	Specifies the clear buffer flags. This can be a bitwise OR combination of the "ClearBuffersFlags" enumeration entries.
----	--------------	--

## Remarks

To specify the clear values for each buffer use the respective "SetClear..." function

## See also

[ClearBuffersFlags](#)  
[SetClearColor](#)  
[SetClearDepth](#)  
[SetClearStencil](#)

7.33.3.4 `virtual void LLGL::RenderContext::DispatchCompute ( const Gs::Vector3ui & threadGroupSize )` [pure virtual]

Dispatches a compute command.

## Parameters

in	<i>threadGroupSize</i>	Specifies the number of thread groups, where the number of threads per group is specified statically within the compute shader.
----	------------------------	---

## See also

[SetComputePipeline](#)

7.33.3.5 `virtual void LLGL::RenderContext::Draw ( unsigned int numVertices, unsigned int firstVertex )` [pure virtual]

Draws the specified amount of primitives from the currently set vertex buffer.

## Parameters

in	<i>numVertices</i>	Specifies the number of vertices to generate.
in	<i>firstVertex</i>	Specifies the zero-based offset of the first vertex from the vertex buffer.

7.33.3.6 `virtual void LLGL::RenderContext::DrawIndexed ( unsigned int numVertices, unsigned int firstIndex )` [pure virtual]

## See also

[DrawIndexed\(unsigned int, unsigned int, int\)](#)

7.33.3.7 `virtual void LLGL::RenderContext::DrawIndexed ( unsigned int numVertices, unsigned int firstIndex, int vertexOffset )`  
`[pure virtual]`

Draws the specified amount of primitives from the currently set vertex- and index buffers.

#### Parameters

in	<i>numVertices</i>	Specifies the number of vertices to generate.
in	<i>firstIndex</i>	Specifies the zero-based offset of the first index from the index buffer.
in	<i>vertexOffset</i>	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.

7.33.3.8 `virtual void LLGL::RenderContext::DrawIndexedInstanced ( unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex )`  
`[pure virtual]`

#### See also

[DrawIndexedInstanced\(unsigned int, unsigned int, unsigned int, int, unsigned int\)](#)

7.33.3.9 `virtual void LLGL::RenderContext::DrawIndexedInstanced ( unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset )`  
`[pure virtual]`

#### See also

[DrawIndexedInstanced\(unsigned int, unsigned int, unsigned int, int, unsigned int\)](#)

7.33.3.10 `virtual void LLGL::RenderContext::DrawIndexedInstanced ( unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset, unsigned int instanceOffset )`  
`[pure virtual]`

Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.

#### Parameters

in	<i>numVertices</i>	Specifies the number of vertices to generate.
in	<i>numInstances</i>	Specifies the number of instances to generate.
in	<i>firstIndex</i>	Specifies the zero-based offset of the first index from the index buffer.
in	<i>vertexOffset</i>	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.
in	<i>instanceOffset</i>	Specifies the zero-based instance offset which is added to each instance ID.

7.33.3.11 `virtual void LLGL::RenderContext::DrawInstanced ( unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances )`  
`[pure virtual]`

#### See also

[DrawInstanced\(unsigned int, unsigned int, unsigned int, unsigned int\)](#)

**7.33.3.12** `virtual void LLGL::RenderContext::DrawInstanced ( unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances, unsigned int instanceOffset )` `[pure virtual]`

Draws the specified amount of instances of primitives from the currently set vertex buffer.

#### Parameters

in	<i>numVertices</i>	Specifies the number of vertices to generate.
in	<i>firstVertex</i>	Specifies the zero-based offset of the first vertex from the vertex buffer.
in	<i>numInstances</i>	Specifies the number of instances to generate.
in	<i>instanceOffset</i>	Specifies the zero-based instance offset which is added to each instance ID.

**7.33.3.13** `virtual void LLGL::RenderContext::EndQuery ( Query & query )` `[pure virtual]`

Ends the specified query.

#### See also

[RenderSystem::CreateQuery](#)  
[BeginQuery](#)  
[QueryResult](#)

**7.33.3.14** `virtual void LLGL::RenderContext::EndRenderCondition ( )` `[pure virtual]`

Ends the current render condition.

#### See also

[BeginRenderCondition](#)

**7.33.3.15** `const VideoModeDescriptor& LLGL::RenderContext::GetVideoMode ( ) const` `[inline]`

Returns the video mode for this render context.

**7.33.3.16** `Window& LLGL::RenderContext::GetWindow ( ) const` `[inline]`

Returns the window which is used to draw all content.

**7.33.3.17** `virtual void* LLGL::RenderContext::MapStorageBuffer ( StorageBuffer & storageBuffer, const BufferCPUAccess access )` `[pure virtual]`

Maps the specified storage buffer from GPU to CPU memory space.

## Parameters

in	<i>storageBuffer</i>	Specifies the storage buffer which is to be mapped.
in	<i>access</i>	Specifies the CPU buffer access requirement, i.e. if the CPU can read and/or write the mapped memory.

## Returns

Raw pointer to the mapped memory block. You should be aware of the storage buffer size, to not cause memory violations.

## Exceptions

<i>std::runtime_error</i>	If a storage buffer is already being mapped.
---------------------------	--

## See also

[UnmapStorageBuffer](#)

**7.33.3.18** `RenderContext& LLGL::RenderContext::operator= ( const RenderContext & )` [delete]

**7.33.3.19** `virtual void LLGL::RenderContext::Present ( )` [pure virtual]

Presents the current frame on the screen.

**7.33.3.20** `virtual bool LLGL::RenderContext::QueryResult ( Query & query, std::uint64_t & result )` [pure virtual]

Queries the result of the specified [Query](#) object.

## Parameters

in, out	<i>query</i>	Specifies the <a href="#">Query</a> object whose result is to be queried.
out	<i>result</i>	Specifies the output result.

## Returns

True if the result is available, otherwise false in which case 'result' is not modified.

**7.33.3.21** `virtual void LLGL::RenderContext::SetClearColor ( const ColorRGBAf & color )` [pure virtual]

Sets the new value to clear the color buffer. By default black (0, 0, 0, 0).

**7.33.3.22** `virtual void LLGL::RenderContext::SetClearDepth ( float depth )` [pure virtual]

Sets the new value to clear the depth buffer with. By default 1.0.

7.33.3.23 `virtual void LLGL::RenderContext::SetClearStencil ( int stencil ) [pure virtual]`

Sets the new value to clear the stencil buffer. By default 0.

7.33.3.24 `virtual void LLGL::RenderContext::SetComputePipeline ( ComputePipeline & computePipeline ) [pure virtual]`

Sets the active compute pipeline state.

#### Parameters

in	<i>computePipeline</i>	Specifies the compute pipeline state to set.
----	------------------------	--

#### Remarks

This will set the compute shader states. A valid compute pipeline must always be set before any compute operation can be performed.

#### See also

[RenderSystem::CreateComputePipeline](#)

7.33.3.25 `virtual void LLGL::RenderContext::SetConstantBuffer ( ConstantBuffer & constantBuffer, unsigned int slot, long shaderStageFlags = ShaderStageFlags::AllStages ) [pure virtual]`

Sets the active constant buffer of the specified slot index for subsequent drawing and compute operations.

#### Parameters

in	<i>constantBuffer</i>	Specifies the constant buffer to set. This must not be an unspecified constant buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateConstantBuffer" function or with the "RenderSystem::WriteConstantBuffer" function.
in	<i>slot</i>	Specifies the slot index where to put the constant buffer.
in	<i>shaderStageFlags</i>	Specifies at which shader stages the constant buffer is to be set. By default all shader stages are affected.

#### See also

[RenderSystem::WriteConstantBuffer](#)  
[ShaderStageFlags](#)

7.33.3.26 `virtual void LLGL::RenderContext::SetGraphicsAPIDependentState ( const GraphicsAPIDependentState & Descriptor & state ) [pure virtual]`

Sets a few low-level graphics API dependent states.



**Remarks**

This is mainly used to work around uniform render target behavior between different low-level graphics APIs such as OpenGL and Direct3D.

**7.33.3.27** `virtual void LLGL::RenderContext::SetGraphicsPipeline ( GraphicsPipeline & graphicsPipeline ) [pure virtual]`

Sets the active graphics pipeline state.

**Parameters**

in	<i>graphicsPipeline</i>	Specifies the graphics pipeline state to set.
----	-------------------------	---

**Remarks**

This will set all blending-, rasterizer-, depth-, stencil-, and shader states. A valid graphics pipeline must always be set before any drawing operation can be performed.

**See also**

[RenderSystem::CreateGraphicsPipeline](#)

**7.33.3.28** `virtual void LLGL::RenderContext::SetIndexBuffer ( IndexBuffer & indexBuffer ) [pure virtual]`

Sets the active index buffer for subsequent drawing operations.

**Parameters**

in	<i>indexBuffer</i>	Specifies the index buffer to set. This must not be an unspecified index buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateIndexBuffer" function or with the "RenderSystem::WriteIndexBuffer" function.
----	--------------------	--

**Remarks**

An active index buffer is only required for any "DrawIndexed" or "DrawIndexedInstanced" draw call.

**See also**

[RenderSystem::WriteIndexBuffer](#)

**7.33.3.29** `virtual void LLGL::RenderContext::SetRenderTarget ( RenderTarget & renderTarget ) [pure virtual]`

Sets the active render target.

## Parameters

in	<i>renderTarget</i>	Specifies the render target to set.
----	---------------------	-------------------------------------

## Remarks

Subsequent drawing operations will be rendered into the textures that are attached to the specified render target.

## Note

If the specified render-target has not the same resolution as this render context, the viewports and scissor rectangles may be invalidated!

## See also

[UnsetRenderTarget](#)

7.33.3.30 `virtual void LLGL::RenderContext::SetSampler ( Sampler & sampler, unsigned int slot, long shaderStageFlags = ShaderStageFlags::AllStages ) [pure virtual]`

Sets the active sampler of the specified slot index for subsequent drawing and compute operations.

## Parameters

in	<i>sampler</i>	Specifies the sampler to set.
in	<i>slot</i>	Specifies the slot index where to put the sampler.

## See also

[RenderSystem::CreateSampler](#)

7.33.3.31 `virtual void LLGL::RenderContext::SetScissor ( const Scissor & scissor ) [pure virtual]`

Sets a single scissor rectangle.

## Remarks

Similar to SetScissorArray but only a single scissor rectangle is set.

## See also

[SetScissorArray](#)

7.33.3.32 `virtual void LLGL::RenderContext::SetScissorArray ( const std::vector< Scissor > & scissors ) [pure virtual]`

Sets the specified scissor rectangles.

## Parameters

in	<i>scissors</i>	Specifies the list of scissor rectangles.
----	-----------------	---

## Remarks

This function behaves differently on the OpenGL render system, depending on the state configured with the "SetGraphicsAPIDependentState" function. If 'stateOpenGL.screenSpaceOriginLowerLeft' is false, the origin of each scissor rectangle is on the upper-left (like for all other render systems). If 'stateOpenGL.screenSpaceOriginLowerLeft' is true, the origin of each scissor rectangle is on the lower-left.

## See also

[SetGraphicsAPIDependentState](#)

**7.33.3.33** `virtual void LLGL::RenderContext::SetStorageBuffer ( StorageBuffer & storageBuffer, unsigned int slot )`  
`[pure virtual]`

Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.

## Parameters

in	<i>storageBuffer</i>	Specifies the storage buffer to set. This must not be an unspecified storage buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateStorageBuffer" function or with the "RenderSystem::WriteStorageBuffer" function.
in	<i>slot</i>	Specifies the slot index where to put the storage buffer.

## See also

[RenderSystem::WriteStorageBuffer](#)

**7.33.3.34** `virtual void LLGL::RenderContext::SetTexture ( Texture & texture, unsigned int slot, long shaderStageFlags = ShaderStageFlags::AllStages )`  
`[pure virtual]`

Sets the active texture of the specified slot index for subsequent drawing and compute operations.

## Parameters

in	<i>texture</i>	Specifies the texture to set.
in	<i>slot</i>	Specifies the slot index where to put the texture.

**7.33.3.35** `virtual void LLGL::RenderContext::SetVertexBuffer ( VertexBuffer & vertexBuffer )`  
`[pure virtual]`

Sets the active vertex buffer for subsequent drawing operations.

## Parameters

in	<i>vertexBuffer</i>	Specifies the vertex buffer to set. This must not be an unspecified vertex buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateVertexBuffer" function or with the "RenderSystem::WriteVertexBuffer" function.
----	---------------------	--

## See also

[RenderSystem::WriteVertexBuffer](#)

**7.33.3.36** `virtual void LLGL::RenderContext::SetVideoMode ( const VideoModeDescriptor & videoModeDesc )`  
[virtual]

Sets the new video mode for this render context.

## Remarks

This may invalidate the currently set render target.

## See also

[SetRenderTarget](#)

**7.33.3.37** `virtual void LLGL::RenderContext::SetViewport ( const Viewport & viewport )` [pure virtual]

Sets a single viewport.

## Remarks

Similar to SetViewportArray but only a single viewport is set.

## See also

[SetViewportArray](#)

**7.33.3.38** `virtual void LLGL::RenderContext::SetViewportArray ( const std::vector< Viewport > & viewports )` [pure virtual]

Sets the array of viewports.

## Parameters

in	<i>viewports</i>	Specifies the array of viewports.
----	------------------	-----------------------------------

## Remarks

This function behaves differently on the OpenGL render system, depending on the state configured with the "SetGraphicsAPIDependentState" function. If 'stateOpenGL.screenSpaceOriginLowerLeft' is false, the origin of each viewport is on the upper-left (like for all other render systems). If 'stateOpenGL.screenSpaceOriginLowerLeft' is true, the origin of each viewport is on the lower-left.

## See also

[SetGraphicsAPIDependentState](#)

**7.33.3.39** `virtual void LLGL::RenderContext::SetVsync ( const VsyncDescriptor & vsyncDesc ) [pure virtual]`

Sets the new vertical-synchronization (Vsync) configuration for this render context.

**7.33.3.40** `void LLGL::RenderContext::SetWindow ( const std::shared_ptr< Window > & window, VideoModeDescriptor & videoModeDesc, const void * windowContext ) [protected]`

**7.33.3.41** `void LLGL::RenderContext::ShareWindowAndVideoMode ( RenderContext & other ) [protected]`

Shares the window and video mode with another render context.

## Note

This is only used by the renderer debug layer.

**7.33.3.42** `virtual void LLGL::RenderContext::SyncGPU ( ) [pure virtual]`

Synchronizes the GPU, i.e. waits until the GPU has completed all pending commands.

**7.33.3.43** `virtual void LLGL::RenderContext::UnmapStorageBuffer ( ) [pure virtual]`

Unmaps the previously mapped storage buffer.

## See also

[MapStorageBuffer](#)

**7.33.3.44** `virtual void LLGL::RenderContext::UnsetRenderTarget ( ) [pure virtual]`

Unsets the previously set render target.

## Remarks

Subsequent drawing operations will be rendered into the main framebuffer, which can then be presented onto the screen.

## See also

[SetRenderTarget](#)

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

- [RenderContext.h](#)

## 7.34 LLGL::RenderContextDescriptor Struct Reference

```
#include <RenderContextDescriptor.h>
```

### Public Attributes

- [VsyncDescriptor](#) `vsync`  
*Vertical-synchronization (Vsync) descriptor.*
- [AntiAliasingDescriptor](#) `antiAliasing`  
*Multi-sample anti-aliasing descriptor.*
- [VideoModeDescriptor](#) `videoMode`  
*Video mode descriptor.*
- [ProfileOpenGLDescriptor](#) `profileOpenGL`  
*OpenGL profile descriptor (to switch between compatability or core profile).*
- [DebugCallback](#) `debugCallback`  
*Debugging callback descriptor.*

### 7.34.1 Member Data Documentation

#### 7.34.1.1 [AntiAliasingDescriptor](#) LLGL::RenderContextDescriptor::antiAliasing

Multi-sample anti-aliasing descriptor.

#### 7.34.1.2 [DebugCallback](#) LLGL::RenderContextDescriptor::debugCallback

Debugging callback descriptor.

#### 7.34.1.3 [ProfileOpenGLDescriptor](#) LLGL::RenderContextDescriptor::profileOpenGL

OpenGL profile descriptor (to switch between compatability or core profile).

#### 7.34.1.4 [VideoModeDescriptor](#) LLGL::RenderContextDescriptor::videoMode

Video mode descriptor.

#### 7.34.1.5 [VsyncDescriptor](#) LLGL::RenderContextDescriptor::vsync

Vertical-synchronization (Vsync) descriptor.

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

- [RenderContextDescriptor.h](#)

## 7.35 LLGL::RenderingCaps Struct Reference

Rendering capabilities structure.

```
#include <RenderSystemFlags.h>
```

### Public Attributes

- [ScreenOrigin](#) `screenOrigin` = [ScreenOrigin::UpperLeft](#)  
*Screen coordinate system origin.*
- [ClippingRange](#) `clippingRange` = [ClippingRange::ZeroToOne](#)  
*Clipping depth range.*
- bool [hasGLSL](#) = false  
*Specifies whether GLSL shaders are supported or not.*
- bool [hasHLSL](#) = false  
*Specifies whether HLSL shaders are supported or not.*
- bool [hasRenderTargets](#) = false  
*Specifies whether render targets (also "frame buffer objects") are supported.*
- bool [has3DTextures](#) = false  
*Specifies whether 3D textures are supported.*
- bool [hasCubeTextures](#) = false  
*Specifies whether cube textures are supported.*
- bool [hasTextureArrays](#) = false  
*Specifies whether 1D- and 2D array textures are supported.*
- bool [hasCubeTextureArrays](#) = false  
*Specifies whether cube array textures are supported.*
- bool [hasSamplers](#) = false  
*Specifies whether samplers are supported.*
- bool [hasConstantBuffers](#) = false  
*Specifies whether constant buffers (also "uniform buffer objects") are supported.*
- bool [hasStorageBuffers](#) = false  
*Specifies whether storage buffers (also "read/write buffers") are supported.*
- bool [hasUniforms](#) = false  
*Specifies whether individual shader uniforms are supported (typically only for OpenGL 2.0+).*
- bool [hasGeometryShaders](#) = false  
*Specifies whether geometry shaders are supported.*
- bool [hasTessellationShaders](#) = false  
*Specifies whether tessellation shaders are supported.*
- bool [hasComputeShaders](#) = false  
*Specifies whether compute shaders are supported.*
- bool [hasInstancing](#) = false  
*Specifies whether hardware instancing is supported.*
- bool [hasOffsetInstancing](#) = false  
*Specifies whether hardware instancing with instance offsets is supported.*
- bool [hasViewportArrays](#) = false  
*Specifies whether multiple viewports, depth-ranges, and scissors are supported at once.*
- bool [hasConservativeRasterization](#) = false  
*Specifies whether conservative rasterization is supported.*
- unsigned int [maxNumTextureArrayLayers](#) = 0  
*Specifies maximum number of texture array layers (for 1D-, 2D-, and cube textures).*

- unsigned int `maxNumRenderTargetAttachments` = 0  
*Specifies maximum number of attachment points for each render target.*
- unsigned int `maxConstantBufferSize` = 0  
*Specifies maximum size (in bytes) of each constant buffer.*
- int `maxPatchVertices` = 0  
*Specifies maximum number of patch control points.*
- int `max1DTextureSize` = 0  
*Specifies maximum size of each 1D texture.*
- int `max2DTextureSize` = 0  
*Specifies maximum size of each 2D texture (for width and height).*
- int `max3DTextureSize` = 0  
*Specifies maximum size of each 3D texture (for width, height, and depth).*
- int `maxCubeTextureSize` = 0  
*Specifies maximum size of each cube texture (for width and height).*
- int `maxAnisotropy` = 0  
*Specifies maximum anisotropy texture filter.*
- `Gs::Vector3ui` `maxNumComputeShaderWorkGroups`  
*Specifies maximum number of work groups in a compute shader.*
- `Gs::Vector3ui` `maxComputeShaderWorkGroupSize`  
*Specifies maximum work group size in a compute shader.*

### 7.35.1 Detailed Description

Rendering capabilities structure.

### 7.35.2 Member Data Documentation

#### 7.35.2.1 `ClippingRange` `LLGL::RenderingCaps::clippingRange = ClippingRange::ZeroToOne`

Clipping depth range.

#### 7.35.2.2 `bool` `LLGL::RenderingCaps::has3DTextures = false`

Specifies whether 3D textures are supported.

#### 7.35.2.3 `bool` `LLGL::RenderingCaps::hasComputeShaders = false`

Speciifes whether compute shaders are supported.

#### 7.35.2.4 `bool` `LLGL::RenderingCaps::hasConservativeRasterization = false`

Specifies whether conservative rasterization is supported.



**7.35.2.5 bool LLGL::RenderingCaps::hasConstantBuffers = false**

Specifies whether constant buffers (also "uniform buffer objects") are supported.

**7.35.2.6 bool LLGL::RenderingCaps::hasCubeTextureArrays = false**

Specifies whether cube array textures are supported.

**7.35.2.7 bool LLGL::RenderingCaps::hasCubeTextures = false**

Specifies whether cube textures are supported.

**7.35.2.8 bool LLGL::RenderingCaps::hasGeometryShaders = false**

Specifies whether geometry shaders are supported.

**7.35.2.9 bool LLGL::RenderingCaps::hasGLSL = false**

Specifies whether GLSL shaders are supported or not.

**Note**

Only supported with: OpenGL, Vulkan.

**7.35.2.10 bool LLGL::RenderingCaps::hasHLSL = false**

Specifies whether HLSL shaders are supported or not.

**Note**

Only supported with: Direct3D 11, Direct3D 12.

**7.35.2.11 bool LLGL::RenderingCaps::hasInstancing = false**

Specifies whether hardware instancing is supported.

**7.35.2.12 bool LLGL::RenderingCaps::hasOffsetInstancing = false**

Specifies whether hardware instancing with instance offsets is supported.

**7.35.2.13 bool LLGL::RenderingCaps::hasRenderTargets = false**

Specifies whether render targets (also "frame buffer objects") are supported.

**7.35.2.14**   `bool LLGL::RenderingCaps::hasSamplers = false`

Specifies whether samplers are supported.

**7.35.2.15**   `bool LLGL::RenderingCaps::hasStorageBuffers = false`

Specifies whether storage buffers (also "read/write buffers") are supported.

**7.35.2.16**   `bool LLGL::RenderingCaps::hasTessellationShaders = false`

Specifies whether tessellation shaders are supported.

**7.35.2.17**   `bool LLGL::RenderingCaps::hasTextureArrays = false`

Specifies whether 1D- and 2D array textures are supported.

**7.35.2.18**   `bool LLGL::RenderingCaps::hasUniforms = false`

Specifies whether individual shader uniforms are supported (typically only for OpenGL 2.0+).

**7.35.2.19**   `bool LLGL::RenderingCaps::hasViewportArrays = false`

Specifies whether multiple viewports, depth-ranges, and scissors are supported at once.

**7.35.2.20**   `int LLGL::RenderingCaps::max1DTextureSize = 0`

Specifies maximum size of each 1D texture.

**7.35.2.21**   `int LLGL::RenderingCaps::max2DTextureSize = 0`

Specifies maximum size of each 2D texture (for width and height).

**7.35.2.22**   `int LLGL::RenderingCaps::max3DTextureSize = 0`

Specifies maximum size of each 3D texture (for width, height, and depth).

**7.35.2.23**   `int LLGL::RenderingCaps::maxAnisotropy = 0`

Specifies maximum anisotropy texture filter.

**7.35.2.24 Gs::Vector3ui LLGL::RenderingCaps::maxComputeShaderWorkGroupSize**

Specifies maximum work group size in a compute shader.

**7.35.2.25 unsigned int LLGL::RenderingCaps::maxConstantBufferSize = 0**

Specifies maximum size (in bytes) of each constant buffer.

**7.35.2.26 int LLGL::RenderingCaps::maxCubeTextureSize = 0**

Specifies maximum size of each cube texture (for width and height).

**7.35.2.27 Gs::Vector3ui LLGL::RenderingCaps::maxNumComputeShaderWorkGroups**

Specifies maximum number of work groups in a compute shader.

**7.35.2.28 unsigned int LLGL::RenderingCaps::maxNumRenderTargetAttachments = 0**

Specifies maximum number of attachment points for each render target.

**7.35.2.29 unsigned int LLGL::RenderingCaps::maxNumTextureArrayLayers = 0**

Specifies maximum number of texture array layers (for 1D-, 2D-, and cube textures).

**7.35.2.30 int LLGL::RenderingCaps::maxPatchVertices = 0**

Specifies maximum number of patch control points.

**7.35.2.31 ScreenOrigin LLGL::RenderingCaps::screenOrigin = ScreenOrigin::UpperLeft**

Screen coordinate system origin.

**Remarks**

This determines the coordinate space of viewports, scissors, and framebuffers.

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

- [RenderSystemFlags.h](#)

## 7.36 LLGL::RenderingDebugger Class Reference

Rendering debugger interface.

```
#include <RenderingDebugger.h>
```

### Classes

- class [Message](#)  
*Rendering debugger message class.*

### Public Member Functions

- virtual [~RenderingDebugger](#) ()
- void [PostError](#) ([ErrorType](#) type, const std::string &message, const std::string &source)  
*Posts an error message.*
- void [PostWarning](#) ([WarningType](#) type, const std::string &message, const std::string &source)  
*Posts a warning message.*

### Protected Member Functions

- [RenderingDebugger](#) ()=default
- virtual void [OnError](#) ([ErrorType](#) type, [Message](#) &message)
- virtual void [OnWarning](#) ([WarningType](#) type, [Message](#) &message)

#### 7.36.1 Detailed Description

Rendering debugger interface.

#### Remarks

This can be used to profile the renderer draw calls and buffer updates.

#### 7.36.2 Constructor & Destructor Documentation

7.36.2.1 virtual LLGL::RenderingDebugger::~~RenderingDebugger ( ) [virtual]

7.36.2.2 LLGL::RenderingDebugger::RenderingDebugger ( ) [protected],[default]

#### 7.36.3 Member Function Documentation

7.36.3.1 virtual void LLGL::RenderingDebugger::OnError ( [ErrorType](#) type, [Message](#) & message ) [protected],[virtual]

7.36.3.2 virtual void LLGL::RenderingDebugger::OnWarning ( [WarningType](#) type, [Message](#) & message ) [protected],[virtual]

7.36.3.3 void LLGL::RenderingDebugger::PostError ( [ErrorType](#) type, const std::string & message, const std::string & source )

Posts an error message.

## Parameters

in	<i>type</i>	Specifies the type of error.
in	<i>message</i>	Specifies the string which describes the failure.
in	<i>source</i>	Specifies the string which describes the source (typically the function where the failure happend).

7.36.3.4 void LLGL::RenderingDebugger::PostWarning ( WarningType *type*, const std::string & *message*, const std::string & *source* )

Posts a warning message.

## Parameters

in	<i>type</i>	Specifies the type of error.
in	<i>message</i>	Specifies the string which describes the warning.
in	<i>source</i>	Specifies the string which describes the source (typically the function where the failure happend).

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

- [RenderingDebugger.h](#)

## 7.37 LLGL::RenderingProfiler Class Reference

Rendering profiler model class.

```
#include <RenderingProfiler.h>
```

### Classes

- class [Counter](#)

### Public Member Functions

- void [ResetCounters](#) ()  
*Resets all counters.*
- void [RecordDrawCall](#) (const [PrimitiveTopology](#) topology, [Counter::ValueType](#) numVertices)
- void [RecordDrawCall](#) (const [PrimitiveTopology](#) topology, [Counter::ValueType](#) numVertices, [Counter::ValueType](#) numInstances)

## Public Attributes

- [Counter writeVertexBuffer](#)
- [Counter writeIndexBuffer](#)
- [Counter writeConstantBuffer](#)
- [Counter writeStorageBuffer](#)
- [Counter mapConstantBuffer](#)
- [Counter mapStorageBuffer](#)
- [Counter setVertexBuffer](#)
- [Counter setIndexBuffer](#)
- [Counter setConstantBuffer](#)
- [Counter setStorageBuffer](#)
- [Counter setGraphicsPipeline](#)
- [Counter setComputePipeline](#)
- [Counter setTexture](#)
- [Counter setSampler](#)
- [Counter setRenderTarget](#)
- [Counter drawCalls](#)
- [Counter dispatchComputeCalls](#)
- [Counter renderedPoints](#)
- [Counter renderedLines](#)
- [Counter renderedTriangles](#)
- [Counter renderedPatches](#)

### 7.37.1 Detailed Description

Rendering profiler model class.

#### Remarks

This can be used to profile the renderer draw calls and buffer updates.

### 7.37.2 Member Function Documentation

7.37.2.1 `void LLGL::RenderingProfiler::RecordDrawCall ( const PrimitiveTopology topology, Counter::ValueType numVertices )`

7.37.2.2 `void LLGL::RenderingProfiler::RecordDrawCall ( const PrimitiveTopology topology, Counter::ValueType numVertices, Counter::ValueType numInstances )`

7.37.2.3 `void LLGL::RenderingProfiler::ResetCounters ( )`

Resets all counters.

#### See also

[Counter::Reset](#)

### 7.37.3 Member Data Documentation

7.37.3.1 Counter LLGL::RenderingProfiler::dispatchComputeCalls

7.37.3.2 Counter LLGL::RenderingProfiler::drawCalls

7.37.3.3 Counter LLGL::RenderingProfiler::mapConstantBuffer

7.37.3.4 Counter LLGL::RenderingProfiler::mapStorageBuffer

7.37.3.5 Counter LLGL::RenderingProfiler::renderedLines

7.37.3.6 Counter LLGL::RenderingProfiler::renderedPatches

7.37.3.7 Counter LLGL::RenderingProfiler::renderedPoints

7.37.3.8 Counter LLGL::RenderingProfiler::renderedTriangles

7.37.3.9 Counter LLGL::RenderingProfiler::setComputePipeline

7.37.3.10 Counter LLGL::RenderingProfiler::setConstantBuffer

7.37.3.11 Counter LLGL::RenderingProfiler::setGraphicsPipeline

7.37.3.12 Counter LLGL::RenderingProfiler::setIndexBuffer

7.37.3.13 Counter LLGL::RenderingProfiler::setRenderTarget

7.37.3.14 Counter LLGL::RenderingProfiler::setSampler

7.37.3.15 Counter LLGL::RenderingProfiler::setStorageBuffer

7.37.3.16 Counter LLGL::RenderingProfiler::setTexture

7.37.3.17 Counter LLGL::RenderingProfiler::setVertexBuffer

7.37.3.18 Counter LLGL::RenderingProfiler::writeConstantBuffer

7.37.3.19 Counter LLGL::RenderingProfiler::writeIndexBuffer

7.37.3.20 Counter LLGL::RenderingProfiler::writeStorageBuffer

7.37.3.21 Counter LLGL::RenderingProfiler::writeVertexBuffer

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

- [RenderingProfiler.h](#)

## 7.38 LLGL::RenderSystem Class Reference

Render system interface.

```
#include <RenderSystem.h>
```

### Public Member Functions

- [RenderSystem](#) (const [RenderSystem](#) &)=delete
- [RenderSystem](#) & [operator=](#) (const [RenderSystem](#) &)=delete
- virtual [~RenderSystem](#) ()
- const std::string & [GetName](#) () const  
*Returns the name of this render system.*
- virtual std::map< [RenderInfo](#), std::string > [QueryRenderInfo](#) () const =0  
*Returns all available render information.*
- virtual [RenderingCaps](#) [QueryRenderingCaps](#) () const =0  
*Returns the rendering capabilities.*
- virtual [ShadingLanguage](#) [QueryShadingLanguage](#) () const =0  
*Returns the highest version of the supported shading language.*
- virtual void [SetConfiguration](#) (const [RenderSystemConfiguration](#) &config)  
*Sets the basic configuration.*
- const [RenderSystemConfiguration](#) & [GetConfiguration](#) () const  
*Returns the basic configuration.*
- virtual [RenderContext](#) \* [CreateRenderContext](#) (const [RenderContextDescriptor](#) &desc, const std::shared\_ptr< [Window](#) > &window=nullptr)=0  
*Creates a new render context and returns the raw pointer.*
- virtual void [Release](#) ([RenderContext](#) &renderContext)=0  
*Releases the specified render context. This will all release all resources, that are associated with this render context.*
- bool [MakeCurrent](#) ([RenderContext](#) \*renderContext)  
*Makes the specified render context to the current one.*
- [RenderContext](#) \* [GetCurrentContext](#) () const  
*Returns the current render context. This may also be null.*
- virtual [VertexBuffer](#) \* [CreateVertexBuffer](#) (const [VertexBufferDescriptor](#) &desc, const void \*initialData=nullptr)=0  
*Creates a new vertex buffer.*
- virtual [IndexBuffer](#) \* [CreateIndexBuffer](#) (const [IndexBufferDescriptor](#) &desc, const void \*initialData=nullptr)=0  
*Creates a new index buffer.*
- virtual [ConstantBuffer](#) \* [CreateConstantBuffer](#) (const [ConstantBufferDescriptor](#) &desc, const void \*initialData=nullptr)=0  
*Creates a new buffer (also called "Uniform Buffer Object").*
- virtual [StorageBuffer](#) \* [CreateStorageBuffer](#) (const [StorageBufferDescriptor](#) &desc, const void \*initialData=nullptr)=0  
*Creates a new storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer").*
- virtual void [Release](#) ([VertexBuffer](#) &vertexBuffer)=0
- virtual void [Release](#) ([IndexBuffer](#) &indexBuffer)=0
- virtual void [Release](#) ([ConstantBuffer](#) &constantBuffer)=0
- virtual void [Release](#) ([StorageBuffer](#) &storageBuffer)=0
- virtual void [WriteVertexBuffer](#) ([VertexBuffer](#) &vertexBuffer, const void \*data, std::size\_t dataSize, std::size\_t offset)=0  
*Updates the data of the specified vertex buffer.*



- virtual void [WriteIndexBuffer](#) ([IndexBuffer](#) &indexBuffer, const void \*data, std::size\_t dataSize, std::size\_t offset)=0
- virtual void [WriteConstantBuffer](#) ([ConstantBuffer](#) &constantBuffer, const void \*data, std::size\_t dataSize, std::size\_t offset)=0
- virtual void [WriteStorageBuffer](#) ([StorageBuffer](#) &storageBuffer, const void \*data, std::size\_t dataSize, std::size\_t offset)=0
- virtual [Texture](#) \* [CreateTexture](#) (const [TextureDescriptor](#) &textureDesc, const [ImageDescriptor](#) \*imageDesc=nullptr)=0  
*Creates a new texture.*
- virtual void [Release](#) ([Texture](#) &texture)=0
- virtual [TextureDescriptor](#) [QueryTextureDescriptor](#) (const [Texture](#) &texture)=0  
*Queries a descriptor of the specified texture.*
- virtual void [WriteTexture](#) ([Texture](#) &texture, const [SubTextureDescriptor](#) &subTextureDesc, const [ImageDescriptor](#) &imageDesc)=0  
*Updates the image data of the specified texture.*
- virtual void [ReadTexture](#) (const [Texture](#) &texture, int mipLevel, [ImageFormat](#) imageFormat, [DataType](#) dataType, void \*buffer)=0  
*Reads the image data from the specified texture.*
- virtual void [GenerateMips](#) ([Texture](#) &texture)=0  
*Generates the MIP ("Multum in Parvo") maps for the specified texture.*
- virtual [Sampler](#) \* [CreateSampler](#) (const [SamplerDescriptor](#) &desc)=0  
*Creates a new [Sampler](#) object.*
- virtual void [Release](#) ([Sampler](#) &sampler)=0  
*Releases the specified [Sampler](#) object. After this call, the specified object must no longer be used.*
- virtual [RenderTarget](#) \* [CreateRenderTarget](#) (unsigned int multiSamples=0)=0  
*Creates a new [RenderTarget](#) object with the specified number of samples.*
- virtual void [Release](#) ([RenderTarget](#) &renderTarget)=0  
*Releases the specified [RenderTarget](#) object. After this call, the specified object must no longer be used.*
- virtual [Shader](#) \* [CreateShader](#) (const [ShaderType](#) type)=0  
*Creates a new and empty shader.*
- virtual [ShaderProgram](#) \* [CreateShaderProgram](#) ()=0  
*Creates a new and empty shader program.*
- virtual void [Release](#) ([Shader](#) &shader)=0
- virtual void [Release](#) ([ShaderProgram](#) &shaderProgram)=0
- virtual [GraphicsPipeline](#) \* [CreateGraphicsPipeline](#) (const [GraphicsPipelineDescriptor](#) &desc)=0  
*Creates a new and initialized graphics pipeline state object.*
- virtual [ComputePipeline](#) \* [CreateComputePipeline](#) (const [ComputePipelineDescriptor](#) &desc)=0  
*Creates a new and initialized compute pipeline state object.*
- virtual void [Release](#) ([GraphicsPipeline](#) &graphicsPipeline)=0
- virtual void [Release](#) ([ComputePipeline](#) &computePipeline)=0
- virtual [Query](#) \* [CreateQuery](#) (const [QueryDescriptor](#) &desc)=0  
*Creates a new query.*
- virtual void [Release](#) ([Query](#) &query)=0

## Static Public Member Functions

- static std::vector< std::string > [FindModules](#) ()  
*Returns the list of all available render system modules for the current platform (e.g. on Windows this might be { "OpenGL", "Direct3D12" }, but on MacOS it might be only { "OpenGL" }).*
- static std::shared\_ptr< [RenderSystem](#) > [Load](#) (const std::string &moduleName, [RenderingProfiler](#) \*profiler=nullptr, [RenderingDebugger](#) \*debugger=nullptr)  
*Loads a new render system from the specified module.*

## Protected Member Functions

- [RenderSystem](#) ()=default
- virtual bool [OnMakeCurrent](#) ([RenderContext](#) \*renderContext)  
*Callback when a new render context is about to be made the current one.*
- std::vector< [ColorRGBAub](#) > [GetDefaultTextureImageRGBAub](#) (int numPixels) const  
*Creates an RGBA unsigned-byte image buffer for the specified number of pixels.*

### 7.38.1 Detailed Description

Render system interface.

#### Remarks

This is the main interface for the entire renderer. It manages the ownership of all graphics objects and is used to create, modify, and delete all those objects. The main functions for most graphics objects are "Create...", "Write...", and "Release":

```
// Create and initialize vertex buffer
auto vertexBuffer = renderSystem->CreateVertexBuffer(*vertexBuffer, ...);

// Modify data
renderSystem->WriteVertexBuffer(*vertexBuffer, modificationData, ...);

// Release object
renderSystem->Release(*vertexBuffer);
```

### 7.38.2 Constructor & Destructor Documentation

7.38.2.1 `LLGL::RenderSystem::RenderSystem ( const RenderSystem & )` [delete]

7.38.2.2 `virtual LLGL::RenderSystem::~~RenderSystem ( )` [virtual]

7.38.2.3 `LLGL::RenderSystem::RenderSystem ( )` [protected],[default]

### 7.38.3 Member Function Documentation

7.38.3.1 `virtual ComputePipeline* LLGL::RenderSystem::CreateComputePipeline ( const ComputePipelineDescriptor & desc )` [pure virtual]

Creates a new and initialized compute pipeline state object.

#### Parameters

in	desc	Specifies the compute pipeline descriptor. This will describe the shader states. The "shaderProgram" member of the descriptor must never be null!
----	------	---

#### See also

[ComputePipelineDescriptor](#)

**7.38.3.2** `virtual ConstantBuffer* LLGL::RenderSystem::CreateConstantBuffer ( const ConstantBufferDescriptor & desc, const void * initialData = nullptr ) [pure virtual]`

Creates a new buffer (also called "Uniform Buffer Object").

#### Parameters

in	<i>desc</i>	Specifies the constant buffer descriptor.
in	<i>initialData</i>	Optional raw pointer to the data with which the buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteConstantBuffer" function before it is used for drawing operations.

See also

[WriteConstantBuffer](#)

**7.38.3.3** `virtual GraphicsPipeline* LLGL::RenderSystem::CreateGraphicsPipeline ( const GraphicsPipelineDescriptor & desc ) [pure virtual]`

Creates a new and initialized graphics pipeline state object.

#### Parameters

in	<i>desc</i>	Specifies the graphics pipeline descriptor. This will describe the entire pipeline state, i.e. the blending-, rasterizer-, depth-, stencil- and shader states. The "shaderProgram" member of the descriptor must never be null!
----	-------------	---

See also

[GraphicsPipelineDescriptor](#)

**7.38.3.4** `virtual IndexBuffer* LLGL::RenderSystem::CreateIndexBuffer ( const IndexBufferDescriptor & desc, const void * initialData = nullptr ) [pure virtual]`

Creates a new index buffer.

#### Parameters

in	<i>desc</i>	Specifies the index buffer descriptor.
in	<i>initialData</i>	Optional raw pointer to the data with which the buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteIndexBuffer" function before it is used for drawing operations.

See also

[WriteIndexBuffer](#)

**7.38.3.5** `virtual Query* LLGL::RenderSystem::CreateQuery ( const QueryDescriptor & desc ) [pure virtual]`

Creates a new query.

**7.38.3.6** `virtual RenderContext* LLGL::RenderSystem::CreateRenderContext ( const RenderContextDescriptor & desc, const std::shared_ptr< Window > & window = nullptr ) [pure virtual]`

Creates a new render context and returns the raw pointer.

#### Remarks

The render system takes the ownership of this object. All render contexts are deleted in the destructor of this render system.

**7.38.3.7** `virtual RenderTarget* LLGL::RenderSystem::CreateRenderTarget ( unsigned int multiSamples = 0 ) [pure virtual]`

Creates a new [RenderTarget](#) object with the specified number of samples.

#### Exceptions

<code>std::runtime_error</code>	If the renderer does not support <a href="#">RenderTarget</a> objects (e.g. if OpenGL 2.1 or lower is used).
---------------------------------	--

**7.38.3.8** `virtual Sampler* LLGL::RenderSystem::CreateSampler ( const SamplerDescriptor & desc ) [pure virtual]`

Creates a new [Sampler](#) object.

#### Exceptions

<code>std::runtime_error</code>	If the renderer does not support <a href="#">Sampler</a> objects (e.g. if OpenGL 3.1 or lower is used).
---------------------------------	---

#### See also

`RenderContext::QueryRenderingCaps`

**7.38.3.9** `virtual Shader* LLGL::RenderSystem::CreateShader ( const ShaderType type ) [pure virtual]`

Creates a new and empty shader.

#### Parameters

<code>in</code>	<code>type</code>	Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like.
-----------------	-------------------	---

See also

[Shader](#)

**7.38.3.10** `virtual ShaderProgram* LLGL::RenderSystem::CreateShaderProgram ( ) [pure virtual]`

Creates a new and empty shader program.

Remarks

At least one shader must be attached to a shader program to be used for a graphics or compute pipeline.

See also

[ShaderProgram](#)

**7.38.3.11** `virtual StorageBuffer* LLGL::RenderSystem::CreateStorageBuffer ( const StorageBufferDescriptor & desc, const void * initialData = nullptr ) [pure virtual]`

Creates a new storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer").

Parameters

in	<i>desc</i>	Specifies the storage buffer descriptor.
in	<i>initialData</i>	Optional raw pointer to the data with which the buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteStorageBuffer" function before it is used for drawing operations.

See also

[WriteStorageBuffer](#)

**7.38.3.12** `virtual Texture* LLGL::RenderSystem::CreateTexture ( const TextureDescriptor & textureDesc, const ImageDescriptor * imageDesc = nullptr ) [pure virtual]`

Creates a new texture.

Parameters

in	<i>textureDesc</i>	Specifies the texture descriptor.
in	<i>imageDesc</i>	Optional pointer to the image data descriptor. If this is null, the texture will be initialized with the currently configured default image color. If this is non-null, it is used to initialize the texture data.

Remarks

If the texture type of the descriptor is not an array texture the number of layers will be ignored.

See also

[WriteTexture](#)  
[RenderSystemConfiguration::defaultImageColor](#)

**7.38.3.13** `virtual VertexBuffer* LLGL::RenderSystem::CreateVertexBuffer ( const VertexBufferDescriptor & desc, const void * initialData = nullptr ) [pure virtual]`

Creates a new vertex buffer.

Parameters

in	<i>desc</i>	Specifies the vertex buffer descriptor.
in	<i>initialData</i>	Optional raw pointer to the data with which the buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteVertexBuffer" function before it is used for drawing operations. By default null.

See also

[WriteVertexBuffer](#)

**7.38.3.14** `static std::vector<std::string> LLGL::RenderSystem::FindModules ( ) [static]`

Returns the list of all available render system modules for the current platform (e.g. on Windows this might be { "OpenGL", "Direct3D12" }, but on MacOS it might be only { "OpenGL" }).

**7.38.3.15** `virtual void LLGL::RenderSystem::GenerateMips ( Texture & texture ) [pure virtual]`

Generates the MIP ("Multum in Parvo") maps for the specified texture.

See also

[https://developer.valvesoftware.com/wiki/MIP\\_Mapping](https://developer.valvesoftware.com/wiki/MIP_Mapping)

**7.38.3.16** `const RenderSystemConfiguration& LLGL::RenderSystem::GetConfiguration ( ) const [inline]`

Returns the basic configuration.

See also

[SetConfiguration](#)

**7.38.3.17** `RenderContext* LLGL::RenderSystem::GetCurrentContext ( ) const [inline]`

Returns the current render context. This may also be null.

**7.38.3.18** `std::vector<ColorRGBAub> LLGL::RenderSystem::GetDefaultTextureImageRGBAub ( int numPixels ) const`  
`[protected]`

Creates an RGBA unsigned-byte image buffer for the specified number of pixels.

**7.38.3.19** `const std::string& LLGL::RenderSystem::GetName ( ) const` `[inline]`

Returns the name of this render system.

**7.38.3.20** `static std::shared_ptr<RenderSystem> LLGL::RenderSystem::Load ( const std::string & moduleName,  
 RenderingProfiler * profiler = nullptr, RenderingDebugger * debugger = nullptr )` `[static]`

Loads a new render system from the specified module.

#### Parameters

in	<i>moduleName</i>	Specifies the name from which the new render system is to be loaded. This denotes a dynamic library (*.dll-files on Windows, *.so-files on Unix systems). If compiled in debug mode, the postfix "D" is appended to the module name. Moreover, the platform dependent file extension is always added automatically as well as the prefix "LLGL_", i.e. a module name "OpenGL" will be translated to "LLGL_OpenGLD.dll", if compiled on Windows in Debug mode.
in	<i>profiler</i>	Optional pointer to a rendering profiler. If this is used, the counters of the profiler must be reset manually. This is only supported if LLGL was compiled with the "LLGL_ENABLE_DEBUG_LAYER" flag.
in	<i>debugger</i>	Optional pointer to a rendering debugger. This is only supported if LLGL was compiled with the "LLGL_ENABLE_DEBUG_LAYER" flag.

#### Remarks

Usually the return type is a `std::unique_ptr`, but LLGL needs to keep track of the existence of this render system because only a single instance can be loaded at a time. So a `std::weak_ptr` is stored internally to check if it has been expired (see [http://en.cppreference.com/w/cpp/memory/weak\\_ptr/expired](http://en.cppreference.com/w/cpp/memory/weak_ptr/expired)), and this type can only refer to a `std::shared_ptr`.

#### Exceptions

<i>std::runtime_error</i>	If loading the render system from the specified module failed.
<i>std::runtime_error</i>	If there is already a loaded instance of a render system (make sure there are no more shared pointer references to the previous render system!)

**7.38.3.21** `bool LLGL::RenderSystem::MakeCurrent ( RenderContext * renderContext )`

Makes the specified render context to the current one.

#### Parameters

in	<i>renderContext</i>	Specifies the new current render context. If this is null, no render context is active.
----	----------------------	---

**Returns**

True on success, otherwise false.

**Remarks**

Never draw anything, while no render context is active!

**7.38.3.22** `virtual bool LLGL::RenderSystem::OnMakeCurrent ( RenderContext * renderContext )` [protected],  
[virtual]

Callback when a new render context is about to be made the current one.

**Remarks**

At this point, "GetCurrentContext" returns still the previous render context.

**7.38.3.23** `RenderSystem& LLGL::RenderSystem::operator= ( const RenderSystem & )` [delete]

**7.38.3.24** `virtual std::map<RendererInfo, std::string> LLGL::RenderSystem::QueryRendererInfo ( ) const` [pure  
virtual]

Returns all available renderer information.

**7.38.3.25** `virtual RenderingCaps LLGL::RenderSystem::QueryRenderingCaps ( ) const` [pure virtual]

Returns the rendering capabilities.

**7.38.3.26** `virtual ShadingLanguage LLGL::RenderSystem::QueryShadingLanguage ( ) const` [pure virtual]

Returns the highest version of the supported shading language.

**7.38.3.27** `virtual TextureDescriptor LLGL::RenderSystem::QueryTextureDescriptor ( const Texture & texture )` [pure  
virtual]

Queries a descriptor of the specified texture.

**Remarks**

This can be used to query the type and dimension size of the texture.

**See also**

[TextureDescriptor](#)

**7.38.3.28** `virtual void LLGL::RenderSystem::ReadTexture ( const Texture & texture, int mipLevel, ImageFormat imageFormat, DataType dataType, void* buffer )` [pure virtual]

Reads the image data from the specified texture.



## Parameters

in	<i>texture</i>	Specifies the texture object to read from.
in	<i>mipLevel</i>	Specifies the MIP-level from which to read the image data.
in	<i>imageFormat</i>	Specifies the output image format.
in	<i>dataType</i>	Specifies the output data type.
out	<i>buffer</i>	Specifies the output image buffer. This must be a pointer to a memory block, which is large enough to fit all the image data.

## Remarks

Depending on the image format, data type, and texture size, the output image container must be allocated with enough memory size. The "QueryTextureDescriptor" function can be used to determine the texture dimensions.

```
std::vector<LLGL::ColorRGBAub> image(textureWidth*textureHeight);
renderSystem->ReadTexture(texture, 0, LLGL::ImageFormat::RGBA,
    LLGL::DataType::UInt8, image.data());
```

## See also

[QueryTextureDescriptor](#)

**7.38.3.29** virtual void LLGL::RenderSystem::Release ( **RenderContext & renderContext** ) [pure virtual]

Releases the specified render context. This will all release all resources, that are associated with this render context.

**7.38.3.30** virtual void LLGL::RenderSystem::Release ( **VertexBuffer & vertexBuffer** ) [pure virtual]

**7.38.3.31** virtual void LLGL::RenderSystem::Release ( **IndexBuffer & indexBuffer** ) [pure virtual]

**7.38.3.32** virtual void LLGL::RenderSystem::Release ( **ConstantBuffer & constantBuffer** ) [pure virtual]

**7.38.3.33** virtual void LLGL::RenderSystem::Release ( **StorageBuffer & storageBuffer** ) [pure virtual]

**7.38.3.34** virtual void LLGL::RenderSystem::Release ( **Texture & texture** ) [pure virtual]

**7.38.3.35** virtual void LLGL::RenderSystem::Release ( **Sampler & sampler** ) [pure virtual]

Releases the specified [Sampler](#) object. After this call, the specified object must no longer be used.

**7.38.3.36** virtual void LLGL::RenderSystem::Release ( **RenderTarget & renderTarget** ) [pure virtual]

Releases the specified [RenderTarget](#) object. After this call, the specified object must no longer be used.

- 7.38.3.37 `virtual void LLGL::RenderSystem::Release ( Shader & shader )` [pure virtual]
- 7.38.3.38 `virtual void LLGL::RenderSystem::Release ( ShaderProgram & shaderProgram )` [pure virtual]
- 7.38.3.39 `virtual void LLGL::RenderSystem::Release ( GraphicsPipeline & graphicsPipeline )` [pure virtual]
- 7.38.3.40 `virtual void LLGL::RenderSystem::Release ( ComputePipeline & computePipeline )` [pure virtual]
- 7.38.3.41 `virtual void LLGL::RenderSystem::Release ( Query & query )` [pure virtual]
- 7.38.3.42 `virtual void LLGL::RenderSystem::SetConfiguration ( const RenderSystemConfiguration & config )`  
[virtual]

Sets the basic configuration.

#### Remarks

This can be used to change the behavior of default initialization of textures for instance.

#### See also

[RenderSystemConfiguration](#)

- 7.38.3.43 `virtual void LLGL::RenderSystem::WriteConstantBuffer ( ConstantBuffer & constantBuffer, const void * data, std::size_t dataSize, std::size_t offset )` [pure virtual]

#### See also

[WriteVertexBuffer](#)

- 7.38.3.44 `virtual void LLGL::RenderSystem::WriteIndexBuffer ( IndexBuffer & indexBuffer, const void * data, std::size_t dataSize, std::size_t offset )` [pure virtual]

#### See also

[WriteVertexBuffer](#)

- 7.38.3.45 `virtual void LLGL::RenderSystem::WriteStorageBuffer ( StorageBuffer & storageBuffer, const void * data, std::size_t dataSize, std::size_t offset )` [pure virtual]

#### See also

[WriteVertexBuffer](#)

- 7.38.3.46 `virtual void LLGL::RenderSystem::WriteTexture ( Texture & texture, const SubTextureDescriptor & subTextureDesc, const ImageDescriptor & imageDesc )` [pure virtual]

Updates the image data of the specified texture.

## Parameters

in	<i>texture</i>	Specifies the texture whose data is to be updated.
in	<i>subTextureDesc</i>	Specifies the sub-texture descriptor.
in	<i>imageDesc</i>	Specifies the image data descriptor. Its "data" member must not be null!

7.38.3.47 virtual void LLGL::RenderSystem::WriteVertexBuffer ( VertexBuffer & *vertexBuffer*, const void \* *data*, std::size\_t *dataSize*, std::size\_t *offset* ) [pure virtual]

Updates the data of the specified vertex buffer.

## Parameters

in	<i>vertexBuffer</i>	Specifies the vertex buffer whose data is to be updated.
in	<i>data</i>	Raw pointer to the data with which the vertex buffer is to be updated. This must not be null!
in	<i>dataSize</i>	Specifies the size (in bytes) of the data block which is to be updated. This must be less than or equal to the size of the vertex buffer.
in	<i>offset</i>	Specifies the offset (in bytes) at which the vertex buffer is to be updated. This offset plus the data block size (i.e. 'offset + dataSize') must be less than or equal to the size of the vertex buffer.

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

- [RenderSystem.h](#)

## 7.39 LLGL::RenderSystemConfiguration Struct Reference

Render system configuration structure.

```
#include <RenderSystemFlags.h>
```

## Public Attributes

- [ColorRGBAub defaultImageColor](#) { 0, 0, 0, 0 }  
*Specifies the default color for an uninitialized textures. The default value is black (0, 0, 0, 0).*
- std::size\_t [threadCount](#) = `maxThreadCount`  
*Specifies the number of threads that will be used internally by the render system. By default maxThreadCount.*

### 7.39.1 Detailed Description

Render system configuration structure.

## 7.39.2 Member Data Documentation

### 7.39.2.1 ColorRGBAub LLGL::RenderSystemConfiguration::defaultImageColor { 0, 0, 0, 0 }

Specifies the default color for an uninitialized textures. The default value is black (0, 0, 0, 0).

#### Remarks

This will be used when a texture is created and no initial image data is specified.

### 7.39.2.2 std::size\_t LLGL::RenderSystemConfiguration::threadCount = maxThreadCount

Specifies the number of threads that will be used internally by the render system. By default maxThreadCount.

#### Remarks

This is mainly used by the Direct3D render systems, e.g. inside the "CreateTexture" and "WriteTexture" functions to convert the image data into the respective hardware texture format. OpenGL does this automatically.

#### See also

maxThreadCount

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

- [RenderSystemFlags.h](#)

## 7.40 LLGL::RenderTarget Class Reference

Render target interface.

```
#include <RenderTarget.h>
```

### Public Member Functions

- virtual [~RenderTarget](#) ()
- virtual void [AttachDepthBuffer](#) (const Gs::Vector2i &size)=0  
*Attaches an internal depth buffer to this render target.*
- virtual void [AttachStencilBuffer](#) (const Gs::Vector2i &size)=0  
*Attaches an internal stencil buffer to this render target.*
- virtual void [AttachDepthStencilBuffer](#) (const Gs::Vector2i &size)=0  
*Attaches an internal depth-stencil buffer to this render target.*
- virtual void [AttachTexture1D](#) ([Texture](#) &texture, int mipLevel=0)=0
- virtual void [AttachTexture2D](#) ([Texture](#) &texture, int mipLevel=0)=0
- virtual void [AttachTexture3D](#) ([Texture](#) &texture, int layer, int mipLevel=0)=0
- virtual void [AttachTextureCube](#) ([Texture](#) &texture, const [AxisDirection](#) cubeFace, int mipLevel=0)=0
- virtual void [AttachTexture1DArray](#) ([Texture](#) &texture, int layer, int mipLevel=0)=0
- virtual void [AttachTexture2DArray](#) ([Texture](#) &texture, int layer, int mipLevel=0)=0
- virtual void [AttachTextureCubeArray](#) ([Texture](#) &texture, int layer, const [AxisDirection](#) cubeFace, int mipLevel=0)=0
- virtual void [DetachTextures](#) ()=0  
*Detaches all textures from this render target.*
- const Gs::Vector2i & [GetResolution](#) () const  
*Returns the frame buffer resolution.*

## Protected Member Functions

- void [ApplyResolution](#) (const Gs::Vector2i &resolution)
- void [ApplyMipResolution](#) ([Texture](#) &texture, int mipLevel)
- void [ResetResolution](#) ()

### 7.40.1 Detailed Description

Render target interface.

#### Remarks

A render target in the broader sense is a composition of [Texture](#) objects which can be specified as the destination for drawing operations. After a texture has been attached to a render target, its image content is undefined until something has been rendered into the render target.

### 7.40.2 Constructor & Destructor Documentation

7.40.2.1 virtual LLGL::RenderTarget::~RenderTarget ( ) [virtual]

### 7.40.3 Member Function Documentation

7.40.3.1 void LLGL::RenderTarget::ApplyMipResolution ( [Texture](#) & *texture*, int *mipLevel* ) [protected]

7.40.3.2 void LLGL::RenderTarget::ApplyResolution ( const Gs::Vector2i & *resolution* ) [protected]

7.40.3.3 virtual void LLGL::RenderTarget::AttachDepthBuffer ( const Gs::Vector2i & *size* ) [pure virtual]

Attaches an internal depth buffer to this render target.

#### Parameters

<i>in</i>	<i>size</i>	Specifies the size of the depth buffer. This must be the same as for all other attachemnts.
-----------	-------------	---

#### Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

#### See also

[AttachDepthStencilBuffer](#)

7.40.3.4 virtual void LLGL::RenderTarget::AttachDepthStencilBuffer ( const Gs::Vector2i & *size* ) [pure virtual]

Attaches an internal depth-stencil buffer to this render target.

## Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

## See also

[AttachDepthBuffer](#)

7.40.3.5 `virtual void LLGL::RenderTarget::AttachStencilBuffer ( const Gs::Vector2i & size )` [pure virtual]

Attaches an internal stencil buffer to this render target.

## Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

## See also

[AttachDepthBuffer](#)

7.40.3.6 `virtual void LLGL::RenderTarget::AttachTexture1D ( Texture & texture, int mipLevel = 0 )` [pure virtual]

7.40.3.7 `virtual void LLGL::RenderTarget::AttachTexture1DArray ( Texture & texture, int layer, int mipLevel = 0 )` [pure virtual]

7.40.3.8 `virtual void LLGL::RenderTarget::AttachTexture2D ( Texture & texture, int mipLevel = 0 )` [pure virtual]

7.40.3.9 `virtual void LLGL::RenderTarget::AttachTexture2DArray ( Texture & texture, int layer, int mipLevel = 0 )` [pure virtual]

7.40.3.10 `virtual void LLGL::RenderTarget::AttachTexture3D ( Texture & texture, int layer, int mipLevel = 0 )` [pure virtual]

7.40.3.11 `virtual void LLGL::RenderTarget::AttachTextureCube ( Texture & texture, const AxisDirection cubeFace, int mipLevel = 0 )` [pure virtual]

7.40.3.12 `virtual void LLGL::RenderTarget::AttachTextureCubeArray ( Texture & texture, int layer, const AxisDirection cubeFace, int mipLevel = 0 )` [pure virtual]

7.40.3.13 `virtual void LLGL::RenderTarget::DetachTextures ( )` [pure virtual]

Detaches all textures from this render target.

7.40.3.14 `const Gs::Vector2i& LLGL::RenderTarget::GetResolution ( ) const` `[inline]`

Returns the frame buffer resolution.

#### Remarks

This will be determined by the first texture attachment. Every further attachment must have the same size.

7.40.3.15 `void LLGL::RenderTarget::ResetResolution ( )` `[protected]`

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

- [RenderTarget.h](#)

## 7.41 LLGL::Sampler Class Reference

[Sampler](#) interface.

```
#include <Sampler.h>
```

### Public Member Functions

- virtual [~Sampler](#) ()

#### 7.41.1 Detailed Description

[Sampler](#) interface.

#### 7.41.2 Constructor & Destructor Documentation

7.41.2.1 `virtual LLGL::Sampler::~~Sampler ( )` `[inline],[virtual]`

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

- [Sampler.h](#)

## 7.42 LLGL::SamplerDescriptor Struct Reference

[Texture](#) sampler descriptor structure.

```
#include <SamplerFlags.h>
```

## Public Attributes

- `TextureWrap textureWrapU = TextureWrap::Repeat`  
*Texture coordinate wrap mode in U direction. By default `TextureWrap::Repeat`.*
- `TextureWrap textureWrapV = TextureWrap::Repeat`  
*Texture coordinate wrap mode in V direction. By default `TextureWrap::Repeat`.*
- `TextureWrap textureWrapW = TextureWrap::Repeat`  
*Texture coordinate wrap mode in W direction. By default `TextureWrap::Repeat`.*
- `TextureFilter minFilter = TextureFilter::Linear`  
*Minification filter. By default `TextureFilter::Linear`.*
- `TextureFilter magFilter = TextureFilter::Linear`  
*Magnification filter. By default `TextureFilter::Linear`.*
- `TextureFilter mipMapFilter = TextureFilter::Linear`  
*MIP-mapping filter. By default `TextureFilter::Linear`.*
- `bool mipMapping = true`  
*Specifies whether MIP-maps are used or not. By default true.*
- `float mipMapLODBias = 0.0f`  
*MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.*
- `float minLOD = 0.0f`  
*Lower end of the MIP-map range. By default 0.*
- `float maxLOD = 1000.0f`  
*Upper end of the MIP-map range. Must be greater than or equal to "minLOD". By default 1000.*
- `unsigned int maxAnisotropy = 1`  
*Maximal anisotropy in the range [1, 16].*
- `bool depthCompare = false`  
*Specifies whether the compare operation for depth textures is to be used or not. By default false.*
- `CompareOp compareOp = CompareOp::Less`  
*Compare operation for depth textures. By default `CompareOp::Less`.*
- `ColorRGBAf borderColor = { 0.0f, 0.0f, 0.0f, 0.0f }`  
*Border color. By default black (0, 0, 0, 0).*

### 7.42.1 Detailed Description

`Texture` sampler descriptor structure.

### 7.42.2 Member Data Documentation

#### 7.42.2.1 `ColorRGBAf LLGL::SamplerDescriptor::borderColor = { 0.0f, 0.0f, 0.0f, 0.0f }`

Border color. By default black (0, 0, 0, 0).

#### 7.42.2.2 `CompareOp LLGL::SamplerDescriptor::compareOp = CompareOp::Less`

Compare operation for depth textures. By default `CompareOp::Less`.



**7.42.2.3 bool LLGL::SamplerDescriptor::depthCompare = false**

Specifies whether the compare operation for depth textures is to be used or not. By default false.

**7.42.2.4 TextureFilter LLGL::SamplerDescriptor::magFilter = TextureFilter::Linear**

Magnification filter. By default [TextureFilter::Linear](#).

**7.42.2.5 unsigned int LLGL::SamplerDescriptor::maxAnisotropy = 1**

Maximal anisotropy in the range [1, 16].

**7.42.2.6 float LLGL::SamplerDescriptor::maxLOD = 1000.0f**

Upper end of the MIP-map range. Must be greater than or equal to "minLOD". By default 1000.

**7.42.2.7 TextureFilter LLGL::SamplerDescriptor::minFilter = TextureFilter::Linear**

Minification filter. By default [TextureFilter::Linear](#).

**7.42.2.8 float LLGL::SamplerDescriptor::minLOD = 0.0f**

Lower end of the MIP-map range. By default 0.

**7.42.2.9 TextureFilter LLGL::SamplerDescriptor::mipMapFilter = TextureFilter::Linear**

MIP-mapping filter. By default [TextureFilter::Linear](#).

**7.42.2.10 float LLGL::SamplerDescriptor::mipMapLODBias = 0.0f**

MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.

**7.42.2.11 bool LLGL::SamplerDescriptor::mipMapping = true**

Specifies whether MIP-maps are used or not. By default true.

**7.42.2.12 TextureWrap LLGL::SamplerDescriptor::textureWrapU = TextureWrap::Repeat**

[Texture](#) coordinate wrap mode in U direction. By default [TextureWrap::Repeat](#).

#### 7.42.2.13 TextureWrap LLGL::SamplerDescriptor::textureWrapV = TextureWrap::Repeat

Texture coordinate wrap mode in V direction. By default [TextureWrap::Repeat](#).

#### 7.42.2.14 TextureWrap LLGL::SamplerDescriptor::textureWrapW = TextureWrap::Repeat

Texture coordinate wrap mode in W direction. By default [TextureWrap::Repeat](#).

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

- [SamplerFlags.h](#)

## 7.43 LLGL::Scissor Struct Reference

[Scissor](#) dimensions.

```
#include <RenderContextFlags.h>
```

### Public Member Functions

- [Scissor](#) ()=default
- [Scissor](#) (const [Scissor](#) &)=default
- [Scissor](#) (int [x](#), int [y](#), int [width](#), int [height](#))

### Public Attributes

- int [x](#) = 0
- int [y](#) = 0
- int [width](#) = 0
- int [height](#) = 0

#### 7.43.1 Detailed Description

[Scissor](#) dimensions.

#### Remarks

A scissor is in screen coordinates where the origin is in the left-top corner.

## 7.43.2 Constructor & Destructor Documentation

7.43.2.1 LLGL::Scissor::Scissor ( ) [default]

7.43.2.2 LLGL::Scissor::Scissor ( const Scissor & ) [default]

7.43.2.3 LLGL::Scissor::Scissor ( int x, int y, int width, int height ) [inline]

## 7.43.3 Member Data Documentation

7.43.3.1 int LLGL::Scissor::height = 0

7.43.3.2 int LLGL::Scissor::width = 0

7.43.3.3 int LLGL::Scissor::x = 0

7.43.3.4 int LLGL::Scissor::y = 0

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

- [RenderContextFlags.h](#)

## 7.44 LLGL::Shader Class Reference

[Shader](#) interface.

```
#include <Shader.h>
```

### Public Member Functions

- virtual [~Shader](#) ()
- virtual bool [Compile](#) (const [ShaderSource](#) &shaderSource)=0  
*Compiles the specified shader source.*
- virtual std::string [Disassemble](#) (int flags=0)  
*Disassembles the previously compiled shader byte code.*
- virtual std::string [QueryInfoLog](#) ()=0  
*Returns the information log after the shader compilation.*
- [ShaderType GetType](#) () const  
*Returns the type of this shader.*

### Protected Member Functions

- [Shader](#) (const [ShaderType](#) type)

### 7.44.1 Detailed Description

[Shader](#) interface.

### 7.44.2 Constructor & Destructor Documentation

7.44.2.1 `virtual LLGL::Shader::~Shader ( ) [virtual]`

7.44.2.2 `LLGL::Shader::Shader ( const ShaderType type ) [protected]`

### 7.44.3 Member Function Documentation

7.44.3.1 `virtual bool LLGL::Shader::Compile ( const ShaderSource & shaderSource ) [pure virtual]`

Compiles the specified shader source.

#### Parameters

in	<i>shaderSource</i>	Specifies the shader source code.
----	---------------------	-----------------------------------

#### Returns

True on success, otherwise "QueryInfoLog" can be used to query the reason for failure.

#### See also

[QueryInfoLog](#)

7.44.3.2 `virtual std::string LLGL::Shader::Disassemble ( int flags = 0 ) [virtual]`

Disassembles the previously compiled shader byte code.

#### Parameters

in	<i>flags</i>	Specifies optional disassemble flags. This can be a bitwise OR combination of the ' <a href="#">ShaderDisassembleFlags</a> ' enumeration entries. By default 0.
----	--------------	---

#### Returns

Disassembled assembler code or an empty string if disassembling was not possible.

#### Note

Only supported with: Direct3D 11, Direct3D 12 (for HLSL).

## 7.44.3.3 ShaderType LLGL::Shader::GetType ( ) const [inline]

Returns the type of this shader.

## 7.44.3.4 virtual std::string LLGL::Shader::QueryInfoLog ( ) [pure virtual]

Returns the information log after the shader compilation.

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

- [Shader.h](#)

## 7.45 LLGL::ShaderCompileFlags Struct Reference

[Shader](#) compilation flags enumeration.

```
#include <Shader.h>
```

## Public Types

- enum {  
[Debug](#) = (1 << 0), [O1](#) = (1 << 1), [O2](#) = (1 << 2), [O3](#) = (1 << 3),  
[WarnError](#) = (1 << 4) }

## 7.45.1 Detailed Description

[Shader](#) compilation flags enumeration.

## 7.45.2 Member Enumeration Documentation

## 7.45.2.1 anonymous enum

## Enumerator

- Debug*** Insert debug information.
- O1*** Optimization level 1.
- O2*** Optimization level 2.
- O3*** Optimization level 3.
- WarnError*** Warnings are treated as errors.

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

- [Shader.h](#)

## 7.46 LLGL::ShaderDisassembleFlags Struct Reference

[Shader](#) disassemble flags enumeration.

```
#include <Shader.h>
```

### Public Types

- enum { [InstructionOnly](#) = (1 << 0) }

#### 7.46.1 Detailed Description

[Shader](#) disassemble flags enumeration.

#### 7.46.2 Member Enumeration Documentation

##### 7.46.2.1 anonymous enum

##### Enumerator

***InstructionOnly*** Show only instructions in disassembly output.

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

- [Shader.h](#)

## 7.47 LLGL::ShaderProgram Class Reference

[Shader](#) program interface.

```
#include <ShaderProgram.h>
```

## Public Member Functions

- virtual [~ShaderProgram](#) ()
- virtual void [AttachShader](#) ([Shader](#) &shader)=0  
*Attaches the specified shader to this shader program.*
- virtual bool [LinkShaders](#) ()=0  
*Links all attached shaders to the final shader program.*
- virtual std::string [QueryInfoLog](#) ()=0  
*Returns the information log after the shader linkage.*
- virtual std::vector< [VertexAttribute](#) > [QueryVertexAttributes](#) () const =0  
*Returns a list of vertex attributes, which describe all vertex attributes within this shader program.*
- virtual std::vector< [ConstantBufferViewDescriptor](#) > [QueryConstantBuffers](#) () const =0  
*Returns a list of constant buffer view descriptors, which describe all constant buffers within this shader program.*
- virtual std::vector< [StorageBufferViewDescriptor](#) > [QueryStorageBuffers](#) () const =0  
*Returns a list of storage buffer view descriptors, which describe all storage buffers within this shader program.*
- virtual std::vector< [UniformDescriptor](#) > [QueryUniforms](#) () const =0  
*Returns a list of uniform descriptors, which describe all uniforms within this shader program.*
- virtual void [BindVertexAttributes](#) (const std::vector< [VertexAttribute](#) > &vertexAttribs)=0  
*Binds the specified vertex attributes to this shader program.*
- virtual void [BindConstantBuffer](#) (const std::string &name, unsigned int bindingIndex)=0  
*Binds the specified constant buffer to this shader.*
- virtual void [BindStorageBuffer](#) (const std::string &name, unsigned int bindingIndex)=0  
*Binds the specified storage buffer to this shader.*
- virtual [ShaderUniform](#) \* [LockShaderUniform](#) ()=0  
*Locks the shader uniform handler.*
- virtual void [UnlockShaderUniform](#) ()=0  
*Unlocks the shader uniform handler.*

### 7.47.1 Detailed Description

[Shader](#) program interface.

### 7.47.2 Constructor & Destructor Documentation

7.47.2.1 virtual LLGL::ShaderProgram::~~ShaderProgram ( ) [inline], [virtual]

### 7.47.3 Member Function Documentation

7.47.3.1 virtual void LLGL::ShaderProgram::AttachShader ( [Shader](#) & *shader* ) [pure virtual]

Attaches the specified shader to this shader program.

#### Parameters

in	<i>shader</i>	Specifies the shader which is to be attached to this shader program. Each shader type can only be added once for each shader program.
----	---------------	---

**Remarks**

This must be called, before "LinkShaders" is called.

**Exceptions**

<code>std::invalid_argument</code>	If a shader is attached to this shader program, which is not allowed in the current state. This will happen if a different shader of the same type has already been attached to this shader program for instance.
------------------------------------	---

**See also**

[Shader::GetType](#)

**7.47.3.2** `virtual void LLGL::ShaderProgram::BindConstantBuffer ( const std::string & name, unsigned int bindingIndex )`  
[pure virtual]

Binds the specified constant buffer to this shader.

**Parameters**

in	<i>name</i>	Specifies the name of the constant buffer within this shader.
in	<i>bindingIndex</i>	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindConstantBuffer".

**Remarks**

This function is only necessary if the binding index does not match the default binding index of the constant buffer within the shader.

**See also**

[QueryConstantBuffers](#)

`RenderContext::BindConstantBuffer`

**7.47.3.3** `virtual void LLGL::ShaderProgram::BindStorageBuffer ( const std::string & name, unsigned int bindingIndex )`  
[pure virtual]

Binds the specified storage buffer to this shader.

**Parameters**

in	<i>name</i>	Specifies the name of the storage buffer within this shader.
in	<i>bindingIndex</i>	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindStorageBuffer".



**Remarks**

This function is only necessary if the binding index does not match the default binding index of the storage buffer within the shader.

**See also**

RenderContext::BindStorageBuffer

**7.47.3.4** `virtual void LLGL::ShaderProgram::BindVertexAttributes ( const std::vector< VertexAttribute > & vertexAttribs )`  
`[pure virtual]`

Binds the specified vertex attributes to this shader program.

**Parameters**

in	<i>vertexAttribs</i>	Specifies the vertex attributes.
----	----------------------	----------------------------------

**Remarks**

This is only required for a shader program, which has an attached vertex shader. Moreover, this can only be called after shader compilation but before shader program linking!

**See also**

AttachShader(VertexShader&  
[Shader::Compile](#)  
[LinkShaders](#)

**Exceptions**

<i>std::invalid_argument</i>	If the name of an vertex attribute is invalid or the maximal number of available vertex attributes is exceeded.
------------------------------	---

**7.47.3.5** `virtual bool LLGL::ShaderProgram::LinkShaders ( )` `[pure virtual]`

Links all attached shaders to the final shader program.

**Returns**

True on success, otherwise "QueryInfoLog" can be used to query the reason for failure.

**Remarks**

Each attached shader must be compiled first!

**See also**

[QueryInfoLog](#)

#### 7.47.3.6 virtual ShaderUniform\* LLGL::ShaderProgram::LockShaderUniform ( ) [pure virtual]

Locks the shader uniform handler.

##### Returns

Pointer to the shader uniform handler or null if the render system does not support individual shader uniforms.

##### Remarks

This must be called to set individual shader uniforms.

```
auto uniform = shaderProgram->LockShaderUniform();
if (uniform)
{
    uniform->SetUniform("mySampler1", 0);
    uniform->SetUniform("mySampler2", 1);
    uniform->SetUniform("projection", myProjectionMatrix);
}
shaderProgram->UnlockShaderUniform();
```

##### Note

Only a shader program from an OpenGL render system will return a non-null pointer!

#### 7.47.3.7 virtual std::vector<ConstantBufferViewDescriptor> LLGL::ShaderProgram::QueryConstantBuffers ( ) const [pure virtual]

Returns a list of constant buffer view descriptors, which describe all constant buffers within this shader program.

##### Remarks

Also called "Uniform Buffer Object".

#### 7.47.3.8 virtual std::string LLGL::ShaderProgram::QueryInfoLog ( ) [pure virtual]

Returns the information log after the shader linkage.

#### 7.47.3.9 virtual std::vector<StorageBufferViewDescriptor> LLGL::ShaderProgram::QueryStorageBuffers ( ) const [pure virtual]

Returns a list of storage buffer view descriptors, which describe all storage buffers within this shader program.

##### Remarks

Also called "Shader Storage Buffer Object" or "Read/Write Buffer".

7.47.3.10 `virtual std::vector<UniformDescriptor> LLGL::ShaderProgram::QueryUniforms ( ) const` [pure virtual]

Returns a list of uniform descriptors, which describe all uniforms within this shader program.

#### Remarks

[Shader](#) uniforms are only supported in OpenGL 2.0+.

7.47.3.11 `virtual std::vector<VertexAttribute> LLGL::ShaderProgram::QueryVertexAttributes ( ) const` [pure virtual]

Returns a list of vertex attributes, which describe all vertex attributes within this shader program.

7.47.3.12 `virtual void LLGL::ShaderProgram::UnlockShaderUniform ( )` [pure virtual]

Unlocks the shader uniform handler.

#### See also

[LockShaderUniform](#)

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

- [ShaderProgram.h](#)

## 7.48 LLGL::ShaderSource Union Reference

[Shader](#) source code union.

```
#include <Shader.h>
```

### Classes

- struct [GLSL](#)  
*[Shader](#) source descriptor for [GLSL](#).*
- struct [HLSL](#)  
*[Shader](#) source descriptor for [HLSL](#).*

### Public Member Functions

- [ShaderSource](#) (const std::string &sourceCode)  
*Specifies the shader source code [GLSL](#).*
- [ShaderSource](#) (const std::string &sourceCode, const std::string &entryPoint, const std::string &target, int flags=0)  
*Specifies the shader source code for [HLSL](#).*
- [~ShaderSource](#) ()

## Public Attributes

- struct [LLGL::ShaderSource::GLSL sourceGLSL](#)
- struct [LLGL::ShaderSource::HLSL sourceHLSL](#)

### 7.48.1 Detailed Description

[Shader](#) source code union.

### 7.48.2 Constructor & Destructor Documentation

#### 7.48.2.1 `LLGL::ShaderSource::ShaderSource ( const std::string & sourceCode ) [inline]`

Specifies the shader source code [GLSL](#).

##### Parameters

in	<i>sourceCode</i>	Specifies the shader source code.
----	-------------------	-----------------------------------

##### Note

Only supported with: OpenGL (for [GLSL](#)).

#### 7.48.2.2 `LLGL::ShaderSource::ShaderSource ( const std::string & sourceCode, const std::string & entryPoint, const std::string & target, int flags = 0 ) [inline]`

Specifies the shader source code for [HLSL](#).

##### Parameters

in	<i>sourceCode</i>	Specifies the shader source code.
in	<i>entryPoint</i>	Specifies the shader entry point.
in	<i>target</i>	Specifies the shader version target (see <a href="https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx">https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx</a> ).
in	<i>flags</i>	Specifies optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries. By default 0.

##### Note

Only supported with: Direct3D 11, Direct3D 12 (for [HLSL](#)).

#### 7.48.2.3 `LLGL::ShaderSource::~ShaderSource ( ) [inline]`

### 7.48.3 Member Data Documentation

7.48.3.1 struct LLGL::ShaderSource::GLSL LLGL::ShaderSource::sourceGLSL

7.48.3.2 struct LLGL::ShaderSource::HLSL LLGL::ShaderSource::sourceHLSL

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

- [Shader.h](#)

## 7.49 LLGL::ShaderStageFlags Struct Reference

[Shader](#) stage flags.

```
#include <Shader.h>
```

### Public Types

- enum {  
[VertexStage](#) = (1 << 0), [TessControlStage](#) = (1 << 1), [TessEvaluationStage](#) = (1 << 2), [GeometryStage](#) = (1 << 3),  
[FragmentStage](#) = (1 << 4), [ComputeStage](#) = (1 << 5), [AllTessStages](#) = (TessControlStage | TessEvaluationStage), [AllGraphicsStages](#) = (VertexStage | AllTessStages | GeometryStage | FragmentStage),  
[AllStages](#) = (AllGraphicsStages | ComputeStage) }

### 7.49.1 Detailed Description

[Shader](#) stage flags.

#### Remarks

Specifies which shader stages are affected by a state change, e.g. at which shader stages a constant buffer is set. For the render systems, which do not support these flags, always all shader stages are affected.

#### Note

Only supported with: Direct3D 11

### 7.49.2 Member Enumeration Documentation

#### 7.49.2.1 anonymous enum

##### Enumerator

- [VertexStage](#)** Specifies the vertex shader stage.
- [TessControlStage](#)** Specifies the tessellation-control shader stage (also "Hull Shader").
- [TessEvaluationStage](#)** Specifies the tessellation-evaluation shader stage (also "Domain Shader").
- [GeometryStage](#)** Specifies the geometry shader stage.
- [FragmentStage](#)** Specifies the fragment shader stage (also "Pixel Shader").
- [ComputeStage](#)** Specifies the compute shader stage.
- [AllTessStages](#)** Specifies all tessellation stages, i.e. tessellation-control-, tessellation-evaluation shader stages.
- [AllGraphicsStages](#)** Specifies all graphics pipeline shader stages, i.e. vertex-, tessellation-, geometry-, and fragment shader stages.
- [AllStages](#)** Specifies all shader stages.

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

- [Shader.h](#)

## 7.50 LLGL::ShaderUniform Class Reference

[Shader](#) uniform setter interface.

```
#include <ShaderUniform.h>
```

### Public Member Functions

- virtual [~ShaderUniform](#) ()
- virtual void [SetUniform](#) (int location, const int value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector2i &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector3i &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector4i &value)=0
- virtual void [SetUniform](#) (int location, const float value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector2f &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector3f &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Vector4f &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Matrix2f &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Matrix3f &value)=0
- virtual void [SetUniform](#) (int location, const Gs::Matrix4f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const int value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector2i &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector3i &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector4i &value)=0
- virtual void [SetUniform](#) (const std::string &name, const float value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector2f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector3f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Vector4f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Matrix2f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Matrix3f &value)=0
- virtual void [SetUniform](#) (const std::string &name, const Gs::Matrix4f &value)=0
- virtual void [SetUniformArray](#) (int location, const int \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector2i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector3i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector4i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const float \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector2f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector3f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Vector4f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Matrix2f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Matrix3f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (int location, const Gs::Matrix4f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const int \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector2i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector3i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector4i \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const float \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector2f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector3f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Vector4f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Matrix2f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Matrix3f \*value, std::size\_t count)=0
- virtual void [SetUniformArray](#) (const std::string &name, const Gs::Matrix4f \*value, std::size\_t count)=0

### 7.50.1 Detailed Description

[Shader](#) uniform setter interface.

#### Remarks

This is only used by the OpenGL render system.

### 7.50.2 Constructor & Destructor Documentation

7.50.2.1 `virtual LLGL::ShaderUniform::~~ShaderUniform ( ) [inline],[virtual]`

### 7.50.3 Member Function Documentation

7.50.3.1 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const int value ) [pure virtual]`

7.50.3.2 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector2i & value ) [pure virtual]`

7.50.3.3 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector3i & value ) [pure virtual]`

7.50.3.4 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector4i & value ) [pure virtual]`

7.50.3.5 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const float value ) [pure virtual]`

7.50.3.6 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector2f & value ) [pure virtual]`

7.50.3.7 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector3f & value ) [pure virtual]`

7.50.3.8 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Vector4f & value ) [pure virtual]`

7.50.3.9 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Matrix2f & value ) [pure virtual]`

7.50.3.10 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Matrix3f & value ) [pure virtual]`

7.50.3.11 `virtual void LLGL::ShaderUniform::SetUniform ( int location, const Gs::Matrix4f & value ) [pure virtual]`

7.50.3.12 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const int value ) [pure virtual]`

7.50.3.13 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector2i & value ) [pure virtual]`

7.50.3.14 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector3i & value ) [pure virtual]`

7.50.3.15 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector4i & value ) [pure virtual]`

- 7.50.3.16 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const float value )` [pure virtual]
- 7.50.3.17 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector2f & value )` [pure virtual]
- 7.50.3.18 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector3f & value )` [pure virtual]
- 7.50.3.19 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Vector4f & value )` [pure virtual]
- 7.50.3.20 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Matrix2f & value )` [pure virtual]
- 7.50.3.21 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Matrix3f & value )` [pure virtual]
- 7.50.3.22 `virtual void LLGL::ShaderUniform::SetUniform ( const std::string & name, const Gs::Matrix4f & value )` [pure virtual]
- 7.50.3.23 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const int * value, std::size_t count )` [pure virtual]
- 7.50.3.24 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector2i * value, std::size_t count )` [pure virtual]
- 7.50.3.25 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector3i * value, std::size_t count )` [pure virtual]
- 7.50.3.26 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector4i * value, std::size_t count )` [pure virtual]
- 7.50.3.27 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const float * value, std::size_t count )` [pure virtual]
- 7.50.3.28 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector2f * value, std::size_t count )` [pure virtual]
- 7.50.3.29 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector3f * value, std::size_t count )` [pure virtual]
- 7.50.3.30 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Vector4f * value, std::size_t count )` [pure virtual]
- 7.50.3.31 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Matrix2f * value, std::size_t count )` [pure virtual]



- 7.50.3.32 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Matrix3f * value, std::size_t count )`  
[pure virtual]
- 7.50.3.33 `virtual void LLGL::ShaderUniform::SetUniformArray ( int location, const Gs::Matrix4f * value, std::size_t count )`  
[pure virtual]
- 7.50.3.34 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const int * value, std::size_t count )`  
[pure virtual]
- 7.50.3.35 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector2i * value, std::size_t count )` [pure virtual]
- 7.50.3.36 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector3i * value, std::size_t count )` [pure virtual]
- 7.50.3.37 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector4i * value, std::size_t count )` [pure virtual]
- 7.50.3.38 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const float * value, std::size_t count )`  
[pure virtual]
- 7.50.3.39 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector2f * value, std::size_t count )` [pure virtual]
- 7.50.3.40 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector3f * value, std::size_t count )` [pure virtual]
- 7.50.3.41 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Vector4f * value, std::size_t count )` [pure virtual]
- 7.50.3.42 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Matrix2f * value, std::size_t count )` [pure virtual]
- 7.50.3.43 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Matrix3f * value, std::size_t count )` [pure virtual]
- 7.50.3.44 `virtual void LLGL::ShaderUniform::SetUniformArray ( const std::string & name, const Gs::Matrix4f * value, std::size_t count )` [pure virtual]

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

- [ShaderUniform.h](#)

## 7.51 LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor Struct Reference

```
#include <RenderContextFlags.h>
```

## Public Attributes

- bool [screenSpaceOriginLowerLeft](#)  
*Specifies whether the screen-space origin is on the lower-left. By default false.*
- bool [invertFrontFace](#)  
*Specifies whether to invert front-facing. By default false.*

## 7.51.1 Member Data Documentation

### 7.51.1.1 bool LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::invertFrontFace

Specifies whether to invert front-facing. By default false.

#### Remarks

If this is true, the front facing (either GL\_CW or GL\_CCW) will be inverted, i.e. CCW becomes CW, and CW becomes CCW.

### 7.51.1.2 bool LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::screenSpaceOriginLowerLeft

Specifies whether the screen-space origin is on the lower-left. By default false.

#### Remarks

If this is true, the viewports and scissor rectangles of OpenGL are NOT emulated to the upper-left, which is the default to be uniform with other rendering APIs such as Direct3D and Vulkan.

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

- [RenderContextFlags.h](#)

## 7.52 LLGL::StencilDescriptor Struct Reference

Stencil state descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

## Public Attributes

- bool [testEnabled](#) = false  
*Specifies whether the stencil test is enabled or disabled.*
- [StencilFaceDescriptor](#) [front](#)  
*Specifies the front face settings for the stencil test.*
- [StencilFaceDescriptor](#) [back](#)  
*Specifies the back face settings for the stencil test.*

### 7.52.1 Detailed Description

Stencil state descriptor structure.

### 7.52.2 Member Data Documentation

#### 7.52.2.1 StencilFaceDescriptor LLGL::StencilDescriptor::back

Specifies the back face settings for the stencil test.

#### 7.52.2.2 StencilFaceDescriptor LLGL::StencilDescriptor::front

Specifies the front face settings for the stencil test.

#### 7.52.2.3 bool LLGL::StencilDescriptor::testEnabled = false

Specifies whether the stencil test is enabled or disabled.

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

- [GraphicsPipelineFlags.h](#)

## 7.53 LLGL::StencilFaceDescriptor Struct Reference

Stencil face descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

### Public Attributes

- [StencilOp stencilFailOp = StencilOp::Keep](#)  
*Specifies the operation to take when the stencil test fails.*
- [StencilOp depthFailOp = StencilOp::Keep](#)  
*Specifies the operation to take when the stencil test passes but the depth test fails.*
- [StencilOp depthPassOp = StencilOp::Keep](#)  
*Specifies the operation to take when both the stencil test and the depth test pass.*
- [CompareOp compareOp = CompareOp::Less](#)  
*Specifies the stencil compare operation.*
- `std::uint32_t compareMask = 0`
- `std::uint32_t writeMask = 0`
- `std::uint32_t reference = 0`  
*Specifies the stencil reference value.*

### 7.53.1 Detailed Description

Stencil face descriptor structure.

### 7.53.2 Member Data Documentation

7.53.2.1 `std::uint32_t LLGL::StencilFaceDescriptor::compareMask = 0`

7.53.2.2 `CompareOp LLGL::StencilFaceDescriptor::compareOp = CompareOp::Less`

Specifies the stencil compare operation.

7.53.2.3 `StencilOp LLGL::StencilFaceDescriptor::depthFailOp = StencilOp::Keep`

Specifies the operation to take when the stencil test passes but the depth test fails.

7.53.2.4 `StencilOp LLGL::StencilFaceDescriptor::depthPassOp = StencilOp::Keep`

Specifies the operation to take when both the stencil test and the depth test pass.

7.53.2.5 `std::uint32_t LLGL::StencilFaceDescriptor::reference = 0`

Specifies the stencil reference value.

#### Note

For Direct3D 11, only the stencil reference value of the "front" face will be used.

7.53.2.6 `StencilOp LLGL::StencilFaceDescriptor::stencilFailOp = StencilOp::Keep`

Specifies the operation to take when the stencil test fails.

7.53.2.7 `std::uint32_t LLGL::StencilFaceDescriptor::writeMask = 0`

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

- [GraphicsPipelineFlags.h](#)

## 7.54 LLGL::StorageBuffer Class Reference

Storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer") interface.

```
#include <StorageBuffer.h>
```

## Public Member Functions

- virtual [~StorageBuffer](#) ()

### 7.54.1 Detailed Description

Storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer") interface.

### 7.54.2 Constructor & Destructor Documentation

7.54.2.1 virtual LLGL::StorageBuffer::~StorageBuffer ( ) `[inline], [virtual]`

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

- [StorageBuffer.h](#)

## 7.55 LLGL::StorageBufferDescriptor Struct Reference

Storage buffer descriptor structure.

```
#include <StorageBuffer.h>
```

## Public Member Functions

- [StorageBufferDescriptor](#) ()=default
- [StorageBufferDescriptor](#) (unsigned int [size](#), [BufferUsage](#) [usage](#))

## Public Attributes

- unsigned int [size](#) = 0  
*Buffer size (in bytes).*
- [BufferUsage](#) [usage](#) = [BufferUsage::Dynamic](#)  
*Buffer usage (typically "BufferUsage::Dynamic", since a storage buffer is commonly frequently changed).*
- [StorageBufferType](#) [type](#) = [StorageBufferType::Buffer](#)  
*Specifies the storage buffer type.*

### 7.55.1 Detailed Description

Storage buffer descriptor structure.

## 7.55.2 Constructor & Destructor Documentation

7.55.2.1 `LLGL::StorageBufferDescriptor::StorageBufferDescriptor ( )` `[default]`

7.55.2.2 `LLGL::StorageBufferDescriptor::StorageBufferDescriptor ( unsigned int size, BufferUsage usage )` `[inline]`

## 7.55.3 Member Data Documentation

7.55.3.1 `unsigned int LLGL::StorageBufferDescriptor::size = 0`

Buffer size (in bytes).

7.55.3.2 `StorageBufferType LLGL::StorageBufferDescriptor::type = StorageBufferType::Buffer`

Specifies the storage buffer type.

### Remarks

In OpenGL there are only generic storage buffers (or rather "Shader Storage Buffer Objects").

### Note

Only supported with: Direct3D 11, Direct3D 12.

7.55.3.3 `BufferUsage LLGL::StorageBufferDescriptor::usage = BufferUsage::Dynamic`

Buffer usage (typically "BufferUsage::Dynamic", since a storage buffer is commonly frequently changed).

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

- [StorageBuffer.h](#)

## 7.56 LLGL::StorageBufferViewDescriptor Struct Reference

Storage buffer shader-view descriptor structure.

```
#include <StorageBuffer.h>
```

### Public Attributes

- `std::string` [name](#)  
*Storage buffer name.*
- `unsigned int` [index](#) = 0  
*Index of the storage buffer within the respective shader.*
- `StorageBufferType` [type](#) = `StorageBufferType::Buffer`  
*Storage buffer type.*

### 7.56.1 Detailed Description

Storage buffer shader-view descriptor structure.

#### Remarks

This structure is used to describe the view of a storage buffer within a shader.

### 7.56.2 Member Data Documentation

#### 7.56.2.1 unsigned int LLGL::StorageBufferViewDescriptor::index = 0

Index of the storage buffer within the respective shader.

#### 7.56.2.2 std::string LLGL::StorageBufferViewDescriptor::name

Storage buffer name.

#### 7.56.2.3 StorageBufferType LLGL::StorageBufferViewDescriptor::type = StorageBufferType::Buffer

Storage buffer type.

#### Remarks

For the OpenGL render system, this type is always '[StorageBufferType::Buffer](#)', since GLSL only supports generic shader storage buffers. Here is an example:

```
layout(std430, binding=0) buffer myBuffer
{
    vec4 myBufferArray[];
};
```

#### Note

Only supported with: Direct3D 11, Direct3D 12

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

- [StorageBuffer.h](#)

## 7.57 LLGL::SubTextureDescriptor Struct Reference

Sub-texture descriptor structure.

```
#include <TextureFlags.h>
```

## Classes

- struct [Texture1DDescriptor](#)
- struct [Texture2DDescriptor](#)
- struct [Texture3DDescriptor](#)
- struct [TextureCubeDescriptor](#)

## Public Member Functions

- [SubTextureDescriptor](#) ()
- [~SubTextureDescriptor](#) ()

## Public Attributes

- int [mipLevel](#)  
*Zero-based MIP-map level for the sub-texture.*
- union {  
[Texture1DDescriptor](#) [texture1DDesc](#)  
*Descriptor for 1D- and 1D-Array textures.*  
[Texture2DDescriptor](#) [texture2DDesc](#)  
*Descriptor for 2D- and 2D-Array textures.*  
[Texture3DDescriptor](#) [texture3DDesc](#)  
*Descriptor for 3D textures.*  
[TextureCubeDescriptor](#) [textureCubeDesc](#)  
*Descriptor for Cube- and Cube-Array textures.*  
};

### 7.57.1 Detailed Description

Sub-texture descriptor structure.

#### Remarks

This is used to write (or partially write) the image data of a texture MIP-map level.

### 7.57.2 Constructor & Destructor Documentation

7.57.2.1 `LLGL::SubTextureDescriptor::SubTextureDescriptor ( )` `[inline]`

7.57.2.2 `LLGL::SubTextureDescriptor::~~SubTextureDescriptor ( )` `[inline]`

### 7.57.3 Member Data Documentation

7.57.3.1 `union { ... }`

7.57.3.2 `int LLGL::SubTextureDescriptor::mipLevel`

Zero-based MIP-map level for the sub-texture.



**7.57.3.3 Texture1DDescriptor LLGL::SubTextureDescriptor::texture1DDesc**

Descriptor for 1D- and 1D-Array textures.

**7.57.3.4 Texture2DDescriptor LLGL::SubTextureDescriptor::texture2DDesc**

Descriptor for 2D- and 2D-Array textures.

**7.57.3.5 Texture3DDescriptor LLGL::SubTextureDescriptor::texture3DDesc**

Descriptor for 3D textures.

**7.57.3.6 TextureCubeDescriptor LLGL::SubTextureDescriptor::textureCubeDesc**

Descriptor for Cube- and Cube-Array textures.

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

- [TextureFlags.h](#)

**7.58 LLGL::Texture Class Reference**

[Texture](#) interface.

```
#include <Texture.h>
```

**Public Member Functions**

- virtual [~Texture](#) ()
- [TextureType GetType](#) () const  
*Returns the texture type. This type can only be changed by the render system.*
- virtual [Gs::Vector3i QueryMipLevelSize](#) (int mipLevel) const =0  
*Returns the texture size for the specified MIP-level.*

**Protected Member Functions**

- void [SetType](#) (const [TextureType](#) type)

**7.58.1 Detailed Description**

[Texture](#) interface.

## 7.58.2 Constructor & Destructor Documentation

7.58.2.1 `virtual LLGL::Texture::~Texture ( ) [inline],[virtual]`

## 7.58.3 Member Function Documentation

7.58.3.1 `TextureType LLGL::Texture::GetType ( ) const [inline]`

Returns the texture type. This type can only be changed by the render system.

### See also

RenderSystem::WriteTexture1D  
RenderSystem::WriteTexture2D  
RenderSystem::WriteTexture3D  
RenderSystem::WriteTextureCube

7.58.3.2 `virtual Gs::Vector3i LLGL::Texture::QueryMipLevelSize ( int mipLevel ) const [pure virtual]`

Returns the texture size for the specified MIP-level.

### Parameters

in	<i>mipLevel</i>	Specifies the MIP-map level to query from. The first and largest MIP-map is level zero. If this level is greater than or equal to the number of MIP-maps this texture has, the return value is undefined (i.e. depends on the render system).
----	-----------------	---

### See also

RenderContext::GenerateMips

7.58.3.3 `void LLGL::Texture::SetType ( const TextureType type ) [inline],[protected]`

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

- [Texture.h](#)

## 7.59 LLGL::SubTextureDescriptor::Texture1DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

## Public Attributes

- int [x](#)  
*Sub-texture X-axis offset.*
- unsigned int [layerOffset](#)  
*Zero-based layer offset.*
- int [width](#)  
*Sub-texture width.*
- unsigned int [layers](#)  
*Number of texture array layers.*

## 7.59.1 Member Data Documentation

### 7.59.1.1 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::layerOffset

Zero-based layer offset.

### 7.59.1.2 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::layers

Number of texture array layers.

### 7.59.1.3 int LLGL::SubTextureDescriptor::Texture1DDescriptor::width

Sub-texture width.

### 7.59.1.4 int LLGL::SubTextureDescriptor::Texture1DDescriptor::x

Sub-texture X-axis offset.

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

- [TextureFlags.h](#)

## 7.60 LLGL::TextureDescriptor::Texture1DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

## Public Attributes

- int [width](#)  
*[Texture](#) width.*
- unsigned int [layers](#)  
*Number of texture array layers.*

### 7.60.1 Member Data Documentation

#### 7.60.1.1 unsigned int LLGL::TextureDescriptor::Texture1DDescriptor::layers

Number of texture array layers.

#### 7.60.1.2 int LLGL::TextureDescriptor::Texture1DDescriptor::width

[Texture](#) width.

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

- [TextureFlags.h](#)

## 7.61 LLGL::TextureDescriptor::Texture2DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

### Public Attributes

- int [width](#)  
*[Texture](#) width.*
- int [height](#)  
*[Texture](#) height.*
- unsigned int [layers](#)  
*Number of texture array layers.*

### 7.61.1 Member Data Documentation

#### 7.61.1.1 int LLGL::TextureDescriptor::Texture2DDescriptor::height

[Texture](#) height.

#### 7.61.1.2 unsigned int LLGL::TextureDescriptor::Texture2DDescriptor::layers

Number of texture array layers.

#### 7.61.1.3 int LLGL::TextureDescriptor::Texture2DDescriptor::width

[Texture](#) width.

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

- [TextureFlags.h](#)

## 7.62 LLGL::SubTextureDescriptor::Texture2DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

### Public Attributes

- int `x`  
*Sub-texture X-axis offset.*
- int `y`  
*Sub-texture Y-axis offset.*
- unsigned int `layerOffset`  
*Zero-based layer offset.*
- int `width`  
*Sub-texture width.*
- int `height`  
*Sub-texture height.*
- unsigned int `layers`  
*Number of texture array layers.*

### 7.62.1 Member Data Documentation

#### 7.62.1.1 int LLGL::SubTextureDescriptor::Texture2DDescriptor::height

Sub-texture height.

#### 7.62.1.2 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::layerOffset

Zero-based layer offset.

#### 7.62.1.3 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::layers

Number of texture array layers.

#### 7.62.1.4 int LLGL::SubTextureDescriptor::Texture2DDescriptor::width

Sub-texture width.

#### 7.62.1.5 int LLGL::SubTextureDescriptor::Texture2DDescriptor::x

Sub-texture X-axis offset.

#### 7.62.1.6 int LLGL::SubTextureDescriptor::Texture2DDescriptor::y

Sub-texture Y-axis offset.

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

- [TextureFlags.h](#)

### 7.63 LLGL::TextureDescriptor::Texture3DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

#### Public Attributes

- int [width](#)  
*Texture width.*
- int [height](#)  
*Texture height.*
- int [depth](#)  
*Texture depth.*

#### 7.63.1 Member Data Documentation

##### 7.63.1.1 int LLGL::TextureDescriptor::Texture3DDescriptor::depth

[Texture](#) depth.

##### 7.63.1.2 int LLGL::TextureDescriptor::Texture3DDescriptor::height

[Texture](#) height.

##### 7.63.1.3 int LLGL::TextureDescriptor::Texture3DDescriptor::width

[Texture](#) width.

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

- [TextureFlags.h](#)

### 7.64 LLGL::SubTextureDescriptor::Texture3DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

## Public Attributes

- int [x](#)  
*Sub-texture X-axis offset.*
- int [y](#)  
*Sub-texture Y-axis offset.*
- int [z](#)  
*Sub-texture Z-axis offset.*
- int [width](#)  
*Sub-texture width.*
- int [height](#)  
*Sub-texture height.*
- int [depth](#)  
*Number of texture array layers.*

### 7.64.1 Member Data Documentation

#### 7.64.1.1 int LLGL::SubTextureDescriptor::Texture3DDescriptor::depth

Number of texture array layers.

#### 7.64.1.2 int LLGL::SubTextureDescriptor::Texture3DDescriptor::height

Sub-texture height.

#### 7.64.1.3 int LLGL::SubTextureDescriptor::Texture3DDescriptor::width

Sub-texture width.

#### 7.64.1.4 int LLGL::SubTextureDescriptor::Texture3DDescriptor::x

Sub-texture X-axis offset.

#### 7.64.1.5 int LLGL::SubTextureDescriptor::Texture3DDescriptor::y

Sub-texture Y-axis offset.

#### 7.64.1.6 int LLGL::SubTextureDescriptor::Texture3DDescriptor::z

Sub-texture Z-axis offset.

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

- [TextureFlags.h](#)

## 7.65 LLGL::SubTextureDescriptor::TextureCubeDescriptor Struct Reference

```
#include <TextureFlags.h>
```

### Public Attributes

- `int x`  
*Sub-texture X-axis offset.*
- `int y`  
*Sub-texture Y-axis offset.*
- `unsigned int layerOffset`  
*Zero-based layer offset.*
- `int width`  
*Sub-texture width.*
- `int height`  
*Sub-texture height.*
- `unsigned int cubeFaces`  
*Number of cube-faces. To have all faces of N cube-texture layers, this value must be a N\*6.*
- `AxisDirection cubeFaceOffset`  
*First cube face in the current layer.*

### 7.65.1 Member Data Documentation

#### 7.65.1.1 AxisDirection LLGL::SubTextureDescriptor::TextureCubeDescriptor::cubeFaceOffset

First cube face in the current layer.

#### 7.65.1.2 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::cubeFaces

Number of cube-faces. To have all faces of N cube-texture layers, this value must be a N\*6.

#### 7.65.1.3 int LLGL::SubTextureDescriptor::TextureCubeDescriptor::height

Sub-texture height.

#### 7.65.1.4 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::layerOffset

Zero-based layer offset.

#### 7.65.1.5 int LLGL::SubTextureDescriptor::TextureCubeDescriptor::width

Sub-texture width.



#### 7.65.1.6 int LLGL::SubTextureDescriptor::TextureCubeDescriptor::x

Sub-texture X-axis offset.

#### 7.65.1.7 int LLGL::SubTextureDescriptor::TextureCubeDescriptor::y

Sub-texture Y-axis offset.

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

- [TextureFlags.h](#)

## 7.66 LLGL::TextureDescriptor::TextureCubeDescriptor Struct Reference

```
#include <TextureFlags.h>
```

### Public Attributes

- int [width](#)  
*Texture width.*
- int [height](#)  
*Texture height.*
- unsigned int [layers](#)  
*Number of texture array layers (internally it will be a multiple of 6).*

### 7.66.1 Member Data Documentation

#### 7.66.1.1 int LLGL::TextureDescriptor::TextureCubeDescriptor::height

[Texture](#) height.

#### 7.66.1.2 unsigned int LLGL::TextureDescriptor::TextureCubeDescriptor::layers

Number of texture array layers (internally it will be a multiple of 6).

#### 7.66.1.3 int LLGL::TextureDescriptor::TextureCubeDescriptor::width

[Texture](#) width.

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

- [TextureFlags.h](#)

## 7.67 LLGL::TextureDescriptor Struct Reference

[Texture](#) descriptor structure.

```
#include <TextureFlags.h>
```

### Classes

- struct [Texture1DDescriptor](#)
- struct [Texture2DDescriptor](#)
- struct [Texture3DDescriptor](#)
- struct [TextureCubeDescriptor](#)

### Public Member Functions

- [TextureDescriptor](#) ()
- [~TextureDescriptor](#) ()

### Public Attributes

- [TextureType](#) type  
*Texture type.*
- [TextureFormat](#) format  
*Texture hardware format.*
- union {  
    [Texture1DDescriptor](#) texture1DDesc  
        *Descriptor for 1D- and 1D-Array textures.*  
    [Texture2DDescriptor](#) texture2DDesc  
        *Descriptor for 2D- and 2D-Array textures.*  
    [Texture3DDescriptor](#) texture3DDesc  
        *Descriptor for 3D textures.*  
    [TextureCubeDescriptor](#) textureCubeDesc  
        *Descriptor for Cube- and Cube-Array textures.*  
};

### 7.67.1 Detailed Description

[Texture](#) descriptor structure.

#### Remarks

This is used to specify the dimensions of a texture which is to be created.

## 7.67.2 Constructor & Destructor Documentation

7.67.2.1 LLGL::TextureDescriptor::TextureDescriptor ( ) [inline]

7.67.2.2 LLGL::TextureDescriptor::~~TextureDescriptor ( ) [inline]

## 7.67.3 Member Data Documentation

7.67.3.1 union { ... }

7.67.3.2 TextureFormat LLGL::TextureDescriptor::format

[Texture](#) hardware format.

7.67.3.3 Texture1DDescriptor LLGL::TextureDescriptor::texture1DDesc

Descriptor for 1D- and 1D-Array textures.

7.67.3.4 Texture2DDescriptor LLGL::TextureDescriptor::texture2DDesc

Descriptor for 2D- and 2D-Array textures.

7.67.3.5 Texture3DDescriptor LLGL::TextureDescriptor::texture3DDesc

Descriptor for 3D textures.

7.67.3.6 TextureCubeDescriptor LLGL::TextureDescriptor::textureCubeDesc

Descriptor for Cube- and Cube-Array textures.

7.67.3.7 TextureType LLGL::TextureDescriptor::type

[Texture](#) type.

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

- [TextureFlags.h](#)

## 7.68 LLGL::Timer Class Reference

```
#include <Timer.h>
```

## Public Types

- using `FrameCount` = unsigned long long

## Public Member Functions

- virtual `~Timer` ()
- virtual void `Start` ()=0  
*Starts the timer.*
- virtual double `Stop` ()=0  
*Stops the timer and returns the elapsed time since "Start" was called.*
- virtual double `GetFrequency` () const =0  
*Returns the frequency this timer can measure time (e.g. for milliseconds this is 1000.0).*
- void `MeasureTime` ()  
*Measures the time (elapsed time, and frame count) for each frame.*
- void `ResetFrameCounter` ()  
*Restes the frame counter.*
- double `GetDeltaTime` () const  
*Returns the elapsed time (in seconds) between the current and the previous frame.*
- `FrameCount GetFrameCount` () const  
*Returns the number of counted frames.*

## Static Public Member Functions

- static std::unique\_ptr< `Timer` > `Create` ()  
*Creates a platform specific timer object.*

### 7.68.1 Member Typedef Documentation

- 7.68.1.1 using `LLGL::Timer::FrameCount` = unsigned long long

### 7.68.2 Constructor & Destructor Documentation

- 7.68.2.1 virtual `LLGL::Timer::~~Timer` ( ) [`virtual`]

### 7.68.3 Member Function Documentation

- 7.68.3.1 static std::unique\_ptr<`Timer`> `LLGL::Timer::Create` ( ) [`static`]

Creates a platform specific timer object.

### 7.68.3.2 double LLGL::Timer::GetDeltaTime ( ) const [inline]

Returns the elapsed time (in seconds) between the current and the previous frame.

#### Remarks

This requires that "MeasureTime" is called once every frame.

#### See also

[MeasureTime](#)

### 7.68.3.3 FrameCount LLGL::Timer::GetFrameCount ( ) const [inline]

Returns the number of counted frames.

#### Remarks

This requires that "MeasureTime" is called once every frame.

#### See also

[MeasureTime](#)

### 7.68.3.4 virtual double LLGL::Timer::GetFrequency ( ) const [pure virtual]

Returns the frequency this timer can measure time (e.g. for milliseconds this is 1000.0).

### 7.68.3.5 void LLGL::Timer::MeasureTime ( )

Measures the time (elapsed time, and frame count) for each frame.

#### See also

[GetDeltaTime](#)  
[GetFrameCount\(\)](#)

### 7.68.3.6 void LLGL::Timer::ResetFrameCounter ( )

Restes the frame counter.

#### See also

[GetFrameCount](#)

**7.68.3.7** `virtual void LLGL::Timer::Start ( ) [pure virtual]`

Starts the timer.

**7.68.3.8** `virtual double LLGL::Timer::Stop ( ) [pure virtual]`

Stops the timer and returns the elapsed time since "Start" was called.

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

- [Timer.h](#)

## 7.69 LLGL::UniformDescriptor Struct Reference

[Shader](#) uniform descriptor structure.

```
#include <ShaderUniform.h>
```

### Public Attributes

- `std::string name`
- `UniformType type = UniformType::Float`
- `int location = 0`
- `unsigned int size = 0`

### 7.69.1 Detailed Description

[Shader](#) uniform descriptor structure.

### 7.69.2 Member Data Documentation

**7.69.2.1** `int LLGL::UniformDescriptor::location = 0`

**7.69.2.2** `std::string LLGL::UniformDescriptor::name`

**7.69.2.3** `unsigned int LLGL::UniformDescriptor::size = 0`

**7.69.2.4** `UniformType LLGL::UniformDescriptor::type = UniformType::Float`

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

- [ShaderUniform.h](#)

## 7.70 LLGL::VertexAttribute Struct Reference

Vertex attribute class.

```
#include <VertexAttribute.h>
```

### Public Attributes

- **DataType** `dataType` = **DataType::Float**  
*Data type of the vertex attribute components. By default [DataType::Float](#).*
- **bool** `conversion` = false  
*Specifies whether non-floating-point data types are to be converted to floating-points. By default false.*
- **bool** `perInstance` = false  
*Specifies whether this is a per-instance data. If false, this is a per-vertex data.*
- **unsigned int** `components` = 4  
*Number of components: 1, 2, 3, or 4. By default 4.*
- **unsigned int** `offset` = 0  
*Byte offset for within each vertex. By default 0.*
- **std::string** `name`  
*Vertex attribute name (for GLSL) or semantic name (for HLSL).*
- **unsigned int** `semanticIndex` = 0  
*Semantic index (only relevant for HLSL).*

### 7.70.1 Detailed Description

Vertex attribute class.

### 7.70.2 Member Data Documentation

#### 7.70.2.1 **unsigned int** LLGL::VertexAttribute::components = 4

Number of components: 1, 2, 3, or 4. By default 4.

#### 7.70.2.2 **bool** LLGL::VertexAttribute::conversion = false

Specifies whether non-floating-point data types are to be converted to floating-points. By default false.

#### 7.70.2.3 **DataType** LLGL::VertexAttribute::dataType = **DataType::Float**

Data type of the vertex attribute components. By default [DataType::Float](#).

#### 7.70.2.4 **std::string** LLGL::VertexAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

#### 7.70.2.5 unsigned int LLGL::VertexAttribute::offset = 0

Byte offset for within each vertex. By default 0.

#### 7.70.2.6 bool LLGL::VertexAttribute::perInstance = false

Specifies whether this is a per-instance data. If false, this is a per-vertex data.

#### 7.70.2.7 unsigned int LLGL::VertexAttribute::semanticIndex = 0

Semantic index (only relevant for HLSL).

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

- [VertexAttribute.h](#)

## 7.71 LLGL::VertexBuffer Class Reference

Vertex buffer interface.

```
#include <VertexBuffer.h>
```

### Public Member Functions

- virtual [~VertexBuffer](#) ()
- const [VertexFormat](#) & [GetVertexFormat](#) () const

### Protected Member Functions

- void [SetVertexFormat](#) (const [VertexFormat](#) &vertexFormat)

#### 7.71.1 Detailed Description

Vertex buffer interface.

#### 7.71.2 Constructor & Destructor Documentation

7.71.2.1 virtual LLGL::VertexBuffer::~~VertexBuffer ( ) [\[inline\]](#), [\[virtual\]](#)

#### 7.71.3 Member Function Documentation

7.71.3.1 const [VertexFormat](#)& LLGL::VertexBuffer::GetVertexFormat ( ) const [\[inline\]](#)

7.71.3.2 void LLGL::VertexBuffer::SetVertexFormat ( const [VertexFormat](#) & *vertexFormat* ) [\[inline\]](#), [\[protected\]](#)

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

- [VertexBuffer.h](#)



## 7.72 LLGL::VertexBufferDescriptor Struct Reference

Vertex buffer descriptor structure.

```
#include <VertexBuffer.h>
```

### Public Member Functions

- [VertexBufferDescriptor](#) ()=default
- [VertexBufferDescriptor](#) (unsigned int [size](#), [BufferUsage](#) [usage](#), const [VertexFormat](#) &[vertexFormat](#))

### Public Attributes

- unsigned int [size](#) = 0  
*Buffer size (in bytes).*
- [BufferUsage](#) [usage](#) = [BufferUsage::Static](#)  
*Buffer usage (typically "BufferUsage::Static", since a vertex buffer is rarely changed).*
- [VertexFormat](#) [vertexFormat](#)  
*Specifies the vertex format layout.*

### 7.72.1 Detailed Description

Vertex buffer descriptor structure.

### 7.72.2 Constructor & Destructor Documentation

7.72.2.1 LLGL::VertexBufferDescriptor::VertexBufferDescriptor ( ) [default]

7.72.2.2 LLGL::VertexBufferDescriptor::VertexBufferDescriptor ( unsigned int [size](#), [BufferUsage](#) [usage](#), const [VertexFormat](#) & [vertexFormat](#) ) [inline]

### 7.72.3 Member Data Documentation

7.72.3.1 unsigned int LLGL::VertexBufferDescriptor::size = 0

Buffer size (in bytes).

7.72.3.2 [BufferUsage](#) LLGL::VertexBufferDescriptor::usage = [BufferUsage::Static](#)

Buffer usage (typically "BufferUsage::Static", since a vertex buffer is rarely changed).

### 7.72.3.3 VertexFormat LLGL::VertexBufferDescriptor::vertexFormat

Specifies the vertex format layout.

#### Remarks

This is required to tell the renderer how the vertex attributes are stored inside the vertex buffer and it must be the same vertex format which is used for the respective graphics pipeline shader program.

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

- [VertexBuffer.h](#)

## 7.73 LLGL::VertexFormat Class Reference

Vertex format descriptor class.

```
#include <VertexFormat.h>
```

### Public Member Functions

- void [AddAttribute](#) (const std::string &name, const [DataType](#) dataType, unsigned int components, bool conversion=false, bool perInstance=false)  
*Adds a new vertex attribute to this vertex format with a specified name (used for GLSL).*
- void [AddAttribute](#) (const std::string &semanticName, unsigned int semanticIndex, const [DataType](#) dataType, unsigned int components, bool conversion=false, bool perInstance=false)  
*Adds a new vertex attribute to this vertex format with a specified semantic (used for HLSL).*
- const std::vector< [VertexAttribute](#) > & [GetAttributes](#) () const  
*Returns the list of all vertex attributes.*
- unsigned int [GetFormatSize](#) () const  
*Returns the size of this vertex format (in bytes).*

### 7.73.1 Detailed Description

Vertex format descriptor class.

#### Remarks

A vertex format is required to describe how the vertex attributes are supported inside a vertex buffer.

#### See also

[VertexBuffer](#)

### 7.73.2 Member Function Documentation

- 7.73.2.1 void LLGL::VertexFormat::AddAttribute ( const std::string & name, const [DataType](#) dataType, unsigned int components, bool conversion = false, bool perInstance = false )

Adds a new vertex attribute to this vertex format with a specified name (used for GLSL).

## Parameters

in	<i>name</i>	Specifies the attribute name.
in	<i>dataType</i>	Specifies the data type of the attribute components.
in	<i>components</i>	Specifies the number of attribute components. This must be 1, 2, 3, or 4.
in	<i>conversion</i>	Specifies whether to convert integral vertex attributes to normalized floating-point types. By default false.
in	<i>perInstance</i>	Specifies whether this is per-instance data. If false, this is per-vertex data. By default false.

## Remarks

This is equivalent to:

```
AddAttribute(name, 0, dataType, components, conversion);
```

## Exceptions

<i>std::invalid_argument</i>	If 'components' is neither 1, 2, 3, nor 4.
------------------------------	--

## See also

```
AddAttribute(const std::string&, unsigned int, const DataType, unsigned int, bool, bool)
```

**7.73.2.2** `void LLGL::VertexFormat::AddAttribute ( const std::string & semanticName, unsigned int semanticIndex, const DataType dataType, unsigned int components, bool conversion = false, bool perInstance = false )`

Adds a new vertex attribute to this vertex format with a specified semantic (used for HLSL).

## Parameters

in	<i>semanticName</i>	Specifies the semantic name (For Direct3D).
in	<i>semanticIndex</i>	Specifies the semantic index (For Direct3D).
in	<i>dataType</i>	Specifies the data type of the attribute components.
in	<i>components</i>	Specifies the number of attribute components. This must be 1, 2, 3, or 4.
in	<i>conversion</i>	Specifies whether to convert integral vertex attributes to normalized floating-point types. By default false.
in	<i>perInstance</i>	Specifies whether this is per-instance data. If false, this is per-vertex data. By default false.

## Exceptions

<i>std::invalid_argument</i>	If 'components' is neither 1, 2, 3, nor 4.
------------------------------	--

**7.73.2.3** `const std::vector<VertexAttribute>& LLGL::VertexFormat::GetAttributes ( ) const` `[inline]`

Returns the list of all vertex attributes.

See also

[AddAttribute](#)

**7.73.2.4** `unsigned int LLGL::VertexFormat::GetFormatSize ( ) const` `[inline]`

Returns the size of this vertex format (in bytes).

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

- [VertexFormat.h](#)

## 7.74 LLGL::VideoAdapterDescriptor Struct Reference

Video adapter descriptor structure.

```
#include <VideoAdapter.h>
```

### Public Attributes

- `std::wstring` [name](#)  
*Hardware adapter name (name of the GPU).*
- `std::string` [vendor](#)  
*Vendor name.*
- `unsigned long long` [videoMemory](#) = 0  
*Video memory size (in bytes).*
- `std::vector< VideoOutput >` [outputs](#)  
*Adapter outputs.*

### 7.74.1 Detailed Description

Video adapter descriptor structure.

### 7.74.2 Member Data Documentation

**7.74.2.1** `std::wstring LLGL::VideoAdapterDescriptor::name`

Hardware adapter name (name of the GPU).

**7.74.2.2** `std::vector<VideoOutput> LLGL::VideoAdapterDescriptor::outputs`

Adapter outputs.

#### 7.74.2.3 std::string LLGL::VideoAdapterDescriptor::vendor

Vendor name.

#### 7.74.2.4 unsigned long long LLGL::VideoAdapterDescriptor::videoMemory = 0

Video memory size (in bytes).

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

- [VideoAdapter.h](#)

## 7.75 LLGL::VideoDisplayMode Struct Reference

```
#include <VideoAdapter.h>
```

### Public Attributes

- unsigned int [width](#) = 0
- unsigned int [height](#) = 0
- unsigned int [refreshRate](#) = 0

### 7.75.1 Member Data Documentation

#### 7.75.1.1 unsigned int LLGL::VideoDisplayMode::height = 0

#### 7.75.1.2 unsigned int LLGL::VideoDisplayMode::refreshRate = 0

#### 7.75.1.3 unsigned int LLGL::VideoDisplayMode::width = 0

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

- [VideoAdapter.h](#)

## 7.76 LLGL::VideoModeDescriptor Struct Reference

```
#include <RenderContextDescriptor.h>
```

## Public Attributes

- [Size resolution](#)  
*Screen resolution.*
- `int colorDepth = 32`  
*Color bit depth. Should be 24 or 32. By default 32.*
- `bool fullscreen = false`  
*Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.*
- `SwapChainMode swapChainMode = SwapChainMode::DoubleBuffering`  
*Swap chain buffering mode.*

## 7.76.1 Member Data Documentation

### 7.76.1.1 `int LLGL::VideoModeDescriptor::colorDepth = 32`

[Color](#) bit depth. Should be 24 or 32. By default 32.

### 7.76.1.2 `bool LLGL::VideoModeDescriptor::fullscreen = false`

Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

### 7.76.1.3 `Size LLGL::VideoModeDescriptor::resolution`

Screen resolution.

### 7.76.1.4 `SwapChainMode LLGL::VideoModeDescriptor::swapChainMode = SwapChainMode::DoubleBuffering`

Swap chain buffering mode.

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

- [RenderContextDescriptor.h](#)

## 7.77 LLGL::VideoOutput Struct Reference

```
#include <VideoAdapter.h>
```

## Public Attributes

- `std::vector< VideoDisplayMode > displayModes`

### 7.77.1 Member Data Documentation

#### 7.77.1.1 `std::vector<VideoDisplayMode> LLGL::VideoOutput::displayModes`

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

- [VideoAdapter.h](#)

## 7.78 LLGL::Viewport Struct Reference

[Viewport](#) dimensions.

```
#include <RenderContextFlags.h>
```

### Public Member Functions

- [Viewport](#) ()=default
- [Viewport](#) (const [Viewport](#) &)=default
- [Viewport](#) (float [x](#), float [y](#), float [width](#), float [height](#))
- [Viewport](#) (float [x](#), float [y](#), float [width](#), float [height](#), float [minDepth](#), float [maxDepth](#))

### Public Attributes

- float [x](#) = 0.0f  
*Left-top X coordinate.*
- float [y](#) = 0.0f  
*Left-top Y coordinate.*
- float [width](#) = 0.0f  
*Right-bottom width.*
- float [height](#) = 0.0f  
*Right-bottom height.*
- float [minDepth](#) = 0.0f  
*Minimal depth range.*
- float [maxDepth](#) = 1.0f  
*Maximal depth range.*

### 7.78.1 Detailed Description

[Viewport](#) dimensions.

#### Remarks

A viewport is in screen coordinates where the origin is in the left-top corner.

## 7.78.2 Constructor & Destructor Documentation

7.78.2.1 `LLGL::Viewport::Viewport ( )` `[default]`

7.78.2.2 `LLGL::Viewport::Viewport ( const Viewport & )` `[default]`

7.78.2.3 `LLGL::Viewport::Viewport ( float x, float y, float width, float height )` `[inline]`

7.78.2.4 `LLGL::Viewport::Viewport ( float x, float y, float width, float height, float minDepth, float maxDepth )` `[inline]`

## 7.78.3 Member Data Documentation

7.78.3.1 `float LLGL::Viewport::height = 0.0f`

Right-bottom height.

7.78.3.2 `float LLGL::Viewport::maxDepth = 1.0f`

Maximal depth range.

7.78.3.3 `float LLGL::Viewport::minDepth = 0.0f`

Minimal depth range.

7.78.3.4 `float LLGL::Viewport::width = 0.0f`

Right-bottom width.

7.78.3.5 `float LLGL::Viewport::x = 0.0f`

Left-top X coordinate.

7.78.3.6 `float LLGL::Viewport::y = 0.0f`

Left-top Y coordinate.

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

- [RenderContextFlags.h](#)

## 7.79 LLGL::VsyncDescriptor Struct Reference

```
#include <RenderContextDescriptor.h>
```



## Public Attributes

- bool [enabled](#) = false  
*Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.*
- unsigned int [refreshRate](#) = 60  
*Refresh rate (in Hz). By default 60.*
- unsigned int [interval](#) = 1  
*Synchronisation interval. Can be 1, 2, 3, or 4. If Vsync is disabled, this value is implicit zero.*

### 7.79.1 Member Data Documentation

#### 7.79.1.1 bool LLGL::VsyncDescriptor::enabled = false

Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.

#### 7.79.1.2 unsigned int LLGL::VsyncDescriptor::interval = 1

Synchronisation interval. Can be 1, 2, 3, or 4. If Vsync is disabled, this value is implicit zero.

#### 7.79.1.3 unsigned int LLGL::VsyncDescriptor::refreshRate = 60

Refresh rate (in Hz). By default 60.

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

- [RenderContextDescriptor.h](#)

## 7.80 LLGL::Window Class Reference

```
#include <Window.h>
```

### Classes

- class [EventListener](#)

## Public Member Functions

- virtual `~Window ()`
- virtual void `SetPosition` (const `Point` &position)=0
- virtual `Point GetPosition ()` const =0
- virtual void `SetSize` (const `Size` &size, bool useClientArea=true)=0
- virtual `Size GetSize` (bool useClientArea=true) const =0
- virtual void `SetTitle` (const std::wstring &title)=0
- virtual std::wstring `GetTitle ()` const =0
- virtual void `Show` (bool show=true)=0
- virtual bool `IsShown ()` const =0
- virtual `WindowDescriptor QueryDesc ()` const =0  
*Query a window descriptor, which describes the current state of this window.*
- virtual void `SetDesc` (const `WindowDescriptor` &desc)=0  
*Sets the new window descriptor.*
- virtual void `Recreate` (const `WindowDescriptor` &desc)=0  
*Recreates the internal window object. This may invalidate the native handle previously returned by "GetNativeHandle".*
- virtual void `GetNativeHandle` (void \*nativeHandle) const =0  
*Returns the native window handle.*
- bool `ProcessEvents ()`  
*Processes the events for this window (i.e. mouse movement, key presses etc.).*
- void `AddEventListener` (const std::shared\_ptr< `EventListener` > &eventListener)
- void `RemoveEventListener` (const `EventListener` \*eventListener)
- void `PostKeyDown` (`Key` keyCode)
- void `PostKeyUp` (`Key` keyCode)
- void `PostDoubleClick` (`Key` keyCode)
- void `PostChar` (wchar\_t chr)
- void `PostWheelMotion` (int motion)
- void `PostLocalMotion` (const `Point` &position)
- void `PostGlobalMotion` (const `Point` &motion)
- void `PostResize` (const `Size` &clientAreaSize)
- void `PostQuit ()`  
*Posts the 'OnQuit' event to all event listeners.*

## Static Public Member Functions

- static std::unique\_ptr< `Window` > `Create` (const `WindowDescriptor` &desc)

## Protected Member Functions

- virtual void `ProcessSystemEvents ()`=0

## 7.80.1 Constructor & Destructor Documentation

7.80.1.1 `virtual LLGL::Window::~~Window ( ) [virtual]`

## 7.80.2 Member Function Documentation

7.80.2.1 `void LLGL::Window::AddEventListener ( const std::shared_ptr< EventListener > & eventListener )`

7.80.2.2 `static std::unique_ptr<Window> LLGL::Window::Create ( const WindowDescriptor & desc ) [static]`

7.80.2.3 `virtual void LLGL::Window::GetNativeHandle ( void * nativeHandle ) const [pure virtual]`

Returns the native window handle.

### Remarks

This must be casted to a platform specific structure:

```
#include <LLGL/Platform/NativeHandle.h>
//...
LLGL::NativeHandle handle;
window.GetNativeHandle (&handle);
```

7.80.2.4 `virtual Point LLGL::Window::GetPosition ( ) const [pure virtual]`

7.80.2.5 `virtual Size LLGL::Window::GetSize ( bool useClientArea =true ) const [pure virtual]`

7.80.2.6 `virtual std::wstring LLGL::Window::GetTitle ( ) const [pure virtual]`

7.80.2.7 `virtual bool LLGL::Window::IsShown ( ) const [pure virtual]`

7.80.2.8 `void LLGL::Window::PostChar ( wchar_t chr )`

7.80.2.9 `void LLGL::Window::PostDoubleClick ( Key keyCode )`

7.80.2.10 `void LLGL::Window::PostGlobalMotion ( const Point & motion )`

7.80.2.11 `void LLGL::Window::PostKeyDown ( Key keyCode )`

7.80.2.12 `void LLGL::Window::PostKeyUp ( Key keyCode )`

7.80.2.13 `void LLGL::Window::PostLocalMotion ( const Point & position )`

7.80.2.14 `void LLGL::Window::PostQuit ( )`

Posts the 'OnQuit' event to all event listeners.

### Remarks

If at least one event listener returns false within the "OnQuit" callback, the window will not quit. If all event listener return true within the "OnQuit" callback, "ProcessEvents" will returns false from now on.

7.80.2.15 void LLGL::Window::PostResize ( const **Size** & *clientAreaSize* )

7.80.2.16 void LLGL::Window::PostWheelMotion ( int *motion* )

7.80.2.17 bool LLGL::Window::ProcessEvents ( )

Processes the events for this window (i.e. mouse movement, key presses etc.).

#### Returns

Once the "PostQuit" function was called on this window object, this function returns false. This will happen, when the user clicks on the close button.

7.80.2.18 virtual void LLGL::Window::ProcessSystemEvents ( ) [protected],[pure virtual]

7.80.2.19 virtual **WindowDescriptor** LLGL::Window::QueryDesc ( ) const [pure virtual]

[Query](#) a window descriptor, which describes the current state of this window.

7.80.2.20 virtual void LLGL::Window::Recreate ( const **WindowDescriptor** & *desc* ) [pure virtual]

Recreates the internal window object. This may invalidate the native handle previously returned by "GetNativeHandle".

#### See also

[GetNativeHandle](#)

7.80.2.21 void LLGL::Window::RemoveEventListener ( const **EventListener** \* *eventListener* )

7.80.2.22 virtual void LLGL::Window::SetDesc ( const **WindowDescriptor** & *desc* ) [pure virtual]

Sets the new window descriptor.

7.80.2.23 virtual void LLGL::Window::SetPosition ( const **Point** & *position* ) [pure virtual]

7.80.2.24 virtual void LLGL::Window::SetSize ( const **Size** & *size*, bool *useClientArea* = true ) [pure virtual]

7.80.2.25 virtual void LLGL::Window::SetTitle ( const std::wstring & *title* ) [pure virtual]

7.80.2.26 virtual void LLGL::Window::Show ( bool *show* = true ) [pure virtual]

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

- [Window.h](#)

## 7.81 LLGL::WindowDescriptor Struct Reference

[Window](#) descriptor structure.

```
#include <Window.h>
```

### Public Attributes

- `std::wstring` [title](#)
- [Point](#) [position](#)  
*Window position (relative to the client area).*
- [Size](#) [size](#)  
*Client area size.*
- `bool` [visible](#) = false
- `bool` [borderless](#) = false
- `bool` [resizable](#) = false
- `bool` [acceptDropFiles](#) = false
- `bool` [preventForPowerSafe](#) = false
- `bool` [centered](#) = false
- `const void *` [windowContext](#) = nullptr  
*Window context handle.*

### 7.81.1 Detailed Description

[Window](#) descriptor structure.

### 7.81.2 Member Data Documentation

7.81.2.1 `bool` LLGL::WindowDescriptor::acceptDropFiles = false

7.81.2.2 `bool` LLGL::WindowDescriptor::borderless = false

7.81.2.3 `bool` LLGL::WindowDescriptor::centered = false

7.81.2.4 `Point` LLGL::WindowDescriptor::position

[Window](#) position (relative to the client area).

7.81.2.5 `bool` LLGL::WindowDescriptor::preventForPowerSafe = false

7.81.2.6 `bool` LLGL::WindowDescriptor::resizable = false

7.81.2.7 `Size` LLGL::WindowDescriptor::size

Client area size.

7.81.2.8 `std::wstring LLGL::WindowDescriptor::title`

7.81.2.9 `bool LLGL::WindowDescriptor::visible = false`

7.81.2.10 `const void* LLGL::WindowDescriptor::windowContext = nullptr`

[Window](#) context handle.

#### Remarks

If used, this must be casted from a platform specific structure:

```
#include <LLGL/Platform/NativeHandle.h>
//...
LLGL::NativeContextHandle handle;
//handle.parentWindow = ...
windowDesc.windowContext = reinterpret_cast<const void*>(&handle);
```

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

- [Window.h](#)

## Chapter 8

# File Documentation

### 8.1 Color.h File Reference

```
#include <Gauss/Real.h>
#include <Gauss/Assert.h>
#include <Gauss/Tags.h>
#include <Gauss/Equals.h>
#include <algorithm>
```

#### Classes

- class [LLGL::Color< T, N >](#)  
*Base color class with N components.*

#### Namespaces

- [LLGL](#)

#### Functions

- template<typename T >  
T [LLGL::MaxColorValue](#) ()  
*Returns the maximal color value for the data type T. By default 1.*
- template<>  
unsigned char [LLGL::MaxColorValue< unsigned char >](#) ()  
*Specialized version. For unsigned 8-bit integers, the return value is 255.*
- template<>  
bool [LLGL::MaxColorValue< bool >](#) ()  
*Specialized version. For booleans, the return value is true.*
- template<typename T , std::size\_t N>  
Color< T, N > [LLGL::operator+](#) (const Color< T, N > &lhs, const Color< T, N > &rhs)
- template<typename T , std::size\_t N>  
Color< T, N > [LLGL::operator-](#) (const Color< T, N > &lhs, const Color< T, N > &rhs)

- `template<typename T, std::size_t N>`  
`Color< T, N > LLGL::operator*` (const Color< T, N > &lhs, const Color< T, N > &rhs)
- `template<typename T, std::size_t N>`  
`Color< T, N > LLGL::operator/` (const Color< T, N > &lhs, const Color< T, N > &rhs)
- `template<typename T, std::size_t N>`  
`Color< T, N > LLGL::operator*` (const Color< T, N > &lhs, const T &rhs)
- `template<typename T, std::size_t N>`  
`Color< T, N > LLGL::operator*` (const T &lhs, const Color< T, N > &rhs)
- `template<typename T, std::size_t N>`  
`Color< T, N > LLGL::operator/` (const Color< T, N > &lhs, const T &rhs)
- `template<typename T, std::size_t N>`  
`bool LLGL::operator==` (const Color< T, N > &lhs, const Color< T, N > &rhs)
- `template<typename T, std::size_t N>`  
`bool LLGL::operator!=` (const Color< T, N > &lhs, const Color< T, N > &rhs)

## 8.2 ColorRGB.h File Reference

```
#include "Color.h"
```

### Classes

- class `LLGL::Color< T, 3u >`  
*RGB color class with components: r, g, and b.*

### Namespaces

- `LLGL`

### Typedefs

- `template<typename T >`  
`using LLGL::ColorRGBT = Color< T, 3 >`
- `using LLGL::ColorRGB = ColorRGBT< Gs::Real >`
- `using LLGL::ColorRGBb = ColorRGBT< bool >`
- `using LLGL::ColorRGBf = ColorRGBT< float >`
- `using LLGL::ColorRGBd = ColorRGBT< double >`
- `using LLGL::ColorRGBub = ColorRGBT< unsigned char >`

## 8.3 ColorRGBA.h File Reference

```
#include "Color.h"
```

### Classes

- class `LLGL::Color< T, 4u >`  
*RGBA color class with components: r, g, b, and a.*



## Namespaces

- [LLGL](#)

## Typedefs

- `template<typename T>`  
`using LLGL::ColorRGBAT = Color< T, 4 >`
- `using LLGL::ColorRGBA = ColorRGBAT< Gs::Real >`
- `using LLGL::ColorRGBAb = ColorRGBAT< bool >`
- `using LLGL::ColorRGBAf = ColorRGBAT< float >`
- `using LLGL::ColorRGBAd = ColorRGBAT< double >`
- `using LLGL::ColorRGB Aub = ColorRGBAT< unsigned char >`

## 8.4 ComputePipeline.h File Reference

```
#include "Export.h"
```

## Classes

- struct [LLGL::ComputePipelineDescriptor](#)  
*Compute pipeline descriptor structure.*
- class [LLGL::ComputePipeline](#)  
*Compute pipeline interface.*

## Namespaces

- [LLGL](#)

## 8.5 ConstantBuffer.h File Reference

```
#include "Export.h"
#include "RenderSystemFlags.h"
#include <string>
```

## Classes

- struct [LLGL::ConstantBufferDescriptor](#)  
*Constant buffer descriptor structure.*
- struct [LLGL::ConstantBufferViewDescriptor](#)  
*Constant buffer shader-view descriptor structure.*
- class [LLGL::ConstantBuffer](#)  
*Constant buffer (also "Uniform Buffer Object") interface.*

## Namespaces

- [LLGL](#)

## 8.6 Desktop.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "RenderContextDescriptor.h"
```

## Namespaces

- [LLGL](#)
- [LLGL::Desktop](#)

## Functions

- [LLGL\\_EXPORT](#) `Size` [LLGL::Desktop::GetResolution](#) ()  
*Returns the desktop resolution.*
- [LLGL\\_EXPORT](#) `int` [LLGL::Desktop::GetColorDepth](#) ()  
*Returns the desktop color depth (bits per pixel).*
- [LLGL\\_EXPORT](#) `bool` [LLGL::Desktop::SetVideoMode](#) (const VideoModeDescriptor &videoMode)  
*Sets the new specified video mode for the desktop (resolution and fullscreen mode).*
- [LLGL\\_EXPORT](#) `bool` [LLGL::Desktop::ResetVideoMode](#) ()  
*Restes the standard video mode for the desktop.*

## 8.7 Export.h File Reference

## Macros

- `#define` [LLGL\\_EXPORT](#)

### 8.7.1 Macro Definition Documentation

#### 8.7.1.1 `#define LLGL_EXPORT`

## 8.8 GraphicsPipeline.h File Reference

```
#include "Export.h"
#include "GraphicsPipelineFlags.h"
```

## Classes

- class [LLGL::GraphicsPipeline](#)  
*Graphics pipeline interface.*

## Namespaces

- [LLGL](#)

## 8.9 GraphicsPipelineFlags.h File Reference

```
#include "Export.h"  
#include "ColorRGBA.h"  
#include <vector>  
#include <cstdint>
```

## Classes

- struct [LLGL::DepthDescriptor](#)  
*Depth state descriptor structure.*
- struct [LLGL::StencilFaceDescriptor](#)  
*Stencil face descriptor structure.*
- struct [LLGL::StencilDescriptor](#)  
*Stencil state descriptor structure.*
- struct [LLGL::RasterizerDescriptor](#)  
*Rasterizer state descriptor structure.*
- struct [LLGL::BlendTargetDescriptor](#)  
*Blend target state descriptor structure.*
- struct [LLGL::BlendDescriptor](#)  
*Blending state descriptor structure.*
- struct [LLGL::GraphicsPipelineDescriptor](#)  
*Graphics pipeline descriptor structure.*

## Namespaces

- [LLGL](#)

## Enumerations

- enum `LLGL::PrimitiveTopology` {  
`LLGL::PrimitiveTopology::PointList`, `LLGL::PrimitiveTopology::LineList`, `LLGL::PrimitiveTopology::LineStrip`,  
`LLGL::PrimitiveTopology::LineLoop`,  
`LLGL::PrimitiveTopology::LineListAdjacency`, `LLGL::PrimitiveTopology::LineStripAdjacency`, `LLGL::PrimitiveTopology::TriangleList`, `LLGL::PrimitiveTopology::TriangleStrip`,  
`LLGL::PrimitiveTopology::TriangleFan`, `LLGL::PrimitiveTopology::TriangleListAdjacency`, `LLGL::PrimitiveTopology::TriangleStripAdjacency`, `LLGL::PrimitiveTopology::Patches1`,  
`LLGL::PrimitiveTopology::Patches2`, `LLGL::PrimitiveTopology::Patches3`, `LLGL::PrimitiveTopology::Patches4`,  
`LLGL::PrimitiveTopology::Patches5`,  
`LLGL::PrimitiveTopology::Patches6`, `LLGL::PrimitiveTopology::Patches7`, `LLGL::PrimitiveTopology::Patches8`,  
`LLGL::PrimitiveTopology::Patches9`,  
`LLGL::PrimitiveTopology::Patches10`, `LLGL::PrimitiveTopology::Patches11`, `LLGL::PrimitiveTopology::Patches12`, `LLGL::PrimitiveTopology::Patches13`,  
`LLGL::PrimitiveTopology::Patches14`, `LLGL::PrimitiveTopology::Patches15`, `LLGL::PrimitiveTopology::Patches16`, `LLGL::PrimitiveTopology::Patches17`,  
`LLGL::PrimitiveTopology::Patches18`, `LLGL::PrimitiveTopology::Patches19`, `LLGL::PrimitiveTopology::Patches20`, `LLGL::PrimitiveTopology::Patches21`,  
`LLGL::PrimitiveTopology::Patches22`, `LLGL::PrimitiveTopology::Patches23`, `LLGL::PrimitiveTopology::Patches24`, `LLGL::PrimitiveTopology::Patches25`,  
`LLGL::PrimitiveTopology::Patches26`, `LLGL::PrimitiveTopology::Patches27`, `LLGL::PrimitiveTopology::Patches28`, `LLGL::PrimitiveTopology::Patches29`,  
`LLGL::PrimitiveTopology::Patches30`, `LLGL::PrimitiveTopology::Patches31`, `LLGL::PrimitiveTopology::Patches32` }

*Primitive topology enumeration.*

- enum `LLGL::CompareOp` {  
`LLGL::CompareOp::Never`, `LLGL::CompareOp::Less`, `LLGL::CompareOp::Equal`, `LLGL::CompareOp::LessEqual`,  
`LLGL::CompareOp::Greater`, `LLGL::CompareOp::NotEqual`, `LLGL::CompareOp::GreaterEqual`, `LLGL::CompareOp::Ever` }

*Compare operations enumeration.*

- enum `LLGL::StencilOp` {  
`LLGL::StencilOp::Keep`, `LLGL::StencilOp::Zero`, `LLGL::StencilOp::Replace`, `LLGL::StencilOp::IncClamp`,  
`LLGL::StencilOp::DecClamp`, `LLGL::StencilOp::Invert`, `LLGL::StencilOp::IncWrap`, `LLGL::StencilOp::DecWrap` }

*Stencil operations enumeration.*

- enum `LLGL::BlendOp` {  
`LLGL::BlendOp::Zero`, `LLGL::BlendOp::One`, `LLGL::BlendOp::SrcColor`, `LLGL::BlendOp::InvSrcColor`,  
`LLGL::BlendOp::SrcAlpha`, `LLGL::BlendOp::InvSrcAlpha`, `LLGL::BlendOp::DestColor`, `LLGL::BlendOp::InvDestColor`,  
`LLGL::BlendOp::DestAlpha`, `LLGL::BlendOp::InvDestAlpha` }

*Blending operations enumeration.*

- enum `LLGL::BlendArithmetic` {  
`LLGL::BlendArithmetic::Add`, `LLGL::BlendArithmetic::Subtract`, `LLGL::BlendArithmetic::RevSubtract`, `LLGL::BlendArithmetic::Min`,  
`LLGL::BlendArithmetic::Max` }

*Blending arithmetic operations enumeration.*

- enum `LLGL::PolygonMode` { `LLGL::PolygonMode::Fill`, `LLGL::PolygonMode::Wireframe`, `LLGL::PolygonMode::Points` }

*Polygon filling modes enumeration.*

- enum `LLGL::CullMode` { `LLGL::CullMode::Disabled`, `LLGL::CullMode::Front`, `LLGL::CullMode::Back` }

*Polygon culling modes enumeration.*

## 8.10 Image.h File Reference

```
#include "Export.h"
#include "RenderSystemFlags.h"
#include "TextureFlags.h"
#include <memory>
```

### Classes

- struct [LLGL::ImageDescriptor](#)  
*Image descriptor structure.*

### Namespaces

- [LLGL](#)

### Typedefs

- using [LLGL::ByteBuffer](#) = std::unique\_ptr< char[]>  
*Common byte buffer type.*

### Enumerations

- enum [LLGL::DataType](#) {  
[LLGL::DataType::Int8](#), [LLGL::DataType::UInt8](#), [LLGL::DataType::Int16](#), [LLGL::DataType::UInt16](#),  
[LLGL::DataType::Int32](#), [LLGL::DataType::UInt32](#), [LLGL::DataType::Float](#), [LLGL::DataType::Double](#) }  
*Renderer data types enumeration.*
- enum [LLGL::ImageFormat](#) {  
[LLGL::ImageFormat::R](#), [LLGL::ImageFormat::RG](#), [LLGL::ImageFormat::RGB](#), [LLGL::ImageFormat::BGR](#),  
[LLGL::ImageFormat::RGBA](#), [LLGL::ImageFormat::BGRA](#), [LLGL::ImageFormat::Depth](#), [LLGL::ImageFormat::DepthStencil](#),  
[LLGL::ImageFormat::CompressedRGB](#), [LLGL::ImageFormat::CompressedRGBA](#) }  
*Image format used to write texture data.*

### Functions

- [LLGL\\_EXPORT](#) std::size\_t [LLGL::DataTypeSize](#) (const DataType dataType)  
*Returns the size (in bytes) of the specified data type.*
- [LLGL\\_EXPORT](#) std::size\_t [LLGL::ImageFormatSize](#) (const ImageFormat imageFormat)  
*Returns the size (in number of components) of the specified image format.*
- [LLGL\\_EXPORT](#) bool [LLGL::IsCompressedFormat](#) (const ImageFormat format)  
*Returns true if the specified color format is a compressed format, i.e. either [ImageFormat::CompressedRGB](#), or [ImageFormat::CompressedRGBA](#).*
- [LLGL\\_EXPORT](#) bool [LLGL::IsDepthStencilFormat](#) (const ImageFormat format)  
*Returns true if the specified color format is a depth-stencil format, i.e. either [ImageFormat::Depth](#) or [ImageFormat::DepthStencil](#).*
- [LLGL\\_EXPORT](#) ByteBuffer [LLGL::ConvertImageBuffer](#) (ImageFormat srcFormat, DataType srcDataType, const void \*srcBuffer, std::size\_t srcBufferSize, ImageFormat dstFormat, DataType dstDataType, std::size\_t threadCount=0)  
*Converts the image format and data type of the source image (only uncompressed color formats).*

## 8.11 IndexBuffer.h File Reference

```
#include "Export.h"
#include "IndexFormat.h"
```

### Classes

- struct [LLGL::IndexBufferDescriptor](#)  
*Index buffer descriptor structure.*
- class [LLGL::IndexBuffer](#)  
*Index buffer interface.*

### Namespaces

- [LLGL](#)

## 8.12 IndexFormat.h File Reference

```
#include "Export.h"
#include "Image.h"
```

### Classes

- class [LLGL::IndexFormat](#)

### Namespaces

- [LLGL](#)

## 8.13 Input.h File Reference

```
#include <LLGL/Window.h>
#include <LLGL/Types.h>
#include <array>
#include <string>
```

### Classes

- class [LLGL::Input](#)

## Namespaces

- [LLGL](#)

## 8.14 Key.h File Reference

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::Key](#) {  
[LLGL::Key::LButton](#), [LLGL::Key::RButton](#), [LLGL::Key::Cancel](#), [LLGL::Key::MButton](#),  
[LLGL::Key::XButton1](#), [LLGL::Key::XButton2](#), [LLGL::Key::Back](#), [LLGL::Key::Tab](#),  
[LLGL::Key::Clear](#), [LLGL::Key::Return](#), [LLGL::Key::Shift](#), [LLGL::Key::Control](#),  
[LLGL::Key::Menu](#), [LLGL::Key::Pause](#), [LLGL::Key::Capital](#), [LLGL::Key::Escape](#),  
[LLGL::Key::Space](#), [LLGL::Key::PageUp](#), [LLGL::Key::PageDown](#), [LLGL::Key::End](#),  
[LLGL::Key::Home](#), [LLGL::Key::Left](#), [LLGL::Key::Up](#), [LLGL::Key::Right](#),  
[LLGL::Key::Down](#), [LLGL::Key::Select](#), [LLGL::Key::Print](#), [LLGL::Key::Exe](#),  
[LLGL::Key::Snapshot](#), [LLGL::Key::Insert](#), [LLGL::Key::Delete](#), [LLGL::Key::Help](#),  
[LLGL::Key::D0](#), [LLGL::Key::D1](#), [LLGL::Key::D2](#), [LLGL::Key::D3](#),  
[LLGL::Key::D4](#), [LLGL::Key::D5](#), [LLGL::Key::D6](#), [LLGL::Key::D7](#),  
[LLGL::Key::D8](#), [LLGL::Key::D9](#), [LLGL::Key::A](#), [LLGL::Key::B](#),  
[LLGL::Key::C](#), [LLGL::Key::D](#), [LLGL::Key::E](#), [LLGL::Key::F](#),  
[LLGL::Key::G](#), [LLGL::Key::H](#), [LLGL::Key::I](#), [LLGL::Key::J](#),  
[LLGL::Key::K](#), [LLGL::Key::L](#), [LLGL::Key::M](#), [LLGL::Key::N](#),  
[LLGL::Key::O](#), [LLGL::Key::P](#), [LLGL::Key::Q](#), [LLGL::Key::R](#),  
[LLGL::Key::S](#), [LLGL::Key::T](#), [LLGL::Key::U](#), [LLGL::Key::V](#),  
[LLGL::Key::W](#), [LLGL::Key::X](#), [LLGL::Key::Y](#), [LLGL::Key::Z](#),  
[LLGL::Key::LWin](#), [LLGL::Key::RWin](#), [LLGL::Key::Apps](#), [LLGL::Key::Sleep](#),  
[LLGL::Key::Keypad0](#), [LLGL::Key::Keypad1](#), [LLGL::Key::Keypad2](#), [LLGL::Key::Keypad3](#),  
[LLGL::Key::Keypad4](#), [LLGL::Key::Keypad5](#), [LLGL::Key::Keypad6](#), [LLGL::Key::Keypad7](#),  
[LLGL::Key::Keypad8](#), [LLGL::Key::Keypad9](#), [LLGL::Key::KeypadMultiply](#), [LLGL::Key::KeypadPlus](#),  
[LLGL::Key::KeypadSeparator](#), [LLGL::Key::KeypadMinus](#), [LLGL::Key::KeypadDecimal](#), [LLGL::Key::Keypad←](#)  
[Divide](#),  
[LLGL::Key::F1](#), [LLGL::Key::F2](#), [LLGL::Key::F3](#), [LLGL::Key::F4](#),  
[LLGL::Key::F5](#), [LLGL::Key::F6](#), [LLGL::Key::F7](#), [LLGL::Key::F8](#),  
[LLGL::Key::F9](#), [LLGL::Key::F10](#), [LLGL::Key::F11](#), [LLGL::Key::F12](#),  
[LLGL::Key::F13](#), [LLGL::Key::F14](#), [LLGL::Key::F15](#), [LLGL::Key::F16](#),  
[LLGL::Key::F17](#), [LLGL::Key::F18](#), [LLGL::Key::F19](#), [LLGL::Key::F20](#),  
[LLGL::Key::F21](#), [LLGL::Key::F22](#), [LLGL::Key::F23](#), [LLGL::Key::F24](#),  
[LLGL::Key::NumLock](#), [LLGL::Key::ScrollLock](#), [LLGL::Key::LShift](#), [LLGL::Key::RShift](#),  
[LLGL::Key::LControl](#), [LLGL::Key::RControl](#), [LLGL::Key::LMenu](#), [LLGL::Key::RMenu](#),  
[LLGL::Key::BrowserBack](#), [LLGL::Key::BrowserForward](#), [LLGL::Key::BrowserRefresh](#), [LLGL::Key::Browser←](#)  
[Stop](#),  
[LLGL::Key::BrowserSearch](#), [LLGL::Key::BrowserFavorites](#), [LLGL::Key::BrowserHome](#), [LLGL::Key::Volume←](#)  
[Mute](#),  
[LLGL::Key::VolumeDown](#), [LLGL::Key::VolumeUp](#), [LLGL::Key::MediaNextTrack](#), [LLGL::Key::MediaPrevTrack](#),  
[LLGL::Key::MediaStop](#), [LLGL::Key::MediaPlayPause](#), [LLGL::Key::LaunchMail](#), [LLGL::Key::LaunchMedia←](#)  
[Select](#),  
[LLGL::Key::LaunchApp1](#), [LLGL::Key::LaunchApp2](#), [LLGL::Key::Plus](#), [LLGL::Key::Comma](#),  
[LLGL::Key::Minus](#), [LLGL::Key::Period](#), [LLGL::Key::Exponent](#), [LLGL::Key::Attn](#),  
[LLGL::Key::CrSel](#), [LLGL::Key::ExSel](#), [LLGL::Key::ErEOF](#), [LLGL::Key::Play](#),  
[LLGL::Key::Zoom](#), [LLGL::Key::NoName](#), [LLGL::Key::PA1](#), [LLGL::Key::OEMClear](#) }

*Input key codes.*

## 8.15 LinuxNativeHandle.h File Reference

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

### Classes

- struct [LLGL::NativeHandle](#)  
*Linux native handle structure.*
- struct [LLGL::NativeContextHandle](#)  
*Linux native context handle structure.*

### Namespaces

- [LLGL](#)

## 8.16 LLGL.h File Reference

```
#include "Window.h"
#include "Input.h"
#include "Timer.h"
#include "RenderSystem.h"
#include "ColorRGB.h"
#include "ColorRGBA.h"
#include "Desktop.h"
```

## 8.17 Log.h File Reference

```
#include "Export.h"
#include <ostream>
```

### Namespaces

- [LLGL](#)
- [LLGL::Log](#)

### Functions

- [LLGL\\_EXPORT](#) void [LLGL::Log::SetStdOut](#) (std::ostream &stream)  
*Sets the standard output stream. By default std::cout.*
- [LLGL\\_EXPORT](#) void [LLGL::Log::SetStdErr](#) (std::ostream &stream)  
*Sets the standard output stream for error and warning messages. By default std::cerr.*
- [LLGL\\_EXPORT](#) std::ostream & [LLGL::Log::StdOut](#) ()  
*Returns the standard output stream.*
- [LLGL\\_EXPORT](#) std::ostream & [LLGL::Log::StdErr](#) ()  
*Returns the standard output stream for error and warning messages.*



## 8.18 MacOSNativeHandle.h File Reference

```
#include <Cocoa/Cocoa.h>
```

### Classes

- struct [LLGL::NativeHandle](#)  
*Linux native handle structure.*
- struct [LLGL::NativeContextHandle](#)  
*Linux native context handle structure.*

### Namespaces

- [LLGL](#)

## 8.19 NativeHandle.h File Reference

## 8.20 Query.h File Reference

```
#include "Export.h"
```

### Classes

- struct [LLGL::QueryDescriptor](#)  
*Query descriptor structure.*
- class [LLGL::Query](#)  
*Query interface.*

### Namespaces

- [LLGL](#)

### Enumerations

- enum [LLGL::QueryType](#) {  
[LLGL::QueryType::SamplesPassed](#),    [LLGL::QueryType::AnySamplesPassed](#),    [LLGL::QueryType::Any↵](#)  
[SamplesPassedConservative](#), [LLGL::QueryType::PrimitivesGenerated](#),  
[LLGL::QueryType::TimeElapsed](#),    [LLGL::QueryType::StreamOutPrimitivesWritten](#),    [LLGL::QueryType::↵](#)  
[StreamOutOverflow](#), [LLGL::QueryType::VerticesSubmitted](#),  
[LLGL::QueryType::PrimitivesSubmitted](#), [LLGL::QueryType::VertexShaderInvocations](#), [LLGL::QueryType::↵](#)  
[TessControlShaderInvocations](#), [LLGL::QueryType::TessEvaluationShaderInvocations](#),  
[LLGL::QueryType::GeometryShaderInvocations](#), [LLGL::QueryType::FragmentShaderInvocations](#), [LLGL::↵](#)  
[QueryType::ComputeShaderInvocations](#), [LLGL::QueryType::GeometryPrimitivesGenerated](#),  
[LLGL::QueryType::ClippingInputPrimitives](#), [LLGL::QueryType::ClippingOutputPrimitives](#) }  
*Query type enumeration.*

## 8.21 RenderContext.h File Reference

```
#include "Export.h"
#include "Window.h"
#include "RenderContextDescriptor.h"
#include "RenderContextFlags.h"
#include "RenderSystemFlags.h"
#include "ColorRGBA.h"
#include "VertexBuffer.h"
#include "IndexBuffer.h"
#include "ConstantBuffer.h"
#include "StorageBuffer.h"
#include "ShaderProgram.h"
#include "Texture.h"
#include "RenderTarget.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "Sampler.h"
#include "Query.h"
#include <Gauss/Vector3.h>
#include <string>
#include <map>
```

### Classes

- class [LLGL::RenderContext](#)  
*Render context interface.*

### Namespaces

- [LLGL](#)

## 8.22 RenderContextDescriptor.h File Reference

```
#include "Export.h"
#include "Types.h"
#include <functional>
```

### Classes

- struct [LLGL::VsyncDescriptor](#)
- struct [LLGL::AntiAliasingDescriptor](#)
- struct [LLGL::VideoModeDescriptor](#)
- struct [LLGL::ProfileOpenGLDescriptor](#)
- struct [LLGL::RenderContextDescriptor](#)

## Namespaces

- [LLGL](#)

## Typedefs

- using [LLGL::DebugCallback](#) = std::function< void(const std::string &type, const std::string &message)>  
*Debug callback function interface.*

## Enumerations

- enum [LLGL::OpenGLVersion](#) {  
[LLGL::OpenGLVersion::OpenGL\\_Latest](#) = 0, [LLGL::OpenGLVersion::OpenGL\\_1\\_0](#) = 100, [LLGL::OpenGLVersion::OpenGL\\_1\\_1](#) = 110, [LLGL::OpenGLVersion::OpenGL\\_1\\_2](#) = 120,  
[LLGL::OpenGLVersion::OpenGL\\_1\\_3](#) = 130, [LLGL::OpenGLVersion::OpenGL\\_1\\_4](#) = 140, [LLGL::OpenGLVersion::OpenGL\\_1\\_5](#) = 150, [LLGL::OpenGLVersion::OpenGL\\_2\\_0](#) = 200,  
[LLGL::OpenGLVersion::OpenGL\\_2\\_1](#) = 210, [LLGL::OpenGLVersion::OpenGL\\_3\\_0](#) = 300, [LLGL::OpenGLVersion::OpenGL\\_3\\_1](#) = 310, [LLGL::OpenGLVersion::OpenGL\\_3\\_2](#) = 320,  
[LLGL::OpenGLVersion::OpenGL\\_3\\_3](#) = 330, [LLGL::OpenGLVersion::OpenGL\\_4\\_0](#) = 400, [LLGL::OpenGLVersion::OpenGL\\_4\\_1](#) = 410, [LLGL::OpenGLVersion::OpenGL\\_4\\_2](#) = 420,  
[LLGL::OpenGLVersion::OpenGL\\_4\\_3](#) = 430, [LLGL::OpenGLVersion::OpenGL\\_4\\_4](#) = 440, [LLGL::OpenGLVersion::OpenGL\\_4\\_5](#) = 450 }
- enum [LLGL::SwapChainMode](#) { [LLGL::SwapChainMode::SingleBuffering](#) = 1, [LLGL::SwapChainMode::DoubleBuffering](#) = 2, [LLGL::SwapChainMode::TripleBuffering](#) = 3 }  
*Swap chain mode enumeration.*

## Functions

- [LLGL\\_EXPORT](#) bool [LLGL::operator==](#) (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)
- [LLGL\\_EXPORT](#) bool [LLGL::operator!=](#) (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)
- [LLGL\\_EXPORT](#) bool [LLGL::operator==](#) (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)
- [LLGL\\_EXPORT](#) bool [LLGL::operator!=](#) (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)

## 8.23 RenderContextFlags.h File Reference

### Classes

- struct [LLGL::ClearBuffersFlags](#)  
*Render context clear buffer flags.*
- struct [LLGL::Viewport](#)  
*Viewport dimensions.*
- struct [LLGL::Scissor](#)  
*Scissor dimensions.*
- union [LLGL::GraphicsAPIDependentStateDescriptor](#)  
*Low-level graphics API dependent state descriptor union.*
- struct [LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor](#)

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::RendererInfo](#) { [LLGL::RendererInfo::Version](#), [LLGL::RendererInfo::Vendor](#), [LLGL::RendererInfo::Hardware](#), [LLGL::RendererInfo::ShadingLanguageVersion](#) }  
*Renderer info enumeration.*
- enum [LLGL::RenderConditionMode](#) { [LLGL::RenderConditionMode::Wait](#), [LLGL::RenderConditionMode::NoWait](#), [LLGL::RenderConditionMode::ByRegionWait](#), [LLGL::RenderConditionMode::ByRegionNoWait](#), [LLGL::RenderConditionMode::WaitInverted](#), [LLGL::RenderConditionMode::NoWaitInverted](#), [LLGL::RenderConditionMode::ByRegionWaitInverted](#), [LLGL::RenderConditionMode::ByRegionNoWaitInverted](#) }  
*Render condition mode enumeration.*

## 8.24 RenderingDebugger.h File Reference

```
#include "Export.h"
#include <map>
#include <string>
```

## Classes

- class [LLGL::RenderingDebugger](#)  
*Rendering debugger interface.*
- class [LLGL::RenderingDebugger::Message](#)  
*Rendering debugger message class.*

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::ErrorType](#) { [LLGL::ErrorType::InvalidArgument](#), [LLGL::ErrorType::InvalidState](#), [LLGL::ErrorType::UnsupportedFeature](#) }  
*Rendering debugger error types enumeration.*
- enum [LLGL::WarningType](#) { [LLGL::WarningType::ImproperArgument](#), [LLGL::WarningType::ImproperState](#), [LLGL::WarningType::PointlessOperation](#) }

## 8.25 RenderingProfiler.h File Reference

```
#include "Export.h"
#include "RenderContextFlags.h"
#include "GraphicsPipelineFlags.h"
```

## Classes

- class [LLGL::RenderingProfiler](#)  
*Rendering profiler model class.*
- class [LLGL::RenderingProfiler::Counter](#)

## Namespaces

- [LLGL](#)

## 8.26 RenderSystem.h File Reference

```
#include "Export.h"
#include "RenderContext.h"
#include "RenderSystemFlags.h"
#include "RenderingProfiler.h"
#include "RenderingDebugger.h"
#include "VertexBuffer.h"
#include "VertexFormat.h"
#include "IndexBuffer.h"
#include "IndexFormat.h"
#include "ConstantBuffer.h"
#include "StorageBuffer.h"
#include "Texture.h"
#include "RenderTarget.h"
#include "ShaderProgram.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "Sampler.h"
#include "Query.h"
#include <string>
#include <memory>
#include <vector>
```

## Classes

- class [LLGL::RenderSystem](#)  
*Render system interface.*

## Namespaces

- [LLGL](#)

## 8.27 RenderSystemFlags.h File Reference

```
#include <Gauss/Vector3.h>
#include "ColorRGBA.h"
#include <cstdint>
```

## Classes

- struct [LLGL::RenderSystemConfiguration](#)  
*Render system configuration structure.*
- struct [LLGL::RenderingCaps](#)  
*Rendering capabilities structure.*

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::BufferUsage](#) { [LLGL::BufferUsage::Static](#), [LLGL::BufferUsage::Dynamic](#) }  
*Hardware buffer usage enumeration.*
- enum [LLGL::BufferCPUAccess](#) { [LLGL::BufferCPUAccess::ReadOnly](#), [LLGL::BufferCPUAccess::WriteOnly](#), [LLGL::BufferCPUAccess::ReadWrite](#) }  
*Hardware buffer CPU access enumeration.*
- enum [LLGL::ShadingLanguage](#) {  
[LLGL::ShadingLanguage::GLSL\\_110](#), [LLGL::ShadingLanguage::GLSL\\_120](#), [LLGL::ShadingLanguage::GLSL\\_130](#), [LLGL::ShadingLanguage::GLSL\\_140](#),  
[LLGL::ShadingLanguage::GLSL\\_150](#), [LLGL::ShadingLanguage::GLSL\\_330](#), [LLGL::ShadingLanguage::GLSL\\_400](#), [LLGL::ShadingLanguage::GLSL\\_410](#),  
[LLGL::ShadingLanguage::GLSL\\_420](#), [LLGL::ShadingLanguage::GLSL\\_430](#), [LLGL::ShadingLanguage::GLSL\\_440](#), [LLGL::ShadingLanguage::GLSL\\_450](#),  
[LLGL::ShadingLanguage::HLSL\\_2\\_0](#), [LLGL::ShadingLanguage::HLSL\\_2\\_0a](#), [LLGL::ShadingLanguage::HLSL\\_2\\_0b](#), [LLGL::ShadingLanguage::HLSL\\_3\\_0](#),  
[LLGL::ShadingLanguage::HLSL\\_4\\_0](#), [LLGL::ShadingLanguage::HLSL\\_4\\_1](#), [LLGL::ShadingLanguage::HLSL\\_5\\_0](#) }  
*Shading language version enumeration.*
- enum [LLGL::ScreenOrigin](#) { [LLGL::ScreenOrigin::LowerLeft](#), [LLGL::ScreenOrigin::UpperLeft](#) }  
*Screen coordinate system origin enumeration.*
- enum [LLGL::ClippingRange](#) { [LLGL::ClippingRange::MinusOneToOne](#), [LLGL::ClippingRange::ZeroToOne](#) }  
*Clipping depth range enumeration.*

## 8.28 RenderTarget.h File Reference

```
#include "Export.h"
#include "TextureFlags.h"
#include <Gauss/Vector2.h>
```

## Classes

- class [LLGL::RenderTarget](#)  
*Render target interface.*

## Namespaces

- [LLGL](#)

## 8.29 Sampler.h File Reference

```
#include "Export.h"
#include "SamplerFlags.h"
```

## Classes

- class [LLGL::Sampler](#)  
*[Sampler](#) interface.*

## Namespaces

- [LLGL](#)

## 8.30 SamplerFlags.h File Reference

```
#include "Export.h"
#include "GraphicsPipelineFlags.h"
#include "ColorRGBA.h"
#include <stddef>
```

## Classes

- struct [LLGL::SamplerDescriptor](#)  
*[Texture](#) sampler descriptor structure.*

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::TextureWrap](#) {  
    [LLGL::TextureWrap::Repeat](#), [LLGL::TextureWrap::Mirror](#), [LLGL::TextureWrap::Clamp](#), [LLGL::TextureWrap::Border](#),  
    [LLGL::TextureWrap::MirrorOnce](#) }  
    *Texture coordinate wrap enumeration.*
- enum [LLGL::TextureFilter](#) { [LLGL::TextureFilter::Nearest](#), [LLGL::TextureFilter::Linear](#) }  
    *Texture sampling filter enumeration.*

## 8.31 Shader.h File Reference

```
#include "Export.h"
#include <string>
```

### Classes

- struct [LLGL::ShaderCompileFlags](#)  
*Shader compilation flags enumeration.*
- struct [LLGL::ShaderDisassembleFlags](#)  
*Shader disassemble flags enumeration.*
- struct [LLGL::ShaderStageFlags](#)  
*Shader stage flags.*
- union [LLGL::ShaderSource](#)  
*Shader source code union.*
- struct [LLGL::ShaderSource::GLSL](#)  
*Shader source descriptor for GLSL.*
- struct [LLGL::ShaderSource::HLSL](#)  
*Shader source descriptor for HLSL.*
- class [LLGL::Shader](#)  
*Shader interface.*

### Namespaces

- [LLGL](#)

### Enumerations

- enum [LLGL::ShaderType](#) {  
[LLGL::ShaderType::Vertex](#), [LLGL::ShaderType::TessControl](#), [LLGL::ShaderType::TessEvaluation](#), [LLGL::ShaderType::Geometry](#),  
[LLGL::ShaderType::Fragment](#), [LLGL::ShaderType::Compute](#) }  
*Shader type enumeration.*

## 8.32 ShaderProgram.h File Reference

```
#include "Export.h"
#include "Shader.h"
#include "VertexAttribute.h"
#include "ConstantBuffer.h"
#include "StorageBuffer.h"
#include "ShaderUniform.h"
#include <string>
#include <vector>
```



## Classes

- class [LLGL::ShaderProgram](#)  
*Shader program interface.*

## Namespaces

- [LLGL](#)

## 8.33 ShaderUniform.h File Reference

```
#include "Export.h"
#include <string>
#include <Gauss/Vector2.h>
#include <Gauss/Vector3.h>
#include <Gauss/Vector4.h>
#include <Gauss/Matrix.h>
```

## Classes

- struct [LLGL::UniformDescriptor](#)  
*Shader uniform descriptor structure.*
- class [LLGL::ShaderUniform](#)  
*Shader uniform setter interface.*

## Namespaces

- [LLGL](#)

## Enumerations

- enum [LLGL::UniformType](#) {  
[LLGL::UniformType::Float](#), [LLGL::UniformType::Float2](#), [LLGL::UniformType::Float3](#), [LLGL::UniformType::↔](#)  
[Float4](#),  
[LLGL::UniformType::Double](#), [LLGL::UniformType::Double2](#), [LLGL::UniformType::Double3](#), [LLGL::Uniform↔](#)  
[Type::Double4](#),  
[LLGL::UniformType::Int](#), [LLGL::UniformType::Int2](#), [LLGL::UniformType::Int3](#), [LLGL::UniformType::Int4](#),  
[LLGL::UniformType::Float2x2](#), [LLGL::UniformType::Float3x3](#), [LLGL::UniformType::Float4x4](#), [LLGL::↔](#)  
[UniformType::Double2x2](#),  
[LLGL::UniformType::Double3x3](#), [LLGL::UniformType::Double4x4](#), [LLGL::UniformType::Sampler1D](#), [LLGL::↔](#)  
[UniformType::Sampler2D](#),  
[LLGL::UniformType::Sampler3D](#), [LLGL::UniformType::SamplerCube](#) }  
*Shader uniform type enumeration.*

## 8.34 StorageBuffer.h File Reference

```
#include "Export.h"
#include "RenderSystemFlags.h"
#include <string>
```

### Classes

- struct [LLGL::StorageBufferDescriptor](#)  
*Storage buffer descriptor structure.*
- struct [LLGL::StorageBufferViewDescriptor](#)  
*Storage buffer shader-view descriptor structure.*
- class [LLGL::StorageBuffer](#)  
*Storage buffer (also called "Shader Storage Buffer Object" or "Read/Write Buffer") interface.*

### Namespaces

- [LLGL](#)

### Enumerations

- enum [LLGL::StorageBufferType](#) {  
[LLGL::StorageBufferType::Buffer](#), [LLGL::StorageBufferType::StructuredBuffer](#), [LLGL::StorageBufferType::↳](#)  
[ByteAddressBuffer](#), [LLGL::StorageBufferType::RWBuffer](#),  
[LLGL::StorageBufferType::RWStructuredBuffer](#), [LLGL::StorageBufferType::RWByteAddressBuffer](#), [LLGL::↳](#)  
[StorageBufferType::AppendStructuredBuffer](#), [LLGL::StorageBufferType::ConsumeStructuredBuffer](#) }  
*Storage buffer type enumeration.*

## 8.35 Texture.h File Reference

```
#include "Export.h"
#include "Image.h"
#include "TextureFlags.h"
#include <Gauss/Vector3.h>
```

### Classes

- class [LLGL::Texture](#)  
*Texture interface.*

### Namespaces

- [LLGL](#)

## 8.36 TextureFlags.h File Reference

```
#include "Export.h"
#include <Gauss/Vector3.h>
#include <cstdint>
```

### Classes

- struct [LLGL::TextureDescriptor](#)  
*Texture descriptor structure.*
- struct [LLGL::TextureDescriptor::Texture1DDescriptor](#)
- struct [LLGL::TextureDescriptor::Texture2DDescriptor](#)
- struct [LLGL::TextureDescriptor::Texture3DDescriptor](#)
- struct [LLGL::TextureDescriptor::TextureCubeDescriptor](#)
- struct [LLGL::SubTextureDescriptor](#)  
*Sub-texture descriptor structure.*
- struct [LLGL::SubTextureDescriptor::Texture1DDescriptor](#)
- struct [LLGL::SubTextureDescriptor::Texture2DDescriptor](#)
- struct [LLGL::SubTextureDescriptor::Texture3DDescriptor](#)
- struct [LLGL::SubTextureDescriptor::TextureCubeDescriptor](#)

### Namespaces

- [LLGL](#)

### Enumerations

- enum [LLGL::TextureType](#) {  
[LLGL::TextureType::Undefined](#), [LLGL::TextureType::Texture1D](#), [LLGL::TextureType::Texture2D](#), [LLGL::TextureType::Texture3D](#),  
[LLGL::TextureType::TextureCube](#), [LLGL::TextureType::Texture1DArray](#), [LLGL::TextureType::Texture2DArray](#),  
[LLGL::TextureType::TextureCubeArray](#) }  
*Texture type enumeration.*
- enum [LLGL::TextureFormat](#) {  
[LLGL::TextureFormat::Unknown](#), [LLGL::TextureFormat::DepthComponent](#), [LLGL::TextureFormat::DepthStencil](#), [LLGL::TextureFormat::R](#),  
[LLGL::TextureFormat::RG](#), [LLGL::TextureFormat::RGB](#), [LLGL::TextureFormat::RGBA](#), [LLGL::TextureFormat::R8](#),  
[LLGL::TextureFormat::R8Sgn](#), [LLGL::TextureFormat::R16](#), [LLGL::TextureFormat::R16Sgn](#), [LLGL::TextureFormat::R16Float](#),  
[LLGL::TextureFormat::R32UInt](#), [LLGL::TextureFormat::R32SInt](#), [LLGL::TextureFormat::R32Float](#), [LLGL::TextureFormat::RG8](#),  
[LLGL::TextureFormat::RG8Sgn](#), [LLGL::TextureFormat::RG16](#), [LLGL::TextureFormat::RG16Sgn](#), [LLGL::TextureFormat::RG16Float](#),  
[LLGL::TextureFormat::RG32UInt](#), [LLGL::TextureFormat::RG32SInt](#), [LLGL::TextureFormat::RG32Float](#), [LLGL::TextureFormat::RGB8](#),  
[LLGL::TextureFormat::RGB8Sgn](#), [LLGL::TextureFormat::RGB16](#), [LLGL::TextureFormat::RGB16Sgn](#), [LLGL::TextureFormat::RGB16Float](#),  
[LLGL::TextureFormat::RGB32UInt](#), [LLGL::TextureFormat::RGB32SInt](#), [LLGL::TextureFormat::RGB32Float](#), [LLGL::TextureFormat::RGBA8](#),  
[LLGL::TextureFormat::RGBA8Sgn](#), [LLGL::TextureFormat::RGBA16](#), [LLGL::TextureFormat::RGBA16Sgn](#),

```

LLGL::TextureFormat::RGBA16Float,
LLGL::TextureFormat::RGBA32UInt, LLGL::TextureFormat::RGBA32SInt, LLGL::TextureFormat::RGBA32F↵
Float, LLGL::TextureFormat::RGB_DXT1,
LLGL::TextureFormat::RGBA_DXT1, LLGL::TextureFormat::RGBA_DXT3, LLGL::TextureFormat::RGBA_↵
DXT5 }

```

*Hardware texture format enumeration.*

- enum [LLGL::AxisDirection](#) {  
[LLGL::AxisDirection::XPos](#) = 0, [LLGL::AxisDirection::XNeg](#), [LLGL::AxisDirection::YPos](#), [LLGL::Axis↵](#)  
[Direction::YNeg](#),  
[LLGL::AxisDirection::ZPos](#), [LLGL::AxisDirection::ZNeg](#) }

*Axis direction (also used for texture cube face).*

## Functions

- [LLGL\\_EXPORT](#) int [LLGL::NumMipLevels](#) (const [Gs::Vector3i](#) &textureSize)  
*Returns the number of MIP-map levels for a texture with the specified size.*
- [LLGL\\_EXPORT](#) bool [LLGL::IsCompressedFormat](#) (const [TextureFormat](#) format)  
*Returns true if the specified texture format is a compressed format, i.e. either [TextureFormat::RGB\\_DXT1](#), [Texture↵](#)  
[Format::RGBA\\_DXT1](#), [TextureFormat::RGBA\\_DXT3](#), or [TextureFormat::RGBA\\_DXT5](#).*

## 8.37 Timer.h File Reference

```

#include <LLGL/Export.h>
#include <memory>

```

### Classes

- class [LLGL::Timer](#)

### Namespaces

- [LLGL](#)

## 8.38 Types.h File Reference

```

#include <Gauss/Vector2.h>

```

### Namespaces

- [LLGL](#)

## Typedefs

- using [LLGL::Point](#) = Gs::Vector2i  
*2D point (integer)*
- using [LLGL::Size](#) = Gs::Vector2i  
*2D size (integer)*

## 8.39 VertexAttribute.h File Reference

```
#include "Image.h"  
#include <string>
```

## Classes

- struct [LLGL::VertexAttribute](#)  
*Vertex attribute class.*

## Namespaces

- [LLGL](#)

## Functions

- [LLGL\\_EXPORT](#) bool [LLGL::operator==](#) (const VertexAttribute &lhs, const VertexAttribute &rhs)
- [LLGL\\_EXPORT](#) bool [LLGL::operator!=](#) (const VertexAttribute &lhs, const VertexAttribute &rhs)

## 8.40 VertexBuffer.h File Reference

```
#include "Export.h"  
#include "VertexFormat.h"
```

## Classes

- struct [LLGL::VertexBufferDescriptor](#)  
*Vertex buffer descriptor structure.*
- class [LLGL::VertexBuffer](#)  
*Vertex buffer interface.*

## Namespaces

- [LLGL](#)

## 8.41 VertexFormat.h File Reference

```
#include "Export.h"
#include "Image.h"
#include "VertexAttribute.h"
#include <vector>
```

### Classes

- class [LLGL::VertexFormat](#)  
*Vertex format descriptor class.*

### Namespaces

- [LLGL](#)

## 8.42 VideoAdapter.h File Reference

```
#include "Export.h"
#include <vector>
#include <string>
```

### Classes

- struct [LLGL::VideoDisplayMode](#)
- struct [LLGL::VideoOutput](#)
- struct [LLGL::VideoAdapterDescriptor](#)  
*Video adapter descriptor structure.*

### Namespaces

- [LLGL](#)

### Functions

- [LLGL\\_EXPORT](#) bool [LLGL::operator==](#) (const VideoDisplayMode &lhs, const VideoDisplayMode &rhs)
- [LLGL\\_EXPORT](#) bool [LLGL::CompareSWO](#) (const VideoDisplayMode &lhs, const VideoDisplayMode &rhs)  
*Compares the two video display modes in a strict-weak-order (SWO) fashion.*

## 8.43 Win32NativeHandle.h File Reference

```
#include <Windows.h>
```

## Classes

- struct [LLGL::NativeHandle](#)  
*Linux native handle structure.*
- struct [LLGL::NativeContextHandle](#)  
*Linux native context handle structure.*

## Namespaces

- [LLGL](#)

## 8.44 Window.h File Reference

```
#include <string>
#include <memory>
#include <vector>
#include <LLGL/Export.h>
#include <LLGL/Key.h>
#include <LLGL/Types.h>
```

## Classes

- struct [LLGL::WindowDescriptor](#)  
*Window descriptor structure.*
- class [LLGL::Window](#)
- class [LLGL::Window::EventListener](#)

## Namespaces

- [LLGL](#)





# Index

- ~ComputePipeline
  - LLGL::ComputePipeline, [57](#)
- ~ConstantBuffer
  - LLGL::ConstantBuffer, [59](#)
- ~EventListener
  - LLGL::Window::EventListener, [63](#)
- ~GraphicsPipeline
  - LLGL::GraphicsPipeline, [66](#)
- ~IndexBuffer
  - LLGL::IndexBuffer, [70](#)
- ~Query
  - LLGL::Query, [79](#)
- ~RenderContext
  - LLGL::RenderContext, [84](#)
- ~RenderSystem
  - LLGL::RenderSystem, [108](#)
- ~RenderTarget
  - LLGL::RenderTarget, [119](#)
- ~RenderingDebugger
  - LLGL::RenderingDebugger, [102](#)
- ~Sampler
  - LLGL::Sampler, [121](#)
- ~Shader
  - LLGL::Shader, [126](#)
- ~ShaderProgram
  - LLGL::ShaderProgram, [129](#)
- ~ShaderSource
  - LLGL::ShaderSource, [134](#)
- ~ShaderUniform
  - LLGL::ShaderUniform, [137](#)
- ~StorageBuffer
  - LLGL::StorageBuffer, [143](#)
- ~SubTextureDescriptor
  - LLGL::SubTextureDescriptor, [146](#)
- ~Texture
  - LLGL::Texture, [148](#)
- ~TextureDescriptor
  - LLGL::TextureDescriptor, [157](#)
- ~Timer
  - LLGL::Timer, [158](#)
- ~VertexBuffer
  - LLGL::VertexBuffer, [162](#)
- ~Window
  - LLGL::Window, [173](#)
- A
  - LLGL, [27](#)
- a
  - LLGL::Color< T, 4u >, [57](#)
- acceptDropFiles
  - LLGL::WindowDescriptor, [175](#)
- Add
  - LLGL, [23](#)
- AddAttribute
  - LLGL::VertexFormat, [164](#), [165](#)
- AddEventListener
  - LLGL::Window, [173](#)
- AllGraphicsStages
  - LLGL::ShaderStageFlags, [135](#)
- AllStages
  - LLGL::ShaderStageFlags, [135](#)
- AllTessStages
  - LLGL::ShaderStageFlags, [135](#)
- alphaArithmetic
  - LLGL::BlendTargetDescriptor, [47](#)
- antiAliasedLineEnabled
  - LLGL::RasterizerDescriptor, [81](#)
- antiAliasing
  - LLGL::RenderContextDescriptor, [96](#)
- AnySamplesPassed
  - LLGL, [32](#)
- AnySamplesPassedConservative
  - LLGL, [32](#)
- AppendStructuredBuffer
  - LLGL, [35](#)
- ApplyMipResolution
  - LLGL::RenderTarget, [119](#)
- ApplyResolution
  - LLGL::RenderTarget, [119](#)
- Apps
  - LLGL, [28](#)
- AttachDepthBuffer
  - LLGL::RenderTarget, [119](#)
- AttachDepthStencilBuffer
  - LLGL::RenderTarget, [119](#)
- AttachShader
  - LLGL::ShaderProgram, [129](#)
- AttachStencilBuffer
  - LLGL::RenderTarget, [120](#)
- AttachTexture1DArray
  - LLGL::RenderTarget, [120](#)
- AttachTexture1D
  - LLGL::RenderTarget, [120](#)
- AttachTexture2DArray
  - LLGL::RenderTarget, [120](#)
- AttachTexture2D
  - LLGL::RenderTarget, [120](#)
- AttachTexture3D
  - LLGL::RenderTarget, [120](#)

- AttachTextureCube
  - LLGL::RenderTarget, [120](#)
- AttachTextureCubeArray
  - LLGL::RenderTarget, [120](#)
- Attn
  - LLGL, [29](#)
- AxisDirection
  - LLGL, [23](#)
- B
  - LLGL, [27](#)
- b
  - LLGL::Color< T, 3u >, [54](#)
  - LLGL::Color< T, 4u >, [57](#)
- BGRA
  - LLGL, [26](#)
- BGR
  - LLGL, [26](#)
- Back
  - LLGL, [25](#), [26](#)
- back
  - LLGL::StencilDescriptor, [141](#)
- BeginQuery
  - LLGL::RenderContext, [84](#)
- BeginRenderCondition
  - LLGL::RenderContext, [85](#)
- BindConstantBuffer
  - LLGL::ShaderProgram, [130](#)
- BindStorageBuffer
  - LLGL::ShaderProgram, [130](#)
- BindVertexAttributes
  - LLGL::ShaderProgram, [131](#)
- blend
  - LLGL::GraphicsPipelineDescriptor, [66](#)
- BlendArithmetic
  - LLGL, [23](#)
- blendEnabled
  - LLGL::BlendDescriptor, [46](#)
- BlendOp
  - LLGL, [23](#)
- Block
  - LLGL::RenderingDebugger::Message, [75](#)
- BlockAfter
  - LLGL::RenderingDebugger::Message, [75](#)
- Border
  - LLGL, [37](#)
- borderColor
  - LLGL::SamplerDescriptor, [122](#)
- borderless
  - LLGL::WindowDescriptor, [175](#)
- BrowserBack
  - LLGL, [29](#)
- BrowserFavorites
  - LLGL, [29](#)
- BrowserForward
  - LLGL, [29](#)
- BrowserHome
  - LLGL, [29](#)
- BrowserRefresh
  - LLGL, [29](#)
- BrowserSearch
  - LLGL, [29](#)
- BrowserStop
  - LLGL, [29](#)
- Buffer
  - LLGL, [35](#)
- buffer
  - LLGL::ImageDescriptor, [69](#)
- BufferCPUAccess
  - LLGL, [23](#)
- BufferUsage
  - LLGL, [24](#)
- ByRegionNoWait
  - LLGL, [33](#)
- ByRegionNoWaitInverted
  - LLGL, [33](#)
- ByRegionWait
  - LLGL, [33](#)
- ByRegionWaitInverted
  - LLGL, [33](#)
- ByteAddressBuffer
  - LLGL, [35](#)
- ByteBuffer
  - LLGL, [21](#)
- C
  - LLGL, [27](#)
- Cancel
  - LLGL, [26](#)
- Capital
  - LLGL, [27](#)
- Cast
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [52](#)
  - LLGL::Color< T, 4u >, [55](#)
- centered
  - LLGL::WindowDescriptor, [175](#)
- Clamp
  - LLGL, [37](#)
- Clear
  - LLGL, [26](#)
- ClearBuffers
  - LLGL::RenderContext, [85](#)
- ClippingInputPrimitives
  - LLGL, [32](#)
- ClippingOutputPrimitives
  - LLGL, [32](#)
- ClippingRange
  - LLGL, [24](#)
- clippingRange
  - LLGL::RenderingCaps, [98](#)
- Color
  - LLGL::ClearBuffersFlags, [48](#)
  - LLGL::Color, [49](#)
  - LLGL::Color< T, 3u >, [52](#)
  - LLGL::Color< T, 4u >, [55](#)
- Color.h, [177](#)
- colorArithmetic

- LLGL::BlendTargetDescriptor, [47](#)
- colorDepth
  - LLGL::VideoModeDescriptor, [168](#)
- colorMap
  - LLGL::NativeContextHandle, [76](#)
- colorMask
  - LLGL::BlendTargetDescriptor, [47](#)
- ColorRGB.h, [178](#)
- ColorRGBA.h, [178](#)
- ColorRGBAb
  - LLGL, [22](#)
- ColorRGBAd
  - LLGL, [22](#)
- ColorRGBAf
  - LLGL, [22](#)
- ColorRGBAT
  - LLGL, [22](#)
- ColorRGBAub
  - LLGL, [22](#)
- ColorRGBA
  - LLGL, [22](#)
- ColorRGBb
  - LLGL, [22](#)
- ColorRGBd
  - LLGL, [22](#)
- ColorRGBf
  - LLGL, [22](#)
- ColorRGBT
  - LLGL, [22](#)
- ColorRGBub
  - LLGL, [22](#)
- ColorRGB
  - LLGL, [21](#)
- Comma
  - LLGL, [29](#)
- compareMask
  - LLGL::StencilFaceDescriptor, [142](#)
- CompareOp
  - LLGL, [24](#)
- compareOp
  - LLGL::DepthDescriptor, [62](#)
  - LLGL::SamplerDescriptor, [122](#)
  - LLGL::StencilFaceDescriptor, [142](#)
- CompareSWO
  - LLGL, [38](#)
- Compile
  - LLGL::Shader, [126](#)
- components
  - LLGL::Color, [51](#)
  - LLGL::Color< T, 3u >, [54](#)
  - LLGL::Color< T, 4u >, [57](#)
  - LLGL::VertexAttribute, [161](#)
- CompressedRGBA
  - LLGL, [26](#)
- CompressedRGB
  - LLGL, [26](#)
- compressedSize
  - LLGL::ImageDescriptor, [69](#)
- Compute
  - LLGL, [34](#)
- ComputePipeline.h, [179](#)
- ComputePipelineDescriptor
  - LLGL::ComputePipelineDescriptor, [58](#)
- ComputeShaderInvocations
  - LLGL, [32](#)
- ComputeStage
  - LLGL::ShaderStageFlags, [135](#)
- conservativeRasterization
  - LLGL::RasterizerDescriptor, [81](#)
- ConstantBuffer.h, [179](#)
- ConstantBufferDescriptor
  - LLGL::ConstantBufferDescriptor, [60](#)
- ConsumeStructuredBuffer
  - LLGL, [35](#)
- Control
  - LLGL, [26](#)
- conversion
  - LLGL::VertexAttribute, [161](#)
- ConvertImageBuffer
  - LLGL, [38](#)
- coreProfile
  - LLGL::ProfileOpenGLDescriptor, [78](#)
- Count
  - LLGL::RenderingProfiler::Counter, [61](#)
- CrSel
  - LLGL, [29](#)
- Create
  - LLGL::Timer, [158](#)
  - LLGL::Window, [173](#)
- CreateComputePipeline
  - LLGL::RenderSystem, [108](#)
- CreateConstantBuffer
  - LLGL::RenderSystem, [108](#)
- CreateGraphicsPipeline
  - LLGL::RenderSystem, [109](#)
- CreateIndexBuffer
  - LLGL::RenderSystem, [109](#)
- CreateQuery
  - LLGL::RenderSystem, [109](#)
- CreateRenderContext
  - LLGL::RenderSystem, [110](#)
- CreateRenderTarget
  - LLGL::RenderSystem, [110](#)
- CreateSampler
  - LLGL::RenderSystem, [110](#)
- CreateShader
  - LLGL::RenderSystem, [110](#)
- CreateShaderProgram
  - LLGL::RenderSystem, [111](#)
- CreateStorageBuffer
  - LLGL::RenderSystem, [111](#)
- CreateTexture
  - LLGL::RenderSystem, [111](#)
- CreateVertexBuffer
  - LLGL::RenderSystem, [112](#)
- cubeFaceOffset

- LLGL::SubTextureDescriptor::TextureCube↔  
Descriptor, [154](#)
- cubeFaces
  - LLGL::SubTextureDescriptor::TextureCube↔  
Descriptor, [154](#)
- CullMode
  - LLGL, [25](#)
- cullMode
  - LLGL::RasterizerDescriptor, [81](#)
- D
  - LLGL, [27](#)
- D0
  - LLGL, [27](#)
- D1
  - LLGL, [27](#)
- D2
  - LLGL, [27](#)
- D3
  - LLGL, [27](#)
- D4
  - LLGL, [27](#)
- D5
  - LLGL, [27](#)
- D6
  - LLGL, [27](#)
- D7
  - LLGL, [27](#)
- D8
  - LLGL, [27](#)
- D9
  - LLGL, [27](#)
- DataType
  - LLGL, [25](#)
- dataType
  - LLGL::ImageDescriptor, [69](#)
  - LLGL::VertexAttribute, [161](#)
- DataTypeSize
  - LLGL, [39](#)
- Debug
  - LLGL::ShaderCompileFlags, [127](#)
- DebugCallback
  - LLGL, [22](#)
- debugCallback
  - LLGL::RenderContextDescriptor, [96](#)
- debugDump
  - LLGL::ProfileOpenGLDescriptor, [78](#)
- DecClamp
  - LLGL, [34](#)
- DecWrap
  - LLGL, [34](#)
- defaultImageColor
  - LLGL::RenderSystemConfiguration, [118](#)
- Delete
  - LLGL, [27](#)
- Depth
  - LLGL::ClearBuffersFlags, [48](#)
  - LLGL, [26](#)
- depth
  - LLGL::GraphicsPipelineDescriptor, [66](#)
  - LLGL::SubTextureDescriptor::Texture3DDescriptor,  
[153](#)
  - LLGL::TextureDescriptor::Texture3DDescriptor,  
[152](#)
- depthBias
  - LLGL::RasterizerDescriptor, [81](#)
- depthBiasClamp
  - LLGL::RasterizerDescriptor, [81](#)
- depthClampEnabled
  - LLGL::RasterizerDescriptor, [81](#)
- depthCompare
  - LLGL::SamplerDescriptor, [122](#)
- DepthComponent
  - LLGL, [36](#)
- depthFailOp
  - LLGL::StencilFaceDescriptor, [142](#)
- depthPassOp
  - LLGL::StencilFaceDescriptor, [142](#)
- DepthStencil
  - LLGL, [26](#), [36](#)
- Desktop.h, [180](#)
- DestAlpha
  - LLGL, [23](#)
- destAlpha
  - LLGL::BlendTargetDescriptor, [47](#)
- DestColor
  - LLGL, [23](#)
- destColor
  - LLGL::BlendTargetDescriptor, [47](#)
- DetachTextures
  - LLGL::RenderTarget, [120](#)
- Disabled
  - LLGL, [25](#)
- Disassemble
  - LLGL::Shader, [126](#)
- DispatchCompute
  - LLGL::RenderContext, [86](#)
- dispatchComputeCalls
  - LLGL::RenderingProfiler, [105](#)
- display
  - LLGL::NativeContextHandle, [76](#)
  - LLGL::NativeHandle, [77](#)
- displayModes
  - LLGL::VideoOutput, [169](#)
- Double
  - LLGL, [25](#), [38](#)
- Double2
  - LLGL, [38](#)
- Double2x2
  - LLGL, [38](#)
- Double3
  - LLGL, [38](#)
- Double3x3
  - LLGL, [38](#)
- Double4
  - LLGL, [38](#)
- Double4x4

- LLGL, [38](#)
- DoubleBuffering
  - LLGL, [35](#)
- Down
  - LLGL, [27](#)
- Draw
  - LLGL::RenderContext, [86](#)
- drawCalls
  - LLGL::RenderingProfiler, [105](#)
- DrawIndexed
  - LLGL::RenderContext, [86](#)
- DrawIndexedInstanced
  - LLGL::RenderContext, [87](#)
- DrawInstanced
  - LLGL::RenderContext, [87](#)
- Dynamic
  - LLGL, [24](#)
- E
  - LLGL, [27](#)
- enabled
  - LLGL::AntiAliasingDescriptor, [45](#)
  - LLGL::VsyncDescriptor, [171](#)
- End
  - LLGL, [27](#)
- EndQuery
  - LLGL::RenderContext, [88](#)
- EndRenderCondition
  - LLGL::RenderContext, [88](#)
- entryPoint
  - LLGL::ShaderSource::HLSL, [68](#)
- Equal
  - LLGL, [25](#)
- ErEOF
  - LLGL, [29](#)
- ErrorType
  - LLGL, [25](#)
- Escape
  - LLGL, [27](#)
- Ever
  - LLGL, [25](#)
- ExSel
  - LLGL, [29](#)
- Exe
  - LLGL, [27](#)
- Exponent
  - LLGL, [29](#)
- Export.h, [180](#)
  - LLGL\_EXPORT, [180](#)
- extProfile
  - LLGL::ProfileOpenGLDescriptor, [78](#)
- F
  - LLGL, [27](#)
- F1
  - LLGL, [28](#)
- F10
  - LLGL, [28](#)
- F11
  - LLGL, [28](#)
- F12
  - LLGL, [28](#)
- F13
  - LLGL, [28](#)
- F14
  - LLGL, [28](#)
- F15
  - LLGL, [29](#)
- F16
  - LLGL, [29](#)
- F17
  - LLGL, [29](#)
- F18
  - LLGL, [29](#)
- F19
  - LLGL, [29](#)
- F2
  - LLGL, [28](#)
- F20
  - LLGL, [29](#)
- F21
  - LLGL, [29](#)
- F22
  - LLGL, [29](#)
- F23
  - LLGL, [29](#)
- F24
  - LLGL, [29](#)
- F3
  - LLGL, [28](#)
- F4
  - LLGL, [28](#)
- F5
  - LLGL, [28](#)
- F6
  - LLGL, [28](#)
- F7
  - LLGL, [28](#)
- F8
  - LLGL, [28](#)
- F9
  - LLGL, [28](#)
- Fill
  - LLGL, [30](#)
- FindModules
  - LLGL::RenderSystem, [112](#)
- flags
  - LLGL::ShaderSource::HLSL, [68](#)
- Float
  - LLGL, [25](#), [38](#)
- Float2
  - LLGL, [38](#)
- Float2x2
  - LLGL, [38](#)
- Float3
  - LLGL, [38](#)
- Float3x3

- LLGL, [38](#)
- Float4
  - LLGL, [38](#)
- Float4x4
  - LLGL, [38](#)
- format
  - LLGL::ImageDescriptor, [70](#)
  - LLGL::TextureDescriptor, [157](#)
- Fragment
  - LLGL, [34](#)
- FragmentShaderInvocations
  - LLGL, [32](#)
- FragmentStage
  - LLGL::ShaderStageFlags, [135](#)
- FrameCount
  - LLGL::Timer, [158](#)
- Front
  - LLGL, [25](#)
- front
  - LLGL::StencilDescriptor, [141](#)
- frontCCW
  - LLGL::RasterizerDescriptor, [81](#)
- fullscreen
  - LLGL::VideoModeDescriptor, [168](#)
- G
  - LLGL, [27](#)
- g
  - LLGL::Color< T, 3u >, [54](#)
  - LLGL::Color< T, 4u >, [57](#)
- GLSL\_110
  - LLGL, [34](#)
- GLSL\_120
  - LLGL, [34](#)
- GLSL\_130
  - LLGL, [34](#)
- GLSL\_140
  - LLGL, [34](#)
- GLSL\_150
  - LLGL, [34](#)
- GLSL\_330
  - LLGL, [34](#)
- GLSL\_400
  - LLGL, [34](#)
- GLSL\_410
  - LLGL, [34](#)
- GLSL\_420
  - LLGL, [34](#)
- GLSL\_430
  - LLGL, [34](#)
- GLSL\_440
  - LLGL, [34](#)
- GLSL\_450
  - LLGL, [34](#)
- GenerateMips
  - LLGL::RenderSystem, [112](#)
- Geometry
  - LLGL, [34](#)
- GeometryPrimitivesGenerated
  - LLGL, [32](#)
- GeometryShaderInvocations
  - LLGL, [32](#)
- GeometryStage
  - LLGL::ShaderStageFlags, [135](#)
- GetAttributes
  - LLGL::VertexFormat, [165](#)
- GetColorDepth
  - LLGL::Desktop, [42](#)
- GetConfiguration
  - LLGL::RenderSystem, [112](#)
- GetCurrentContext
  - LLGL::RenderSystem, [112](#)
- GetDataType
  - LLGL::IndexFormat, [72](#)
- GetDefaultTextureImageRGBAub
  - LLGL::RenderSystem, [112](#)
- GetDeltaTime
  - LLGL::Timer, [158](#)
- GetEnteredChars
  - LLGL::Input, [73](#)
- GetFormatSize
  - LLGL::IndexFormat, [72](#)
  - LLGL::VertexFormat, [166](#)
- GetFrameCount
  - LLGL::Timer, [159](#)
- GetFrequency
  - LLGL::Timer, [159](#)
- GetIndexFormat
  - LLGL::IndexBuffer, [70](#)
- GetMouseMotion
  - LLGL::Input, [73](#)
- GetMousePosition
  - LLGL::Input, [73](#)
- GetName
  - LLGL::RenderSystem, [113](#)
- GetNativeHandle
  - LLGL::Window, [173](#)
- GetOccurrences
  - LLGL::RenderingDebugger::Message, [75](#)
- GetPosition
  - LLGL::Window, [173](#)
- GetResolution
  - LLGL::Desktop, [42](#)
  - LLGL::RenderTarget, [120](#)
- GetSize
  - LLGL::Window, [173](#)
- GetSource
  - LLGL::RenderingDebugger::Message, [75](#)
- GetText
  - LLGL::RenderingDebugger::Message, [75](#)
- GetTitle
  - LLGL::Window, [173](#)
- GetType
  - LLGL::Query, [79](#)
  - LLGL::Shader, [126](#)
  - LLGL::Texture, [148](#)
- GetVertexFormat

- LLGL::VertexBuffer, 162
- GetVideoMode
  - LLGL::RenderContext, 88
- GetWheelMotion
  - LLGL::Input, 74
- GetWindow
  - LLGL::RenderContext, 88
- GraphicsAPIDependentStateDescriptor
  - LLGL::GraphicsAPIDependentStateDescriptor, 65
- GraphicsPipeline.h, 180
- GraphicsPipelineFlags.h, 181
- Greater
  - LLGL, 25
- GreaterEqual
  - LLGL, 25
- H
  - LLGL, 27
- HLSL\_2\_0
  - LLGL, 34
- HLSL\_2\_0a
  - LLGL, 34
- HLSL\_2\_0b
  - LLGL, 34
- HLSL\_3\_0
  - LLGL, 34
- HLSL\_4\_0
  - LLGL, 34
- HLSL\_4\_1
  - LLGL, 34
- HLSL\_5\_0
  - LLGL, 34
- Hardware
  - LLGL, 33
- has3DTextures
  - LLGL::RenderingCaps, 98
- hasComputeShaders
  - LLGL::RenderingCaps, 98
- hasConservativeRasterization
  - LLGL::RenderingCaps, 98
- hasConstantBuffers
  - LLGL::RenderingCaps, 98
- hasCubeTextureArrays
  - LLGL::RenderingCaps, 99
- hasCubeTextures
  - LLGL::RenderingCaps, 99
- hasGLSL
  - LLGL::RenderingCaps, 99
- hasGeometryShaders
  - LLGL::RenderingCaps, 99
- hasHLSL
  - LLGL::RenderingCaps, 99
- hasInstancing
  - LLGL::RenderingCaps, 99
- hasOffsetInstancing
  - LLGL::RenderingCaps, 99
- hasRenderTargets
  - LLGL::RenderingCaps, 99
- hasSamplers
  - LLGL::RenderingCaps, 99
- hasStorageBuffers
  - LLGL::RenderingCaps, 100
- hasTessellationShaders
  - LLGL::RenderingCaps, 100
- hasTextureArrays
  - LLGL::RenderingCaps, 100
- hasUniforms
  - LLGL::RenderingCaps, 100
- hasViewportArrays
  - LLGL::RenderingCaps, 100
- height
  - LLGL::Scissor, 125
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, 151
  - LLGL::SubTextureDescriptor::Texture3DDescriptor, 153
  - LLGL::SubTextureDescriptor::TextureCubeDescriptor, 154
  - LLGL::TextureDescriptor::Texture2DDescriptor, 150
  - LLGL::TextureDescriptor::Texture3DDescriptor, 152
  - LLGL::TextureDescriptor::TextureCubeDescriptor, 155
  - LLGL::VideoDisplayMode, 167
  - LLGL::Viewport, 170
- Help
  - LLGL, 27
- Home
  - LLGL, 27
- I
  - LLGL, 27
- Image.h, 183
- ImageDescriptor
  - LLGL::ImageDescriptor, 69
- ImageFormat
  - LLGL, 26
- ImageFormatSize
  - LLGL, 39
- ImproperArgument
  - LLGL, 38
- ImproperState
  - LLGL, 38
- Inc
  - LLGL::RenderingProfiler::Counter, 61
- IncClamp
  - LLGL, 34
- IncOccurrence
  - LLGL::RenderingDebugger::Message, 75
- IncWrap
  - LLGL, 34
- index
  - LLGL::ConstantBufferViewDescriptor, 61
  - LLGL::StorageBufferViewDescriptor, 145
- IndexBuffer.h, 184
- IndexBufferDescriptor
  - LLGL::IndexBufferDescriptor, 71

- IndexFormat
  - LLGL::IndexFormat, 72
- indexFormat
  - LLGL::IndexBufferDescriptor, 71
- IndexFormat.h, 184
- Input
  - LLGL::Input, 73
- Input.h, 184
- Insert
  - LLGL, 27
- InstructionOnly
  - LLGL::ShaderDisassembleFlags, 128
- Int
  - LLGL, 38
- Int16
  - LLGL, 25
- Int2
  - LLGL, 38
- Int3
  - LLGL, 38
- Int32
  - LLGL, 25
- Int4
  - LLGL, 38
- Int8
  - LLGL, 25
- interval
  - LLGL::VsyncDescriptor, 171
- InvDestAlpha
  - LLGL, 23
- InvDestColor
  - LLGL, 23
- InvSrcAlpha
  - LLGL, 23
- InvSrcColor
  - LLGL, 23
- InvalidArgument
  - LLGL, 26
- InvalidState
  - LLGL, 26
- Invert
  - LLGL, 34
- invertFrontFace
  - LLGL::GraphicsAPIDependentStateDescriptor::↔  
StateOpenGLDescriptor, 140
- IsBlocked
  - LLGL::RenderingDebugger::Message, 75
- IsCompressedFormat
  - LLGL, 40
- IsDepthStencilFormat
  - LLGL, 40
- IsShown
  - LLGL::Window, 173
- J
  - LLGL, 27
- K
  - LLGL, 27
- Keep
  - LLGL, 34
- Key
  - LLGL, 26
- Key.h, 185
- KeyDoubleClick
  - LLGL::Input, 74
- KeyDown
  - LLGL::Input, 74
- KeyPressed
  - LLGL::Input, 74
- KeyUp
  - LLGL::Input, 74
- Keypad0
  - LLGL, 28
- Keypad1
  - LLGL, 28
- Keypad2
  - LLGL, 28
- Keypad3
  - LLGL, 28
- Keypad4
  - LLGL, 28
- Keypad5
  - LLGL, 28
- Keypad6
  - LLGL, 28
- Keypad7
  - LLGL, 28
- Keypad8
  - LLGL, 28
- Keypad9
  - LLGL, 28
- KeypadDecimal
  - LLGL, 28
- KeypadDivide
  - LLGL, 28
- KeypadMinus
  - LLGL, 28
- KeypadMultiply
  - LLGL, 28
- KeypadPlus
  - LLGL, 28
- KeypadSeparator
  - LLGL, 28
- L
  - LLGL, 27
- LButton
  - LLGL, 26
- LControl
  - LLGL, 29
- LLGL.h, 186
- LLGL::AntiAliasingDescriptor, 45
  - enabled, 45
  - samples, 45
- LLGL::BlendDescriptor, 45
  - blendEnabled, 46
  - targets, 46



- LLGL::BlendTargetDescriptor, 46
  - alphaArithmetic, 47
  - colorArithmetic, 47
  - colorMask, 47
  - destAlpha, 47
  - destColor, 47
  - srcAlpha, 47
  - srcColor, 47
- LLGL::ClearBuffersFlags, 48
  - Color, 48
  - Depth, 48
  - Stencil, 48
- LLGL::Color
  - Cast, 50
  - Color, 49
  - components, 51
  - operator\*=, 50
  - operator+=, 50
  - operator-, 50
  - operator-=, 50
  - operator/=: 50
  - operator[], 50
  - Ptr, 51
- LLGL::Color< T, 3u >, 51
  - b, 54
  - Cast, 52
  - Color, 52
  - components, 54
  - g, 54
  - operator\*=, 53
  - operator+=, 53
  - operator-, 53
  - operator-=, 53
  - operator/=: 53
  - operator[], 53
  - Ptr, 53
  - r, 54
- LLGL::Color< T, 4u >, 54
  - a, 57
  - b, 57
  - Cast, 55
  - Color, 55
  - components, 57
  - g, 57
  - operator\*=, 56
  - operator+=, 56
  - operator-, 56
  - operator-=, 56
  - operator/=: 56
  - operator[], 56
  - Ptr, 56
  - r, 57
- LLGL::Color< T, N >, 48
- LLGL::ComputePipeline, 57
  - ~ComputePipeline, 57
- LLGL::ComputePipelineDescriptor, 58
  - ComputePipelineDescriptor, 58
  - shaderProgram, 58
- LLGL::ConstantBuffer, 59
  - ~ConstantBuffer, 59
- LLGL::ConstantBufferDescriptor, 59
  - ConstantBufferDescriptor, 60
  - size, 60
  - usage, 60
- LLGL::ConstantBufferViewDescriptor, 60
  - index, 61
  - name, 61
  - size, 61
- LLGL::DepthDescriptor, 62
  - compareOp, 62
  - testEnabled, 62
  - writeEnabled, 62
- LLGL::Desktop, 41
  - GetColorDepth, 42
  - GetResolution, 42
  - ResetVideoMode, 42
  - SetVideoMode, 42
- LLGL::GraphicsAPIDependentStateDescriptor, 64
  - GraphicsAPIDependentStateDescriptor, 65
  - stateOpenGL, 65
- LLGL::GraphicsAPIDependentStateDescriptor::State↔
  - OpenGLDescriptor, 139
  - invertFrontFace, 140
  - screenSpaceOriginLowerLeft, 140
- LLGL::GraphicsPipeline, 65
  - ~GraphicsPipeline, 66
- LLGL::GraphicsPipelineDescriptor, 66
  - blend, 66
  - depth, 66
  - primitiveTopology, 67
  - rasterizer, 67
  - shaderProgram, 67
  - stencil, 67
- LLGL::ImageDescriptor, 68
  - buffer, 69
  - compressedSize, 69
  - dataType, 69
  - format, 70
  - ImageDescriptor, 69
- LLGL::IndexBuffer, 70
  - ~IndexBuffer, 70
  - GetIndexFormat, 70
  - SetIndexFormat, 70
- LLGL::IndexBufferDescriptor, 71
  - IndexBufferDescriptor, 71
  - indexFormat, 71
  - size, 71
  - usage, 72
- LLGL::IndexFormat, 72
  - GetDataType, 72
  - GetFormatSize, 72
  - IndexFormat, 72
- LLGL::Input, 73
  - GetEnteredChars, 73
  - GetMouseMotion, 73
  - GetMousePosition, 73

- GetWheelMotion, 74
- Input, 73
- KeyDoubleClick, 74
- KeyDown, 74
- KeyPressed, 74
- KeyUp, 74
- LLGL::Log, 42
  - SetStdErr, 42
  - SetStdOut, 42
  - StdErr, 43
  - StdOut, 43
- LLGL::NativeContextHandle, 76
  - colorMap, 76
  - display, 76
  - parentWindow, 76
  - screen, 76
  - visual, 76
- LLGL::NativeHandle, 77
  - display, 77
  - visual, 77
  - window, 77
- LLGL::ProfileOpenGLDescriptor, 77
  - coreProfile, 78
  - debugDump, 78
  - extProfile, 78
  - version, 78
- LLGL::Query, 78
  - ~Query, 79
  - GetType, 79
  - operator=, 79
  - Query, 79
- LLGL::QueryDescriptor, 79
  - renderCondition, 80
  - type, 80
- LLGL::RasterizerDescriptor, 80
  - antiAliasedLineEnabled, 81
  - conservativeRasterization, 81
  - cullMode, 81
  - depthBias, 81
  - depthBiasClamp, 81
  - depthClampEnabled, 81
  - frontCCW, 81
  - multiSampleEnabled, 81
  - polygonMode, 81
  - samples, 81
  - scissorTestEnabled, 82
  - slopeScaledDepthBias, 82
- LLGL::RenderContext, 82
  - ~RenderContext, 84
  - BeginQuery, 84
  - BeginRenderCondition, 85
  - ClearBuffers, 85
  - DispatchCompute, 86
  - Draw, 86
  - DrawIndexed, 86
  - DrawIndexedInstanced, 87
  - DrawInstanced, 87
  - EndQuery, 88
  - EndRenderCondition, 88
  - GetVideoMode, 88
  - GetWindow, 88
  - MapStorageBuffer, 88
  - operator=, 89
  - Present, 89
  - QueryResult, 89
  - RenderContext, 84
  - SetClearColor, 89
  - SetClearDepth, 89
  - SetClearStencil, 89
  - SetComputePipeline, 90
  - SetConstantBuffer, 90
  - SetGraphicsAPIDependentState, 90
  - SetGraphicsPipeline, 91
  - SetIndexBuffer, 91
  - SetRenderTarget, 91
  - SetSampler, 92
  - SetScissor, 92
  - SetScissorArray, 92
  - SetStorageBuffer, 93
  - SetTexture, 93
  - SetVertexBuffer, 93
  - SetVideoMode, 94
  - SetViewport, 94
  - SetViewportArray, 94
  - SetVsync, 95
  - SetWindow, 95
  - ShareWindowAndVideoMode, 95
  - SyncGPU, 95
  - UnmapStorageBuffer, 95
  - UnsetRenderTarget, 95
- LLGL::RenderContextDescriptor, 96
  - antiAliasing, 96
  - debugCallback, 96
  - profileOpenGL, 96
  - videoMode, 96
  - vsync, 96
- LLGL::RenderSystem, 106
  - ~RenderSystem, 108
  - CreateComputePipeline, 108
  - CreateConstantBuffer, 108
  - CreateGraphicsPipeline, 109
  - CreateIndexBuffer, 109
  - CreateQuery, 109
  - CreateRenderContext, 110
  - CreateRenderTarget, 110
  - CreateSampler, 110
  - CreateShader, 110
  - CreateShaderProgram, 111
  - CreateStorageBuffer, 111
  - CreateTexture, 111
  - CreateVertexBuffer, 112
  - FindModules, 112
  - GenerateMips, 112
  - GetConfiguration, 112
  - GetCurrentContext, 112
  - GetDefaultTextureImageRGBAub, 112

- GetName, 113
- Load, 113
- MakeCurrent, 113
- OnMakeCurrent, 114
- operator=, 114
- QueryRendererInfo, 114
- QueryRenderingCaps, 114
- QueryShadingLanguage, 114
- QueryTextureDescriptor, 114
- ReadTexture, 114
- Release, 115, 116
- RenderSystem, 108
- SetConfiguration, 116
- WriteConstantBuffer, 116
- WriteIndexBuffer, 116
- WriteStorageBuffer, 116
- WriteTexture, 116
- WriteVertexBuffer, 117
- LLGL::RenderSystemConfiguration, 117
  - defaultImageColor, 118
  - threadCount, 118
- LLGL::RenderTarget, 118
  - ~RenderTarget, 119
  - ApplyMipResolution, 119
  - ApplyResolution, 119
  - AttachDepthBuffer, 119
  - AttachDepthStencilBuffer, 119
  - AttachStencilBuffer, 120
  - AttachTexture1DArray, 120
  - AttachTexture1D, 120
  - AttachTexture2DArray, 120
  - AttachTexture2D, 120
  - AttachTexture3D, 120
  - AttachTextureCube, 120
  - AttachTextureCubeArray, 120
  - DetachTextures, 120
  - GetResolution, 120
  - ResetResolution, 121
- LLGL::RenderingCaps, 97
  - clippingRange, 98
  - has3DTextures, 98
  - hasComputeShaders, 98
  - hasConservativeRasterization, 98
  - hasConstantBuffers, 98
  - hasCubeTextureArrays, 99
  - hasCubeTextures, 99
  - hasGLSL, 99
  - hasGeometryShaders, 99
  - hasHLSL, 99
  - hasInstancing, 99
  - hasOffsetInstancing, 99
  - hasRenderTargets, 99
  - hasSamplers, 99
  - hasStorageBuffers, 100
  - hasTessellationShaders, 100
  - hasTextureArrays, 100
  - hasUniforms, 100
  - hasViewportArrays, 100
  - max1DTextureSize, 100
  - max2DTextureSize, 100
  - max3DTextureSize, 100
  - maxAnisotropy, 100
  - maxComputeShaderWorkGroupSize, 100
  - maxConstantBufferSize, 101
  - maxCubeTextureSize, 101
  - maxNumComputeShaderWorkGroups, 101
  - maxNumRenderTargetAttachments, 101
  - maxNumTextureArrayLayers, 101
  - maxPatchVertices, 101
  - screenOrigin, 101
- LLGL::RenderingDebugger, 102
  - ~RenderingDebugger, 102
  - OnError, 102
  - OnWarning, 102
  - PostError, 102
  - PostWarning, 103
  - RenderingDebugger, 102
- LLGL::RenderingDebugger::Message, 74
  - Block, 75
  - BlockAfter, 75
  - GetOccurrences, 75
  - GetSource, 75
  - GetText, 75
  - IncOccurrence, 75
  - IsBlocked, 75
  - Message, 75
  - operator=, 75
  - RenderingDebugger, 75
- LLGL::RenderingProfiler, 103
  - dispatchComputeCalls, 105
  - drawCalls, 105
  - mapConstantBuffer, 105
  - mapStorageBuffer, 105
  - RecordDrawCall, 104
  - renderedLines, 105
  - renderedPatches, 105
  - renderedPoints, 105
  - renderedTriangles, 105
  - ResetCounters, 104
  - setComputePipeline, 105
  - setConstantBuffer, 105
  - setGraphicsPipeline, 105
  - setIndexBuffer, 105
  - setRenderTarget, 105
  - setSampler, 105
  - setStorageBuffer, 105
  - setTexture, 105
  - setVertexBuffer, 105
  - writeConstantBuffer, 105
  - writeIndexBuffer, 105
  - writeStorageBuffer, 105
  - writeVertexBuffer, 105
- LLGL::RenderingProfiler::Counter, 61
  - Count, 61
  - Inc, 61
  - operator unsigned int, 61

- Reset, 61
- ValueType, 61
- LLGL::Sampler, 121
  - ~Sampler, 121
- LLGL::SamplerDescriptor, 121
  - borderColor, 122
  - compareOp, 122
  - depthCompare, 122
  - magFilter, 123
  - maxAnisotropy, 123
  - maxLOD, 123
  - minFilter, 123
  - minLOD, 123
  - mipMapFilter, 123
  - mipMapLODBias, 123
  - mipMapping, 123
  - textureWrapU, 123
  - textureWrapV, 123
  - textureWrapW, 124
- LLGL::Scissor, 124
  - height, 125
  - Scissor, 125
  - width, 125
  - x, 125
  - y, 125
- LLGL::Shader, 125
  - ~Shader, 126
  - Compile, 126
  - Disassemble, 126
  - GetType, 126
  - QueryInfoLog, 127
  - Shader, 126
- LLGL::ShaderCompileFlags, 127
  - Debug, 127
  - O1, 127
  - O2, 127
  - O3, 127
  - WarnError, 127
- LLGL::ShaderDisassembleFlags, 128
  - InstructionOnly, 128
- LLGL::ShaderProgram, 128
  - ~ShaderProgram, 129
  - AttachShader, 129
  - BindConstantBuffer, 130
  - BindStorageBuffer, 130
  - BindVertexAttributes, 131
  - LinkShaders, 131
  - LockShaderUniform, 131
  - QueryConstantBuffers, 132
  - QueryInfoLog, 132
  - QueryStorageBuffers, 132
  - QueryUniforms, 132
  - QueryVertexAttributes, 133
  - UnlockShaderUniform, 133
- LLGL::ShaderSource, 133
  - ~ShaderSource, 134
  - ShaderSource, 134
  - sourceGLSL, 134
  - sourceHLSL, 135
- LLGL::ShaderSource::GLSL, 64
  - sourceCode, 64
- LLGL::ShaderSource::HLSL, 67
  - entryPoint, 68
  - flags, 68
  - sourceCode, 68
  - target, 68
- LLGL::ShaderStageFlags, 135
  - AllGraphicsStages, 135
  - AllStages, 135
  - AllTessStages, 135
  - ComputeStage, 135
  - FragmentStage, 135
  - GeometryStage, 135
  - TessControlStage, 135
  - TessEvaluationStage, 135
  - VertexStage, 135
- LLGL::ShaderUniform, 136
  - ~ShaderUniform, 137
  - SetUniform, 137, 138
  - SetUniformArray, 138, 139
- LLGL::StencilDescriptor, 140
  - back, 141
  - front, 141
  - testEnabled, 141
- LLGL::StencilFaceDescriptor, 141
  - compareMask, 142
  - compareOp, 142
  - depthFailOp, 142
  - depthPassOp, 142
  - reference, 142
  - stencilFailOp, 142
  - writeMask, 142
- LLGL::StorageBuffer, 142
  - ~StorageBuffer, 143
- LLGL::StorageBufferDescriptor, 143
  - size, 144
  - StorageBufferDescriptor, 144
  - type, 144
  - usage, 144
- LLGL::StorageBufferViewDescriptor, 144
  - index, 145
  - name, 145
  - type, 145
- LLGL::SubTextureDescriptor, 145
  - ~SubTextureDescriptor, 146
  - mipLevel, 146
  - SubTextureDescriptor, 146
  - texture1DDesc, 146
  - texture2DDesc, 147
  - texture3DDesc, 147
  - textureCubeDesc, 147
- LLGL::SubTextureDescriptor::Texture1DDescriptor, 148
  - layerOffset, 149
  - layers, 149
  - width, 149
  - x, 149

- LLGL::SubTextureDescriptor::Texture2DDescriptor, [151](#)
  - height, [151](#)
  - layerOffset, [151](#)
  - layers, [151](#)
  - width, [151](#)
  - x, [151](#)
  - y, [151](#)
- LLGL::SubTextureDescriptor::Texture3DDescriptor, [152](#)
  - depth, [153](#)
  - height, [153](#)
  - width, [153](#)
  - x, [153](#)
  - y, [153](#)
  - z, [153](#)
- LLGL::SubTextureDescriptor::TextureCubeDescriptor, [154](#)
  - cubeFaceOffset, [154](#)
  - cubeFaces, [154](#)
  - height, [154](#)
  - layerOffset, [154](#)
  - width, [154](#)
  - x, [154](#)
  - y, [155](#)
- LLGL::Texture, [147](#)
  - ~Texture, [148](#)
  - GetType, [148](#)
  - QueryMipLevelSize, [148](#)
  - SetType, [148](#)
- LLGL::TextureDescriptor, [156](#)
  - ~TextureDescriptor, [157](#)
  - format, [157](#)
  - texture1DDesc, [157](#)
  - texture2DDesc, [157](#)
  - texture3DDesc, [157](#)
  - textureCubeDesc, [157](#)
  - TextureDescriptor, [157](#)
  - type, [157](#)
- LLGL::TextureDescriptor::Texture1DDescriptor, [149](#)
  - layers, [150](#)
  - width, [150](#)
- LLGL::TextureDescriptor::Texture2DDescriptor, [150](#)
  - height, [150](#)
  - layers, [150](#)
  - width, [150](#)
- LLGL::TextureDescriptor::Texture3DDescriptor, [152](#)
  - depth, [152](#)
  - height, [152](#)
  - width, [152](#)
- LLGL::TextureDescriptor::TextureCubeDescriptor, [155](#)
  - height, [155](#)
  - layers, [155](#)
  - width, [155](#)
- LLGL::Timer, [157](#)
  - ~Timer, [158](#)
  - Create, [158](#)
  - FrameCount, [158](#)
  - GetDeltaTime, [158](#)
  - GetFrameCount, [159](#)
  - GetFrequency, [159](#)
  - MeasureTime, [159](#)
  - ResetFrameCounter, [159](#)
  - Start, [159](#)
  - Stop, [160](#)
- LLGL::UniformDescriptor, [160](#)
  - location, [160](#)
  - name, [160](#)
  - size, [160](#)
  - type, [160](#)
- LLGL::VertexAttribute, [161](#)
  - components, [161](#)
  - conversion, [161](#)
  - dataType, [161](#)
  - name, [161](#)
  - offset, [161](#)
  - perInstance, [162](#)
  - semanticIndex, [162](#)
- LLGL::VertexBuffer, [162](#)
  - ~VertexBuffer, [162](#)
  - GetVertexFormat, [162](#)
  - SetVertexFormat, [162](#)
- LLGL::VertexBufferDescriptor, [163](#)
  - size, [163](#)
  - usage, [163](#)
  - VertexBufferDescriptor, [163](#)
  - vertexFormat, [163](#)
- LLGL::VertexFormat, [164](#)
  - AddAttribute, [164](#), [165](#)
  - GetAttributes, [165](#)
  - GetFormatSize, [166](#)
- LLGL::VideoAdapterDescriptor, [166](#)
  - name, [166](#)
  - outputs, [166](#)
  - vendor, [166](#)
  - videoMemory, [167](#)
- LLGL::VideoDisplayMode, [167](#)
  - height, [167](#)
  - refreshRate, [167](#)
  - width, [167](#)
- LLGL::VideoModeDescriptor, [167](#)
  - colorDepth, [168](#)
  - fullscreen, [168](#)
  - resolution, [168](#)
  - swapChainMode, [168](#)
- LLGL::VideoOutput, [168](#)
  - displayModes, [169](#)
- LLGL::Viewport, [169](#)
  - height, [170](#)
  - maxDepth, [170](#)
  - minDepth, [170](#)
  - Viewport, [170](#)
  - width, [170](#)
  - x, [170](#)
  - y, [170](#)
- LLGL::VsyncDescriptor, [170](#)
  - enabled, [171](#)
  - interval, [171](#)

- refreshRate, 171
- LLGL::Window, 171
  - ~Window, 173
  - AddEventListener, 173
  - Create, 173
  - GetNativeHandle, 173
  - GetPosition, 173
  - GetSize, 173
  - GetTitle, 173
  - IsShown, 173
  - PostChar, 173
  - PostDoubleClick, 173
  - PostGlobalMotion, 173
  - PostKeyDown, 173
  - PostKeyUp, 173
  - PostLocalMotion, 173
  - PostQuit, 173
  - PostResize, 173
  - PostWheelMotion, 174
  - ProcessEvents, 174
  - ProcessSystemEvents, 174
  - QueryDesc, 174
  - Recreate, 174
  - RemoveEventListener, 174
  - SetDesc, 174
  - SetPosition, 174
  - SetSize, 174
  - SetTitle, 174
  - Show, 174
- LLGL::Window::EventListener, 62
  - ~EventListener, 63
  - OnChar, 63
  - OnDoubleClick, 63
  - OnGlobalMotion, 63
  - OnKeyDown, 63
  - OnKeyUp, 63
  - OnLocalMotion, 63
  - OnProcessEvents, 63
  - OnQuit, 63
  - OnResize, 63
  - OnWheelMotion, 64
  - Window, 64
- LLGL::WindowDescriptor, 175
  - acceptDropFiles, 175
  - borderless, 175
  - centered, 175
  - position, 175
  - preventForPowerSafe, 175
  - resizable, 175
  - size, 175
  - title, 175
  - visible, 176
  - windowContext, 176
- LLGL\_EXPORT
  - Export.h, 180
- LLGL, 13
  - A, 27
  - Add, 23
  - AnySamplesPassed, 32
  - AnySamplesPassedConservative, 32
  - AppendStructuredBuffer, 35
  - Apps, 28
  - Attn, 29
  - AxisDirection, 23
  - B, 27
  - BGRA, 26
  - BGR, 26
  - Back, 25, 26
  - BlendArithmetic, 23
  - BlendOp, 23
  - Border, 37
  - BrowserBack, 29
  - BrowserFavorites, 29
  - BrowserForward, 29
  - BrowserHome, 29
  - BrowserRefresh, 29
  - BrowserSearch, 29
  - BrowserStop, 29
  - Buffer, 35
  - BufferCPUAccess, 23
  - BufferUsage, 24
  - ByRegionNoWait, 33
  - ByRegionNoWaitInverted, 33
  - ByRegionWait, 33
  - ByRegionWaitInverted, 33
  - ByteAddressBuffer, 35
  - ByteBuffer, 21
  - C, 27
  - Cancel, 26
  - Capital, 27
  - Clamp, 37
  - Clear, 26
  - ClippingInputPrimitives, 32
  - ClippingOutputPrimitives, 32
  - ClippingRange, 24
  - ColorRGBAb, 22
  - ColorRGBAd, 22
  - ColorRGBAf, 22
  - ColorRGBAT, 22
  - ColorRGBAub, 22
  - ColorRGBA, 22
  - ColorRGBb, 22
  - ColorRGBd, 22
  - ColorRGBf, 22
  - ColorRGBT, 22
  - ColorRGBub, 22
  - ColorRGB, 21
  - Comma, 29
  - CompareOp, 24
  - CompareSWO, 38
  - CompressedRGBA, 26
  - CompressedRGB, 26
  - Compute, 34
  - ComputeShaderInvocations, 32
  - ConsumeStructuredBuffer, 35
  - Control, 26

ConvertImageBuffer, 38  
CrSel, 29  
CullMode, 25  
D, 27  
D0, 27  
D1, 27  
D2, 27  
D3, 27  
D4, 27  
D5, 27  
D6, 27  
D7, 27  
D8, 27  
D9, 27  
DataType, 25  
DataTypeSize, 39  
DebugCallback, 22  
DecClamp, 34  
DecWrap, 34  
Delete, 27  
Depth, 26  
DepthComponent, 36  
DepthStencil, 26, 36  
DestAlpha, 23  
DestColor, 23  
Disabled, 25  
Double, 25, 38  
Double2, 38  
Double2x2, 38  
Double3, 38  
Double3x3, 38  
Double4, 38  
Double4x4, 38  
DoubleBuffering, 35  
Down, 27  
Dynamic, 24  
E, 27  
End, 27  
Equal, 25  
ErEOF, 29  
ErrorType, 25  
Escape, 27  
Ever, 25  
ExSel, 29  
Exe, 27  
Exponent, 29  
F, 27  
F1, 28  
F10, 28  
F11, 28  
F12, 28  
F13, 28  
F14, 28  
F15, 29  
F16, 29  
F17, 29  
F18, 29  
F19, 29  
F2, 28  
F20, 29  
F21, 29  
F22, 29  
F23, 29  
F24, 29  
F3, 28  
F4, 28  
F5, 28  
F6, 28  
F7, 28  
F8, 28  
F9, 28  
Fill, 30  
Float, 25, 38  
Float2, 38  
Float2x2, 38  
Float3, 38  
Float3x3, 38  
Float4, 38  
Float4x4, 38  
Fragment, 34  
FragmentShaderInvocations, 32  
Front, 25  
G, 27  
GLSL\_110, 34  
GLSL\_120, 34  
GLSL\_130, 34  
GLSL\_140, 34  
GLSL\_150, 34  
GLSL\_330, 34  
GLSL\_400, 34  
GLSL\_410, 34  
GLSL\_420, 34  
GLSL\_430, 34  
GLSL\_440, 34  
GLSL\_450, 34  
Geometry, 34  
GeometryPrimitivesGenerated, 32  
GeometryShaderInvocations, 32  
Greater, 25  
GreaterEqual, 25  
H, 27  
HLSL\_2\_0, 34  
HLSL\_2\_0a, 34  
HLSL\_2\_0b, 34  
HLSL\_3\_0, 34  
HLSL\_4\_0, 34  
HLSL\_4\_1, 34  
HLSL\_5\_0, 34  
Hardware, 33  
Help, 27  
Home, 27  
I, 27  
ImageFormat, 26  
ImageFormatSize, 39  
ImproperArgument, 38  
ImproperState, 38

IncClamp, [34](#)  
IncWrap, [34](#)  
Insert, [27](#)  
Int, [38](#)  
Int16, [25](#)  
Int2, [38](#)  
Int3, [38](#)  
Int32, [25](#)  
Int4, [38](#)  
Int8, [25](#)  
InvDestAlpha, [23](#)  
InvDestColor, [23](#)  
InvSrcAlpha, [23](#)  
InvSrcColor, [23](#)  
InvalidArgument, [26](#)  
InvalidState, [26](#)  
Invert, [34](#)  
IsCompressedFormat, [40](#)  
IsDepthStencilFormat, [40](#)  
J, [27](#)  
K, [27](#)  
Keep, [34](#)  
Key, [26](#)  
Keypad0, [28](#)  
Keypad1, [28](#)  
Keypad2, [28](#)  
Keypad3, [28](#)  
Keypad4, [28](#)  
Keypad5, [28](#)  
Keypad6, [28](#)  
Keypad7, [28](#)  
Keypad8, [28](#)  
Keypad9, [28](#)  
KeypadDecimal, [28](#)  
KeypadDivide, [28](#)  
KeypadMinus, [28](#)  
KeypadMultiply, [28](#)  
KeypadPlus, [28](#)  
KeypadSeparator, [28](#)  
L, [27](#)  
LButton, [26](#)  
LControl, [29](#)  
LMenu, [29](#)  
LShift, [29](#)  
LWin, [28](#)  
LaunchApp1, [29](#)  
LaunchApp2, [29](#)  
LaunchMail, [29](#)  
LaunchMediaSelect, [29](#)  
Left, [27](#)  
Less, [25](#)  
LessEqual, [25](#)  
LineList, [31](#)  
LineListAdjacency, [31](#)  
LineLoop, [31](#)  
LineStrip, [31](#)  
LineStripAdjacency, [31](#)  
Linear, [35](#)  
LowerLeft, [33](#)  
M, [27](#)  
MButton, [26](#)  
Max, [23](#)  
MaxColorValue, [40](#)  
MaxColorValue< bool >, [40](#)  
MaxColorValue< unsigned char >, [40](#)  
MediaNextTrack, [29](#)  
MediaPlayPause, [29](#)  
MediaPrevTrack, [29](#)  
MediaStop, [29](#)  
Menu, [27](#)  
Min, [23](#)  
Minus, [29](#)  
MinusOneToOne, [24](#)  
Mirror, [37](#)  
MirrorOnce, [37](#)  
N, [27](#)  
Nearest, [35](#)  
Never, [25](#)  
NoName, [30](#)  
NoWait, [33](#)  
NoWaitInverted, [33](#)  
NotEqual, [25](#)  
NumLock, [29](#)  
NumMipLevels, [40](#)  
O, [27](#)  
OEMClear, [30](#)  
One, [23](#)  
OpenGL\_1\_0, [30](#)  
OpenGL\_1\_1, [30](#)  
OpenGL\_1\_2, [30](#)  
OpenGL\_1\_3, [30](#)  
OpenGL\_1\_4, [30](#)  
OpenGL\_1\_5, [30](#)  
OpenGL\_2\_0, [30](#)  
OpenGL\_2\_1, [30](#)  
OpenGL\_3\_0, [30](#)  
OpenGL\_3\_1, [30](#)  
OpenGL\_3\_2, [30](#)  
OpenGL\_3\_3, [30](#)  
OpenGL\_4\_0, [30](#)  
OpenGL\_4\_1, [30](#)  
OpenGL\_4\_2, [30](#)  
OpenGL\_4\_3, [30](#)  
OpenGL\_4\_4, [30](#)  
OpenGL\_4\_5, [30](#)  
OpenGL\_Latest, [30](#)  
OpenGLVersion, [30](#)  
operator!=, [41](#)  
operator\*, [41](#)  
operator+, [41](#)  
operator-, [41](#)  
operator/, [41](#)  
operator==, [41](#)  
P, [28](#)  
PA1, [30](#)  
PageDown, [27](#)



PageUp, [27](#)  
Patches1, [31](#)  
Patches10, [31](#)  
Patches11, [31](#)  
Patches12, [31](#)  
Patches13, [31](#)  
Patches14, [31](#)  
Patches15, [31](#)  
Patches16, [31](#)  
Patches17, [31](#)  
Patches18, [31](#)  
Patches19, [31](#)  
Patches2, [31](#)  
Patches20, [31](#)  
Patches21, [31](#)  
Patches22, [31](#)  
Patches23, [31](#)  
Patches24, [31](#)  
Patches25, [31](#)  
Patches26, [32](#)  
Patches27, [32](#)  
Patches28, [32](#)  
Patches29, [32](#)  
Patches3, [31](#)  
Patches30, [32](#)  
Patches31, [32](#)  
Patches32, [32](#)  
Patches4, [31](#)  
Patches5, [31](#)  
Patches6, [31](#)  
Patches7, [31](#)  
Patches8, [31](#)  
Patches9, [31](#)  
Pause, [27](#)  
Period, [29](#)  
Play, [30](#)  
Plus, [29](#)  
Point, [22](#)  
PointList, [31](#)  
PointlessOperation, [38](#)  
Points, [30](#)  
PolygonMode, [30](#)  
PrimitiveTopology, [30](#)  
PrimitivesGenerated, [32](#)  
PrimitivesSubmitted, [32](#)  
Print, [27](#)  
Q, [28](#)  
QueryType, [32](#)  
R, [26](#), [28](#), [36](#)  
R16, [36](#)  
R16Float, [36](#)  
R16Sgn, [36](#)  
R32Float, [36](#)  
R32SInt, [36](#)  
R32UInt, [36](#)  
R8, [36](#)  
R8Sgn, [36](#)  
RButton, [26](#)  
RControl, [29](#)  
RG16, [36](#)  
RG16Float, [36](#)  
RG16Sgn, [36](#)  
RG32Float, [36](#)  
RG32SInt, [36](#)  
RG32UInt, [36](#)  
RG8, [36](#)  
RG8Sgn, [36](#)  
RGB16, [36](#)  
RGB16Float, [37](#)  
RGB16Sgn, [36](#)  
RGB32Float, [37](#)  
RGB32SInt, [37](#)  
RGB32UInt, [37](#)  
RGB8, [36](#)  
RGB8Sgn, [36](#)  
RGB\_DXT1, [37](#)  
RGBA16, [37](#)  
RGBA16Float, [37](#)  
RGBA16Sgn, [37](#)  
RGBA32Float, [37](#)  
RGBA32SInt, [37](#)  
RGBA32UInt, [37](#)  
RGBA8, [37](#)  
RGBA8Sgn, [37](#)  
RGBA\_DXT1, [37](#)  
RGBA\_DXT3, [37](#)  
RGBA\_DXT5, [37](#)  
RGBA, [26](#), [36](#)  
RGB, [26](#), [36](#)  
RMenu, [29](#)  
RShift, [29](#)  
RWBuffer, [35](#)  
RWByteAddressBuffer, [35](#)  
RWStructuredBuffer, [35](#)  
RWin, [28](#)  
ReadOnly, [24](#)  
ReadWrite, [24](#)  
RenderConditionMode, [32](#)  
RendererInfo, [33](#)  
Repeat, [37](#)  
Replace, [34](#)  
Return, [26](#)  
RevSubtract, [23](#)  
RG, [26](#), [36](#)  
Right, [27](#)  
S, [28](#)  
Sampler1D, [38](#)  
Sampler2D, [38](#)  
Sampler3D, [38](#)  
SamplerCube, [38](#)  
SamplesPassed, [32](#)  
ScreenOrigin, [33](#)  
ScrollLock, [29](#)  
Select, [27](#)  
ShaderType, [33](#)  
ShadingLanguage, [34](#)

- ShadingLanguageVersion, 33
- Shift, 26
- SingleBuffering, 35
- Size, 22
- Sleep, 28
- Snapshot, 27
- Space, 27
- SrcAlpha, 23
- SrcColor, 23
- Static, 24
- StencilOp, 34
- StorageBufferType, 34
- StreamOutOverflow, 32
- StreamOutPrimitivesWritten, 32
- StructuredBuffer, 35
- Subtract, 23
- SwapChainMode, 35
- T, 28
- Tab, 26
- TessControl, 34
- TessControlShaderInvocations, 32
- TessEvaluation, 34
- TessEvaluationShaderInvocations, 32
- Texture1DArray, 37
- Texture1D, 37
- Texture2DArray, 37
- Texture2D, 37
- Texture3D, 37
- TextureCube, 37
- TextureCubeArray, 37
- TextureFilter, 35
- TextureFormat, 35
- TextureType, 37
- TextureWrap, 37
- TimeElapsed, 32
- TriangleFan, 31
- TriangleList, 31
- TriangleListAdjacency, 31
- TriangleStrip, 31
- TriangleStripAdjacency, 31
- TripleBuffering, 35
- U, 28
- UInt16, 25
- UInt32, 25
- UInt8, 25
- Undefined, 37
- UniformType, 37
- Unknown, 36
- UnsupportedFeature, 26
- Up, 27
- UpperLeft, 33
- V, 28
- Vendor, 33
- Version, 33
- Vertex, 34
- VertexShaderInvocations, 32
- VerticesSubmitted, 32
- VolumeDown, 29
- VolumeMute, 29
- VolumeUp, 29
- W, 28
- Wait, 33
- WaitInverted, 33
- WarningType, 38
- Wireframe, 30
- WriteOnly, 24
- X, 28
- XButton1, 26
- XButton2, 26
- XNeg, 23
- XPos, 23
- Y, 28
- YNeg, 23
- YPos, 23
- Z, 28
- ZNeg, 23
- ZPos, 23
- Zero, 23, 34
- ZeroToOne, 24
- Zoom, 30
- LMenu
  - LLGL, 29
- LShift
  - LLGL, 29
- LWin
  - LLGL, 28
- LaunchApp1
  - LLGL, 29
- LaunchApp2
  - LLGL, 29
- LaunchMail
  - LLGL, 29
- LaunchMediaSelect
  - LLGL, 29
- layerOffset
  - LLGL::SubTextureDescriptor::Texture1DDescriptor, 149
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, 151
  - LLGL::SubTextureDescriptor::TextureCubeDescriptor, 154
- layers
  - LLGL::SubTextureDescriptor::Texture1DDescriptor, 149
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, 151
  - LLGL::TextureDescriptor::Texture1DDescriptor, 150
  - LLGL::TextureDescriptor::Texture2DDescriptor, 150
  - LLGL::TextureDescriptor::TextureCubeDescriptor, 155
- Left
  - LLGL, 27
- Less
  - LLGL, 25

- LessEqual
  - LLGL, [25](#)
- LineList
  - LLGL, [31](#)
- LineListAdjacency
  - LLGL, [31](#)
- LineLoop
  - LLGL, [31](#)
- LineStrip
  - LLGL, [31](#)
- LineStripAdjacency
  - LLGL, [31](#)
- Linear
  - LLGL, [35](#)
- LinkShaders
  - LLGL::ShaderProgram, [131](#)
- LinuxNativeHandle.h, [186](#)
- Load
  - LLGL::RenderSystem, [113](#)
- location
  - LLGL::UniformDescriptor, [160](#)
- LockShaderUniform
  - LLGL::ShaderProgram, [131](#)
- Log.h, [186](#)
- LowerLeft
  - LLGL, [33](#)
- M
  - LLGL, [27](#)
- MButton
  - LLGL, [26](#)
- MacOSNativeHandle.h, [187](#)
- magFilter
  - LLGL::SamplerDescriptor, [123](#)
- MakeCurrent
  - LLGL::RenderSystem, [113](#)
- mapConstantBuffer
  - LLGL::RenderingProfiler, [105](#)
- MapStorageBuffer
  - LLGL::RenderContext, [88](#)
- mapStorageBuffer
  - LLGL::RenderingProfiler, [105](#)
- Max
  - LLGL, [23](#)
- max1DTextureSize
  - LLGL::RenderingCaps, [100](#)
- max2DTextureSize
  - LLGL::RenderingCaps, [100](#)
- max3DTextureSize
  - LLGL::RenderingCaps, [100](#)
- maxAnisotropy
  - LLGL::RenderingCaps, [100](#)
  - LLGL::SamplerDescriptor, [123](#)
- MaxColorValue
  - LLGL, [40](#)
- MaxColorValue< bool >
  - LLGL, [40](#)
- MaxColorValue< unsigned char >
  - LLGL, [40](#)
- maxComputeShaderWorkGroupSize
  - LLGL::RenderingCaps, [100](#)
- maxConstantBufferSize
  - LLGL::RenderingCaps, [101](#)
- maxCubeTextureSize
  - LLGL::RenderingCaps, [101](#)
- maxDepth
  - LLGL::Viewport, [170](#)
- maxLOD
  - LLGL::SamplerDescriptor, [123](#)
- maxNumComputeShaderWorkGroups
  - LLGL::RenderingCaps, [101](#)
- maxNumRenderTargetAttachments
  - LLGL::RenderingCaps, [101](#)
- maxNumTextureArrayLayers
  - LLGL::RenderingCaps, [101](#)
- maxPatchVertices
  - LLGL::RenderingCaps, [101](#)
- MeasureTime
  - LLGL::Timer, [159](#)
- MediaNextTrack
  - LLGL, [29](#)
- MediaPlayPause
  - LLGL, [29](#)
- MediaPrevTrack
  - LLGL, [29](#)
- MediaStop
  - LLGL, [29](#)
- Menu
  - LLGL, [27](#)
- Message
  - LLGL::RenderingDebugger::Message, [75](#)
- Min
  - LLGL, [23](#)
- minDepth
  - LLGL::Viewport, [170](#)
- minFilter
  - LLGL::SamplerDescriptor, [123](#)
- minLOD
  - LLGL::SamplerDescriptor, [123](#)
- Minus
  - LLGL, [29](#)
- MinusOneToOne
  - LLGL, [24](#)
- mipLevel
  - LLGL::SubTextureDescriptor, [146](#)
- mipMapFilter
  - LLGL::SamplerDescriptor, [123](#)
- mipMapLODBias
  - LLGL::SamplerDescriptor, [123](#)
- mipMapping
  - LLGL::SamplerDescriptor, [123](#)
- Mirror
  - LLGL, [37](#)
- MirrorOnce
  - LLGL, [37](#)
- multiSampleEnabled
  - LLGL::RasterizerDescriptor, [81](#)

## N

LLGL, [27](#)

## name

LLGL::ConstantBufferViewDescriptor, [61](#)

LLGL::StorageBufferViewDescriptor, [145](#)

LLGL::UniformDescriptor, [160](#)

LLGL::VertexAttribute, [161](#)

LLGL::VideoAdapterDescriptor, [166](#)

NativeHandle.h, [187](#)

## Nearest

LLGL, [35](#)

## Never

LLGL, [25](#)

## NoName

LLGL, [30](#)

## NoWait

LLGL, [33](#)

## NoWaitInverted

LLGL, [33](#)

## NotEqual

LLGL, [25](#)

## NumLock

LLGL, [29](#)

## NumMipLevels

LLGL, [40](#)

## O

LLGL, [27](#)

## O1

LLGL::ShaderCompileFlags, [127](#)

## O2

LLGL::ShaderCompileFlags, [127](#)

## O3

LLGL::ShaderCompileFlags, [127](#)

## OEMClear

LLGL, [30](#)

## offset

LLGL::VertexAttribute, [161](#)

## OnChar

LLGL::Window::EventListener, [63](#)

## OnDoubleClick

LLGL::Window::EventListener, [63](#)

## OnError

LLGL::RenderingDebugger, [102](#)

## OnGlobalMotion

LLGL::Window::EventListener, [63](#)

## OnKeyDown

LLGL::Window::EventListener, [63](#)

## OnKeyUp

LLGL::Window::EventListener, [63](#)

## OnLocalMotion

LLGL::Window::EventListener, [63](#)

## OnMakeCurrent

LLGL::RenderSystem, [114](#)

## OnProcessEvents

LLGL::Window::EventListener, [63](#)

## OnQuit

LLGL::Window::EventListener, [63](#)

## OnResize

LLGL::Window::EventListener, [63](#)

## OnWarning

LLGL::RenderingDebugger, [102](#)

## OnWheelMotion

LLGL::Window::EventListener, [64](#)

## One

LLGL, [23](#)

## OpenGL\_1\_0

LLGL, [30](#)

## OpenGL\_1\_1

LLGL, [30](#)

## OpenGL\_1\_2

LLGL, [30](#)

## OpenGL\_1\_3

LLGL, [30](#)

## OpenGL\_1\_4

LLGL, [30](#)

## OpenGL\_1\_5

LLGL, [30](#)

## OpenGL\_2\_0

LLGL, [30](#)

## OpenGL\_2\_1

LLGL, [30](#)

## OpenGL\_3\_0

LLGL, [30](#)

## OpenGL\_3\_1

LLGL, [30](#)

## OpenGL\_3\_2

LLGL, [30](#)

## OpenGL\_3\_3

LLGL, [30](#)

## OpenGL\_4\_0

LLGL, [30](#)

## OpenGL\_4\_1

LLGL, [30](#)

## OpenGL\_4\_2

LLGL, [30](#)

## OpenGL\_4\_3

LLGL, [30](#)

## OpenGL\_4\_4

LLGL, [30](#)

## OpenGL\_4\_5

LLGL, [30](#)

## OpenGL\_Latest

LLGL, [30](#)

## OpenGLVersion

LLGL, [30](#)

## operator unsigned int

LLGL::RenderingProfiler::Counter, [61](#)

## operator!=

LLGL, [41](#)

## operator\*

LLGL, [41](#)

## operator\*=

LLGL::Color, [50](#)

LLGL::Color< T, 3u >, [53](#)

LLGL::Color< T, 4u >, [56](#)

## operator+

- LLGL, [41](#)
- operator+=
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- operator-
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- LLGL, [41](#)
- operator-=
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- operator/
  - LLGL, [41](#)
- operator/=
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- operator=
  - LLGL::Query, [79](#)
  - LLGL::RenderContext, [89](#)
  - LLGL::RenderSystem, [114](#)
  - LLGL::RenderingDebugger::Message, [75](#)
- operator==
  - LLGL, [41](#)
- operator[]
  - LLGL::Color, [50](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- outputs
  - LLGL::VideoAdapterDescriptor, [166](#)
- P
  - LLGL, [28](#)
- PA1
  - LLGL, [30](#)
- PageDown
  - LLGL, [27](#)
- PageUp
  - LLGL, [27](#)
- parentWindow
  - LLGL::NativeContextHandle, [76](#)
- Patches1
  - LLGL, [31](#)
- Patches10
  - LLGL, [31](#)
- Patches11
  - LLGL, [31](#)
- Patches12
  - LLGL, [31](#)
- Patches13
  - LLGL, [31](#)
- Patches14
  - LLGL, [31](#)
- Patches15
  - LLGL, [31](#)
- Patches16
  - LLGL, [31](#)
- Patches17
  - LLGL, [31](#)
- Patches18
  - LLGL, [31](#)
- Patches19
  - LLGL, [31](#)
- Patches2
  - LLGL, [31](#)
- Patches20
  - LLGL, [31](#)
- Patches21
  - LLGL, [31](#)
- Patches22
  - LLGL, [31](#)
- Patches23
  - LLGL, [31](#)
- Patches24
  - LLGL, [31](#)
- Patches25
  - LLGL, [31](#)
- Patches26
  - LLGL, [32](#)
- Patches27
  - LLGL, [32](#)
- Patches28
  - LLGL, [32](#)
- Patches29
  - LLGL, [32](#)
- Patches3
  - LLGL, [31](#)
- Patches30
  - LLGL, [32](#)
- Patches31
  - LLGL, [32](#)
- Patches32
  - LLGL, [32](#)
- Patches4
  - LLGL, [31](#)
- Patches5
  - LLGL, [31](#)
- Patches6
  - LLGL, [31](#)
- Patches7
  - LLGL, [31](#)
- Patches8
  - LLGL, [31](#)
- Patches9
  - LLGL, [31](#)
- Pause
  - LLGL, [27](#)
- perInstance
  - LLGL::VertexAttribute, [162](#)
- Period
  - LLGL, [29](#)
- Play
  - LLGL, [30](#)
- Plus

- LLGL, [29](#)
- Point
  - LLGL, [22](#)
- PointList
  - LLGL, [31](#)
- PointlessOperation
  - LLGL, [38](#)
- Points
  - LLGL, [30](#)
- PolygonMode
  - LLGL, [30](#)
- polygonMode
  - LLGL::RasterizerDescriptor, [81](#)
- position
  - LLGL::WindowDescriptor, [175](#)
- PostChar
  - LLGL::Window, [173](#)
- PostDoubleClick
  - LLGL::Window, [173](#)
- PostError
  - LLGL::RenderingDebugger, [102](#)
- PostGlobalMotion
  - LLGL::Window, [173](#)
- PostKeyDown
  - LLGL::Window, [173](#)
- PostKeyUp
  - LLGL::Window, [173](#)
- PostLocalMotion
  - LLGL::Window, [173](#)
- PostQuit
  - LLGL::Window, [173](#)
- PostResize
  - LLGL::Window, [173](#)
- PostWarning
  - LLGL::RenderingDebugger, [103](#)
- PostWheelMotion
  - LLGL::Window, [174](#)
- Present
  - LLGL::RenderContext, [89](#)
- preventForPowerSafe
  - LLGL::WindowDescriptor, [175](#)
- PrimitiveTopology
  - LLGL, [30](#)
- primitiveTopology
  - LLGL::GraphicsPipelineDescriptor, [67](#)
- PrimitivesGenerated
  - LLGL, [32](#)
- PrimitivesSubmitted
  - LLGL, [32](#)
- Print
  - LLGL, [27](#)
- ProcessEvents
  - LLGL::Window, [174](#)
- ProcessSystemEvents
  - LLGL::Window, [174](#)
- profileOpenGL
  - LLGL::RenderContextDescriptor, [96](#)
- Ptr
  - LLGL::Color, [51](#)
  - LLGL::Color< T, 3u >, [53](#)
  - LLGL::Color< T, 4u >, [56](#)
- Q
  - LLGL, [28](#)
- Query
  - LLGL::Query, [79](#)
- Query.h, [187](#)
- QueryConstantBuffers
  - LLGL::ShaderProgram, [132](#)
- QueryDesc
  - LLGL::Window, [174](#)
- QueryInfoLog
  - LLGL::Shader, [127](#)
  - LLGL::ShaderProgram, [132](#)
- QueryMipLevelSize
  - LLGL::Texture, [148](#)
- QueryRendererInfo
  - LLGL::RenderSystem, [114](#)
- QueryRenderingCaps
  - LLGL::RenderSystem, [114](#)
- QueryResult
  - LLGL::RenderContext, [89](#)
- QueryShadingLanguage
  - LLGL::RenderSystem, [114](#)
- QueryStorageBuffers
  - LLGL::ShaderProgram, [132](#)
- QueryTextureDescriptor
  - LLGL::RenderSystem, [114](#)
- QueryType
  - LLGL, [32](#)
- QueryUniforms
  - LLGL::ShaderProgram, [132](#)
- QueryVertexAttributes
  - LLGL::ShaderProgram, [133](#)
- R
  - LLGL, [26, 28, 36](#)
- r
  - LLGL::Color< T, 3u >, [54](#)
  - LLGL::Color< T, 4u >, [57](#)
- R16
  - LLGL, [36](#)
- R16Float
  - LLGL, [36](#)
- R16Sgn
  - LLGL, [36](#)
- R32Float
  - LLGL, [36](#)
- R32SInt
  - LLGL, [36](#)
- R32UInt
  - LLGL, [36](#)
- R8
  - LLGL, [36](#)
- R8Sgn
  - LLGL, [36](#)
- RButton

- LLGL, [26](#)
- RControl
  - LLGL, [29](#)
- RG16
  - LLGL, [36](#)
- RG16Float
  - LLGL, [36](#)
- RG16Sgn
  - LLGL, [36](#)
- RG32Float
  - LLGL, [36](#)
- RG32SInt
  - LLGL, [36](#)
- RG32UInt
  - LLGL, [36](#)
- RG8
  - LLGL, [36](#)
- RG8Sgn
  - LLGL, [36](#)
- RGB16
  - LLGL, [36](#)
- RGB16Float
  - LLGL, [37](#)
- RGB16Sgn
  - LLGL, [36](#)
- RGB32Float
  - LLGL, [37](#)
- RGB32SInt
  - LLGL, [37](#)
- RGB32UInt
  - LLGL, [37](#)
- RGB8
  - LLGL, [36](#)
- RGB8Sgn
  - LLGL, [36](#)
- RGB\_DXT1
  - LLGL, [37](#)
- RGBA16
  - LLGL, [37](#)
- RGBA16Float
  - LLGL, [37](#)
- RGBA16Sgn
  - LLGL, [37](#)
- RGBA32Float
  - LLGL, [37](#)
- RGBA32SInt
  - LLGL, [37](#)
- RGBA32UInt
  - LLGL, [37](#)
- RGBA8
  - LLGL, [37](#)
- RGBA8Sgn
  - LLGL, [37](#)
- RGBA\_DXT1
  - LLGL, [37](#)
- RGBA\_DXT3
  - LLGL, [37](#)
- RGBA\_DXT5
  - LLGL, [37](#)
- RGBA
  - LLGL, [37](#)
- RGBA
  - LLGL, [26](#), [36](#)
- RGB
  - LLGL, [26](#), [36](#)
- RMenu
  - LLGL, [29](#)
- RShift
  - LLGL, [29](#)
- RWBuffer
  - LLGL, [35](#)
- RWByteAddressBuffer
  - LLGL, [35](#)
- RWStructuredBuffer
  - LLGL, [35](#)
- RWin
  - LLGL, [28](#)
- rasterizer
  - LLGL::GraphicsPipelineDescriptor, [67](#)
- ReadOnly
  - LLGL, [24](#)
- ReadTexture
  - LLGL::RenderSystem, [114](#)
- ReadWrite
  - LLGL, [24](#)
- RecordDrawCall
  - LLGL::RenderingProfiler, [104](#)
- Recreate
  - LLGL::Window, [174](#)
- reference
  - LLGL::StencilFaceDescriptor, [142](#)
- refreshRate
  - LLGL::VideoDisplayMode, [167](#)
  - LLGL::VsyncDescriptor, [171](#)
- Release
  - LLGL::RenderSystem, [115](#), [116](#)
- RemoveEventListener
  - LLGL::Window, [174](#)
- renderCondition
  - LLGL::QueryDescriptor, [80](#)
- RenderConditionMode
  - LLGL, [32](#)
- RenderContext
  - LLGL::RenderContext, [84](#)
- RenderContext.h, [188](#)
- RenderContextDescriptor.h, [188](#)
- RenderContextFlags.h, [189](#)
- RenderSystem
  - LLGL::RenderSystem, [108](#)
- RenderSystem.h, [191](#)
- RenderSystemFlags.h, [191](#)
- RenderTarget.h, [192](#)
- renderedLines
  - LLGL::RenderingProfiler, [105](#)
- renderedPatches
  - LLGL::RenderingProfiler, [105](#)
- renderedPoints
  - LLGL::RenderingProfiler, [105](#)

- renderedTriangles
  - LLGL::RenderingProfiler, 105
- RendererInfo
  - LLGL, 33
- RenderingDebugger
  - LLGL::RenderingDebugger, 102
  - LLGL::RenderingDebugger::Message, 75
- RenderingDebugger.h, 190
- RenderingProfiler.h, 190
- Repeat
  - LLGL, 37
- Replace
  - LLGL, 34
- Reset
  - LLGL::RenderingProfiler::Counter, 61
- ResetCounters
  - LLGL::RenderingProfiler, 104
- ResetFrameCounter
  - LLGL::Timer, 159
- ResetResolution
  - LLGL::RenderTarget, 121
- ResetVideoMode
  - LLGL::Desktop, 42
- resizable
  - LLGL::WindowDescriptor, 175
- resolution
  - LLGL::VideoModeDescriptor, 168
- Return
  - LLGL, 26
- RevSubtract
  - LLGL, 23
- RG
  - LLGL, 26, 36
- Right
  - LLGL, 27
- S
  - LLGL, 28
- Sampler.h, 193
- Sampler1D
  - LLGL, 38
- Sampler2D
  - LLGL, 38
- Sampler3D
  - LLGL, 38
- SamplerCube
  - LLGL, 38
- SamplerFlags.h, 193
- samples
  - LLGL::AntiAliasingDescriptor, 45
  - LLGL::RasterizerDescriptor, 81
- SamplesPassed
  - LLGL, 32
- Scissor
  - LLGL::Scissor, 125
- scissorTestEnabled
  - LLGL::RasterizerDescriptor, 82
- screen
  - LLGL::NativeContextHandle, 76
- ScreenOrigin
  - LLGL, 33
- screenOrigin
  - LLGL::RenderingCaps, 101
- screenSpaceOriginLowerLeft
  - LLGL::GraphicsAPIDependentStateDescriptor::↔  
StateOpenGLDescriptor, 140
- ScrollLock
  - LLGL, 29
- Select
  - LLGL, 27
- semanticIndex
  - LLGL::VertexAttribute, 162
- SetClearColor
  - LLGL::RenderContext, 89
- SetClearDepth
  - LLGL::RenderContext, 89
- SetClearStencil
  - LLGL::RenderContext, 89
- SetComputePipeline
  - LLGL::RenderContext, 90
- setComputePipeline
  - LLGL::RenderingProfiler, 105
- SetConfiguration
  - LLGL::RenderSystem, 116
- SetConstantBuffer
  - LLGL::RenderContext, 90
- setConstantBuffer
  - LLGL::RenderingProfiler, 105
- SetDesc
  - LLGL::Window, 174
- SetGraphicsAPIDependentState
  - LLGL::RenderContext, 90
- SetGraphicsPipeline
  - LLGL::RenderContext, 91
- setGraphicsPipeline
  - LLGL::RenderingProfiler, 105
- SetIndexBuffer
  - LLGL::RenderContext, 91
- setIndexBuffer
  - LLGL::RenderingProfiler, 105
- SetIndexFormat
  - LLGL::IndexBuffer, 70
- SetPosition
  - LLGL::Window, 174
- SetRenderTarget
  - LLGL::RenderContext, 91
- setRenderTarget
  - LLGL::RenderingProfiler, 105
- SetSampler
  - LLGL::RenderContext, 92
- setSampler
  - LLGL::RenderingProfiler, 105
- SetScissor
  - LLGL::RenderContext, 92
- SetScissorArray
  - LLGL::RenderContext, 92
- SetSize



- LLGL::Window, [174](#)
- SetStdErr
  - LLGL::Log, [42](#)
- SetStdOut
  - LLGL::Log, [42](#)
- SetStorageBuffer
  - LLGL::RenderContext, [93](#)
- setStorageBuffer
  - LLGL::RenderingProfiler, [105](#)
- SetTexture
  - LLGL::RenderContext, [93](#)
- setTexture
  - LLGL::RenderingProfiler, [105](#)
- SetTitle
  - LLGL::Window, [174](#)
- SetType
  - LLGL::Texture, [148](#)
- SetUniform
  - LLGL::ShaderUniform, [137](#), [138](#)
- SetUniformArray
  - LLGL::ShaderUniform, [138](#), [139](#)
- SetVertexBuffer
  - LLGL::RenderContext, [93](#)
- setVertexBuffer
  - LLGL::RenderingProfiler, [105](#)
- SetVertexFormat
  - LLGL::VertexBuffer, [162](#)
- SetVideoMode
  - LLGL::Desktop, [42](#)
  - LLGL::RenderContext, [94](#)
- SetViewport
  - LLGL::RenderContext, [94](#)
- SetViewportArray
  - LLGL::RenderContext, [94](#)
- SetVsync
  - LLGL::RenderContext, [95](#)
- SetWindow
  - LLGL::RenderContext, [95](#)
- Shader
  - LLGL::Shader, [126](#)
- Shader.h, [194](#)
- shaderProgram
  - LLGL::ComputePipelineDescriptor, [58](#)
  - LLGL::GraphicsPipelineDescriptor, [67](#)
- ShaderProgram.h, [194](#)
- ShaderSource
  - LLGL::ShaderSource, [134](#)
- ShaderType
  - LLGL, [33](#)
- ShaderUniform.h, [195](#)
- ShadingLanguage
  - LLGL, [34](#)
- ShadingLanguageVersion
  - LLGL, [33](#)
- ShareWindowAndVideoMode
  - LLGL::RenderContext, [95](#)
- Shift
  - LLGL, [26](#)
- Show
  - LLGL::Window, [174](#)
- SingleBuffering
  - LLGL, [35](#)
- Size
  - LLGL, [22](#)
- size
  - LLGL::ConstantBufferDescriptor, [60](#)
  - LLGL::ConstantBufferViewDescriptor, [61](#)
  - LLGL::IndexBufferDescriptor, [71](#)
  - LLGL::StorageBufferDescriptor, [144](#)
  - LLGL::UniformDescriptor, [160](#)
  - LLGL::VertexBufferDescriptor, [163](#)
  - LLGL::WindowDescriptor, [175](#)
- Sleep
  - LLGL, [28](#)
- slopeScaledDepthBias
  - LLGL::RasterizerDescriptor, [82](#)
- Snapshot
  - LLGL, [27](#)
- sourceCode
  - LLGL::ShaderSource::GLSL, [64](#)
  - LLGL::ShaderSource::HLSL, [68](#)
- sourceGLSL
  - LLGL::ShaderSource, [134](#)
- sourceHLSL
  - LLGL::ShaderSource, [135](#)
- Space
  - LLGL, [27](#)
- SrcAlpha
  - LLGL, [23](#)
- srcAlpha
  - LLGL::BlendTargetDescriptor, [47](#)
- SrcColor
  - LLGL, [23](#)
- srcColor
  - LLGL::BlendTargetDescriptor, [47](#)
- Start
  - LLGL::Timer, [159](#)
- stateOpenGL
  - LLGL::GraphicsAPIDependentStateDescriptor, [65](#)
- Static
  - LLGL, [24](#)
- StdErr
  - LLGL::Log, [43](#)
- StdOut
  - LLGL::Log, [43](#)
- Stencil
  - LLGL::ClearBuffersFlags, [48](#)
- stencil
  - LLGL::GraphicsPipelineDescriptor, [67](#)
- stencilFailOp
  - LLGL::StencilFaceDescriptor, [142](#)
- StencilOp
  - LLGL, [34](#)
- Stop
  - LLGL::Timer, [160](#)
- StorageBuffer.h, [196](#)

- StorageBufferDescriptor
  - LLGL::StorageBufferDescriptor, [144](#)
- StorageBufferType
  - LLGL, [34](#)
- StreamOutOverflow
  - LLGL, [32](#)
- StreamOutPrimitivesWritten
  - LLGL, [32](#)
- StructuredBuffer
  - LLGL, [35](#)
- SubTextureDescriptor
  - LLGL::SubTextureDescriptor, [146](#)
- Subtract
  - LLGL, [23](#)
- SwapChainMode
  - LLGL, [35](#)
- swapChainMode
  - LLGL::VideoModeDescriptor, [168](#)
- SyncGPU
  - LLGL::RenderContext, [95](#)
- T
  - LLGL, [28](#)
- Tab
  - LLGL, [26](#)
- target
  - LLGL::ShaderSource::HLSL, [68](#)
- targets
  - LLGL::BlendDescriptor, [46](#)
- TessControl
  - LLGL, [34](#)
- TessControlShaderInvocations
  - LLGL, [32](#)
- TessControlStage
  - LLGL::ShaderStageFlags, [135](#)
- TessEvaluation
  - LLGL, [34](#)
- TessEvaluationShaderInvocations
  - LLGL, [32](#)
- TessEvaluationStage
  - LLGL::ShaderStageFlags, [135](#)
- testEnabled
  - LLGL::DepthDescriptor, [62](#)
  - LLGL::StencilDescriptor, [141](#)
- Texture.h, [196](#)
- Texture1DArray
  - LLGL, [37](#)
- texture1DDesc
  - LLGL::SubTextureDescriptor, [146](#)
  - LLGL::TextureDescriptor, [157](#)
- Texture1D
  - LLGL, [37](#)
- Texture2DArray
  - LLGL, [37](#)
- texture2DDesc
  - LLGL::SubTextureDescriptor, [147](#)
  - LLGL::TextureDescriptor, [157](#)
- Texture2D
  - LLGL, [37](#)
- texture3DDesc
  - LLGL::SubTextureDescriptor, [147](#)
  - LLGL::TextureDescriptor, [157](#)
- Texture3D
  - LLGL, [37](#)
- TextureCube
  - LLGL, [37](#)
- TextureCubeArray
  - LLGL, [37](#)
- textureCubeDesc
  - LLGL::SubTextureDescriptor, [147](#)
  - LLGL::TextureDescriptor, [157](#)
- TextureDescriptor
  - LLGL::TextureDescriptor, [157](#)
- TextureFilter
  - LLGL, [35](#)
- TextureFlags.h, [197](#)
- TextureFormat
  - LLGL, [35](#)
- TextureType
  - LLGL, [37](#)
- TextureWrap
  - LLGL, [37](#)
- textureWrapU
  - LLGL::SamplerDescriptor, [123](#)
- textureWrapV
  - LLGL::SamplerDescriptor, [123](#)
- textureWrapW
  - LLGL::SamplerDescriptor, [124](#)
- threadCount
  - LLGL::RenderSystemConfiguration, [118](#)
- TimeElapsed
  - LLGL, [32](#)
- Timer.h, [198](#)
- title
  - LLGL::WindowDescriptor, [175](#)
- TriangleFan
  - LLGL, [31](#)
- TriangleList
  - LLGL, [31](#)
- TriangleListAdjacency
  - LLGL, [31](#)
- TriangleStrip
  - LLGL, [31](#)
- TriangleStripAdjacency
  - LLGL, [31](#)
- TripleBuffering
  - LLGL, [35](#)
- type
  - LLGL::QueryDescriptor, [80](#)
  - LLGL::StorageBufferDescriptor, [144](#)
  - LLGL::StorageBufferViewDescriptor, [145](#)
  - LLGL::TextureDescriptor, [157](#)
  - LLGL::UniformDescriptor, [160](#)
- Types.h, [198](#)
- U
  - LLGL, [28](#)
- UInt16

- LLGL, [25](#)
- UInt32
  - LLGL, [25](#)
- UInt8
  - LLGL, [25](#)
- Undefined
  - LLGL, [37](#)
- UniformType
  - LLGL, [37](#)
- Unknown
  - LLGL, [36](#)
- UnlockShaderUniform
  - LLGL::ShaderProgram, [133](#)
- UnmapStorageBuffer
  - LLGL::RenderContext, [95](#)
- UnsetRenderTarget
  - LLGL::RenderContext, [95](#)
- UnsupportedFeature
  - LLGL, [26](#)
- Up
  - LLGL, [27](#)
- UpperLeft
  - LLGL, [33](#)
- usage
  - LLGL::ConstantBufferDescriptor, [60](#)
  - LLGL::IndexBufferDescriptor, [72](#)
  - LLGL::StorageBufferDescriptor, [144](#)
  - LLGL::VertexBufferDescriptor, [163](#)
- V
  - LLGL, [28](#)
- ValueType
  - LLGL::RenderingProfiler::Counter, [61](#)
- Vendor
  - LLGL, [33](#)
- vendor
  - LLGL::VideoAdapterDescriptor, [166](#)
- Version
  - LLGL, [33](#)
- version
  - LLGL::ProfileOpenGLDescriptor, [78](#)
- Vertex
  - LLGL, [34](#)
- VertexAttribute.h, [199](#)
- VertexBuffer.h, [199](#)
- VertexBufferDescriptor
  - LLGL::VertexBufferDescriptor, [163](#)
- vertexFormat
  - LLGL::VertexBufferDescriptor, [163](#)
- VertexFormat.h, [200](#)
- VertexShaderInvocations
  - LLGL, [32](#)
- VertexStage
  - LLGL::ShaderStageFlags, [135](#)
- VerticesSubmitted
  - LLGL, [32](#)
- VideoAdapter.h, [200](#)
- videoMemory
  - LLGL::VideoAdapterDescriptor, [167](#)
- videoMode
  - LLGL::RenderContextDescriptor, [96](#)
- Viewport
  - LLGL::Viewport, [170](#)
- visible
  - LLGL::WindowDescriptor, [176](#)
- visual
  - LLGL::NativeContextHandle, [76](#)
  - LLGL::NativeHandle, [77](#)
- VolumeDown
  - LLGL, [29](#)
- VolumeMute
  - LLGL, [29](#)
- VolumeUp
  - LLGL, [29](#)
- vsync
  - LLGL::RenderContextDescriptor, [96](#)
- W
  - LLGL, [28](#)
- Wait
  - LLGL, [33](#)
- WaitInverted
  - LLGL, [33](#)
- WarnError
  - LLGL::ShaderCompileFlags, [127](#)
- WarningType
  - LLGL, [38](#)
- width
  - LLGL::Scissor, [125](#)
  - LLGL::SubTextureDescriptor::Texture1DDescriptor, [149](#)
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, [151](#)
  - LLGL::SubTextureDescriptor::Texture3DDescriptor, [153](#)
  - LLGL::SubTextureDescriptor::TextureCubeDescriptor, [154](#)
  - LLGL::TextureDescriptor::Texture1DDescriptor, [150](#)
  - LLGL::TextureDescriptor::Texture2DDescriptor, [150](#)
  - LLGL::TextureDescriptor::Texture3DDescriptor, [152](#)
  - LLGL::TextureDescriptor::TextureCubeDescriptor, [155](#)
  - LLGL::VideoDisplayMode, [167](#)
  - LLGL::Viewport, [170](#)
- Win32NativeHandle.h, [200](#)
- Window
  - LLGL::Window::EventListener, [64](#)
- window
  - LLGL::NativeHandle, [77](#)
- Window.h, [201](#)
- windowContext
  - LLGL::WindowDescriptor, [176](#)
- Wireframe
  - LLGL, [30](#)
- WriteConstantBuffer

- LLGL::RenderSystem, [116](#)
- writeConstantBuffer
  - LLGL::RenderingProfiler, [105](#)
- writeEnabled
  - LLGL::DepthDescriptor, [62](#)
- WriteIndexBuffer
  - LLGL::RenderSystem, [116](#)
- writeIndexBuffer
  - LLGL::RenderingProfiler, [105](#)
- writeMask
  - LLGL::StencilFaceDescriptor, [142](#)
- WriteOnly
  - LLGL, [24](#)
- WriteStorageBuffer
  - LLGL::RenderSystem, [116](#)
- writeStorageBuffer
  - LLGL::RenderingProfiler, [105](#)
- WriteTexture
  - LLGL::RenderSystem, [116](#)
- WriteVertexBuffer
  - LLGL::RenderSystem, [117](#)
- writeVertexBuffer
  - LLGL::RenderingProfiler, [105](#)
- X
  - LLGL, [28](#)
- x
  - LLGL::Scissor, [125](#)
  - LLGL::SubTextureDescriptor::Texture1DDescriptor, [149](#)
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, [151](#)
  - LLGL::SubTextureDescriptor::Texture3DDescriptor, [153](#)
  - LLGL::SubTextureDescriptor::TextureCube↔Descriptor, [154](#)
  - LLGL::Viewport, [170](#)
- XButton1
  - LLGL, [26](#)
- XButton2
  - LLGL, [26](#)
- XNeg
  - LLGL, [23](#)
- XPos
  - LLGL, [23](#)
- Y
  - LLGL, [28](#)
- y
  - LLGL::Scissor, [125](#)
  - LLGL::SubTextureDescriptor::Texture2DDescriptor, [151](#)
  - LLGL::SubTextureDescriptor::Texture3DDescriptor, [153](#)
  - LLGL::SubTextureDescriptor::TextureCube↔Descriptor, [155](#)
  - LLGL::Viewport, [170](#)
- YNeg
  - LLGL, [23](#)
- YPos
  - LLGL, [23](#)
- Z
  - LLGL, [28](#)
- z
  - LLGL::SubTextureDescriptor::Texture3DDescriptor, [153](#)
- ZNeg
  - LLGL, [23](#)
- ZPos
  - LLGL, [23](#)
- Zero
  - LLGL, [23](#), [34](#)
- ZeroToOne
  - LLGL, [24](#)
- Zoom
  - LLGL, [30](#)