LLGL

0.01 Beta

Generated by Doxygen 1.8.11

Contents

1 LLGL 0.01 Beta Documentation **1**

2 Module Index **3**

2.1 Modules 3

3 Namespace Index **5**

3.1 Namespace List 5

4 Hierarchical Index **7**

4.1 Class Hierarchy 7

5 Class Index **9**

5.1 Class List 9

6 File Index **13**

6.1 File List 13

7 Module Documentation **15**

7.1 Global utility functions, especially to fill descriptor structures. 15

7.1.1 Detailed Description 16

7.1.2 Function Documentation 16

7.1.2.1 ConstantBufferDesc(unsigned int size, long flags=BufferFlags::DynamicUsage) . 16

7.1.2.2 IndexBufferDesc(unsigned int size, const IndexFormat &indexFormat, long flags=0) 16

7.1.2.3 StorageBufferDesc(unsigned int size, const StorageBufferType storageType, unsigned int stride, long flags=BufferFlags::MapReadAccess" | BufferFlags::MapWriteAccess) 16

7.1.2.4 Texture1DArrayDesc(TextureFormat format, unsigned int width, unsigned int layers) 16

7.1.2.5 Texture1DDesc(TextureFormat format, unsigned int width) . . . 16
7.1.2.6 Texture2DArrayDesc(TextureFormat format, unsigned int width, unsigned int height, unsigned int layers) . . . 16
7.1.2.7 Texture2DDesc(TextureFormat format, unsigned int width, unsigned int height) . 16
7.1.2.8 Texture2DMSArrayDesc(TextureFormat format, unsigned int width, unsigned int height, unsigned int layers, unsigned int samples, bool fixedSamples=true) . . . 17
7.1.2.9 Texture2DMSDesc(TextureFormat format, unsigned int width, unsigned int height, unsigned int samples, bool fixedSamples=true) . . . 17
7.1.2.10 Texture3DDesc(TextureFormat format, unsigned int width, unsigned int height, unsigned int depth) . . . 17
7.1.2.11 TextureCubeArrayDesc(TextureFormat format, unsigned int width, unsigned int height, unsigned int layers) . . . 17
7.1.2.12 TextureCubeDesc(TextureFormat format, unsigned int width, unsigned int height) 17
7.1.2.13 VertexBufferDesc(unsigned int size, const VertexFormat &vertexFormat, long flags=0) . . . 17

8 Namespace Documentation **19**

8.1 LLGL Namespace Reference . . . 19
8.1.1 Typedef Documentation . . . 28
8.1.1.1 ByteBuffer . . . 28
8.1.1.2 ColorRGB . . . 29
8.1.1.3 ColorRGBA . . . 29
8.1.1.4 ColorRGBAb . . . 29
8.1.1.5 ColorRGBAd . . . 29
8.1.1.6 ColorRGBAf . . . 29
8.1.1.7 ColorRGBAT . . . 29
8.1.1.8 ColorRGBAub . . . 29
8.1.1.9 ColorRGBb . . . 29
8.1.1.10 ColorRGBd . . . 29
8.1.1.11 ColorRGBf . . . 29
8.1.1.12 ColorRGBT . . . 29
8.1.1.13 ColorRGBub . . . 29
8.1.1.14 DebugCallback . . . 29
8.1.1.15 Point . . . 30

8.1.1.16 Size . . . 30
8.1.2 Enumeration Type Documentation . . . 30
8.1.2.1 AxisDirection . . . 30
8.1.2.2 BlendArithmetic . . . 30
8.1.2.3 BlendOp . . . 31
8.1.2.4 BufferCPUAccess . . . 31
8.1.2.5 BufferType . . . 32
8.1.2.6 ClippingRange . . . 32
8.1.2.7 CompareOp . . . 32
8.1.2.8 CullMode . . . 32
8.1.2.9 DataType . . . 33
8.1.2.10 ErrorType . . . 33
8.1.2.11 ImageFormat . . . 33
8.1.2.12 Key . . . 34
8.1.2.13 LogicOp . . . 37
8.1.2.14 OpenGLVersion . . . 38
8.1.2.15 PolygonMode . . . 38
8.1.2.16 PrimitiveTopology . . . 39
8.1.2.17 PrimitiveType . . . 40
8.1.2.18 QueryType . . . 41
8.1.2.19 RenderConditionMode . . . 41
8.1.2.20 ScreenOrigin . . . 42
8.1.2.21 ShaderType . . . 42
8.1.2.22 ShadingLanguage . . . 42
8.1.2.23 StencilOp . . . 43
8.1.2.24 StorageBufferType . . . 43
8.1.2.25 SwapChainMode . . . 44
8.1.2.26 TextureFilter . . . 44
8.1.2.27 TextureFormat . . . 44
8.1.2.28 TextureType . . . 46

8.1.2.29 TextureWrap . . . 46
8.1.2.30 UniformType . . . 46
8.1.2.31 VectorType . . . 47
8.1.2.32 WarningType . . . 47
8.1.3 Function Documentation . . . 48
8.1.3.1 CompareSWO(const VideoDisplayMode &lhs, const VideoDisplayMode &rhs) . . . 48
8.1.3.2 ConvertImageBuffer(ImageFormat srcFormat, DataType srcDataType, const void *srcBuffer, std::size_t srcBufferSize, ImageFormat dstFormat, DataType dst↩ DataType, std::size_t threadCount=0) . . . 48
8.1.3.3 DataTypeSize(const DataType dataType) . . . 48
8.1.3.4 ImageFormatSize(const ImageFormat imageFormat) . . . 49
8.1.3.5 IsArrayTexture(const TextureType type) . . . 49
8.1.3.6 IsCompressedFormat(const ImageFormat format) . . . 49
8.1.3.7 IsCompressedFormat(const TextureFormat format) . . . 49
8.1.3.8 IsDepthStencilFormat(const ImageFormat format) . . . 50
8.1.3.9 IsMultiSampleTexture(const TextureType type) . . . 50
8.1.3.10 MaxColorValue() . . . 50
8.1.3.11 MaxColorValue< bool >() . . . 50
8.1.3.12 MaxColorValue< unsigned char >() . . . 50
8.1.3.13 NumMipLevels(unsigned int width, unsigned int height=1, unsigned int depth=1) 50
8.1.3.14 operator"!=(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 51
8.1.3.15 operator"!=(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . 51
8.1.3.16 operator"!=(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) . . . 51
8.1.3.17 operator"!=(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 51
8.1.3.18 operator"!=(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.19 operator*(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.20 operator*(const Color< T, N > &lhs, const T &rhs) . . . 51
8.1.3.21 operator*(const T &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.22 operator+(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.23 operator-(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.24 operator/(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51

8.1.3.25 operator/(const Color< T, N > &lhs, const T &rhs) . . . 51
8.1.3.26 operator==(const VideoDisplayMode &lhs, const VideoDisplayMode &rhs) . . . 51
8.1.3.27 operator==(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 51
8.1.3.28 operator==(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . 51
8.1.3.29 operator==(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) . . . 51
8.1.3.30 operator==(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 51
8.1.3.31 operator==(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 51
8.1.3.32 VectorTypeFormat(const VectorType vectorType, DataType &dataType, unsigned int &components) . . . 51
8.1.3.33 VectorTypeSize(const VectorType vectorType) . . . 52
8.2 LLGL::Desktop Namespace Reference . . . 52
8.2.1 Function Documentation . . . 52
8.2.1.1 GetColorDepth() . . . 52
8.2.1.2 GetResolution() . . . 52
8.2.1.3 ResetVideoMode() . . . 52
8.2.1.4 SetVideoMode(const VideoModeDescriptor &videoMode) . . . 52
8.3 LLGL::Log Namespace Reference . . . 53
8.3.1 Function Documentation . . . 53
8.3.1.1 SetStdErr(std::ostream &stream) . . . 53
8.3.1.2 SetStdOut(std::ostream &stream) . . . 53
8.3.1.3 StdErr() . . . 53
8.3.1.4 StdOut() . . . 53
8.4 LLGL::Version Namespace Reference . . . 53
8.4.1 Function Documentation . . . 54
8.4.1.1 GetID() . . . 54
8.4.1.2 GetMajor() . . . 54
8.4.1.3 GetMinor() . . . 54
8.4.1.4 GetRevision() . . . 54
8.4.1.5 GetStatus() . . . 54
8.4.1.6 GetString() . . . 54

9 Class Documentation **55**

9.1 LLGL::BlendDescriptor Struct Reference . . . 55

9.1.1 Detailed Description . . . 55

9.1.2 Member Data Documentation . . . 55

9.1.2.1 blendEnabled . . . 55

9.1.2.2 blendFactor . . . 56

9.1.2.3 targets . . . 56

9.2 LLGL::BlendTargetDescriptor Struct Reference . . . 56

9.2.1 Detailed Description . . . 57

9.2.2 Member Data Documentation . . . 57

9.2.2.1 alphaArithmetic . . . 57

9.2.2.2 colorArithmetic . . . 57

9.2.2.3 colorMask . . . 57

9.2.2.4 destAlpha . . . 57

9.2.2.5 destColor . . . 57

9.2.2.6 srcAlpha . . . 57

9.2.2.7 srcColor . . . 57

9.3 LLGL::Buffer Class Reference . . . 57

9.3.1 Detailed Description . . . 58

9.3.2 Constructor & Destructor Documentation . . . 58

9.3.2.1 Buffer(const Buffer &)=delete . . . 58

9.3.2.2 ~Buffer() . . . 58

9.3.2.3 Buffer(const BufferType type) . . . 58

9.3.3 Member Function Documentation . . . 58

9.3.3.1 GetType() const . . . 58

9.3.3.2 operator=(const Buffer &)=delete . . . 58

9.4 LLGL::BufferArray Class Reference . . . 58

9.4.1 Detailed Description . . . 59

9.4.2 Constructor & Destructor Documentation . . . 59

9.4.2.1 BufferArray(const BufferArray &)=delete . . . 59

9.4.2.2 ~BufferArray() . . . 59
9.4.2.3 BufferArray(const BufferType type) . . . 59
9.4.3 Member Function Documentation . . . 59
9.4.3.1 GetType() const . . . 59
9.4.3.2 operator=(const BufferArray &)=delete . . . 59
9.5 LLGL::BufferDescriptor Struct Reference . . . 59
9.5.1 Detailed Description . . . 60
9.5.2 Member Data Documentation . . . 60
9.5.2.1 flags . . . 60
9.5.2.2 indexBuffer . . . 60
9.5.2.3 size . . . 61
9.5.2.4 storageBuffer . . . 61
9.5.2.5 type . . . 61
9.5.2.6 vertexBuffer . . . 61
9.6 LLGL::BufferFlags Struct Reference . . . 61
9.6.1 Detailed Description . . . 61
9.6.2 Member Enumeration Documentation . . . 62
9.6.2.1 anonymous enum . . . 62
9.7 LLGL::ClearFlags Struct Reference . . . 62
9.7.1 Detailed Description . . . 62
9.7.2 Member Enumeration Documentation . . . 63
9.7.2.1 anonymous enum . . . 63
9.8 LLGL::Color< T, N > Class Template Reference . . . 63
9.8.1 Detailed Description . . . 64
9.8.2 Constructor & Destructor Documentation . . . 64
9.8.2.1 Color() . . . 64
9.8.2.2 Color(const Color< T, N > &rhs) . . . 64
9.8.2.3 Color(Gs::UninitializeTag) . . . 64
9.8.3 Member Function Documentation . . . 64
9.8.3.1 Cast() const . . . 64

9.8.3.2 operator*=(const Color< T, N > &rhs) . . . 64
9.8.3.3 operator*=(const T &rhs) . . . 64
9.8.3.4 operator+=(const Color< T, N > &rhs) . . . 65
9.8.3.5 operator-() const . . . 65
9.8.3.6 operator-=(const Color< T, N > &rhs) . . . 65
9.8.3.7 operator/=(const Color< T, N > &rhs) . . . 65
9.8.3.8 operator/=(const T &rhs) . . . 65
9.8.3.9 operator[](std::size_t component) . . . 65
9.8.3.10 operator[](std::size_t component) const . . . 65
9.8.3.11 Ptr() . . . 65
9.8.3.12 Ptr() const . . . 65
9.8.4 Member Data Documentation . . . 65
9.8.4.1 components . . . 65
9.9 LLGL::Color< T, 3u > Class Template Reference . . . 66
9.9.1 Detailed Description . . . 66
9.9.2 Constructor & Destructor Documentation . . . 67
9.9.2.1 Color() . . . 67
9.9.2.2 Color(const Color< T, 3 > &rhs) . . . 67
9.9.2.3 Color(const T &scalar) . . . 67
9.9.2.4 Color(const T &r, const T &g, const T &b) . . . 67
9.9.2.5 Color(Gs::UninitializeTag) . . . 67
9.9.3 Member Function Documentation . . . 67
9.9.3.1 Cast() const . . . 67
9.9.3.2 operator*=(const Color< T, 3 > &rhs) . . . 67
9.9.3.3 operator*=(const T &rhs) . . . 67
9.9.3.4 operator+=(const Color< T, 3 > &rhs) . . . 67
9.9.3.5 operator-() const . . . 67
9.9.3.6 operator-=(const Color< T, 3 > &rhs) . . . 67
9.9.3.7 operator/=(const Color< T, 3 > &rhs) . . . 67
9.9.3.8 operator/=(const T &rhs) . . . 67

9.9.3.9 operator[](std::size_t component) . . . 67
9.9.3.10 operator[](std::size_t component) const . . . 68
9.9.3.11 Ptr() . . . 68
9.9.3.12 Ptr() const . . . 68
9.9.4 Member Data Documentation . . . 68
9.9.4.1 b . . . 68
9.9.4.2 components . . . 68
9.9.4.3 g . . . 68
9.9.4.4 r . . . 68
9.10 LLGL::Color< T, 4u > Class Template Reference . . . 68
9.10.1 Detailed Description . . . 69
9.10.2 Constructor & Destructor Documentation . . . 70
9.10.2.1 Color() . . . 70
9.10.2.2 Color(const Color< T, 4 > &rhs) . . . 70
9.10.2.3 Color(const T &brightness) . . . 70
9.10.2.4 Color(const T &r, const T &g, const T &b) . . . 70
9.10.2.5 Color(const T &r, const T &g, const T &b, const T &a) . . . 70
9.10.2.6 Color(Gs::UninitializeTag) . . . 70
9.10.3 Member Function Documentation . . . 70
9.10.3.1 Cast() const . . . 70
9.10.3.2 operator*=(const Color< T, 4 > &rhs) . . . 70
9.10.3.3 operator*=(const T &rhs) . . . 70
9.10.3.4 operator+=(const Color< T, 4 > &rhs) . . . 70
9.10.3.5 operator-() const . . . 70
9.10.3.6 operator-=(const Color< T, 4 > &rhs) . . . 70
9.10.3.7 operator/=(const Color< T, 4 > &rhs) . . . 70
9.10.3.8 operator/=(const T &rhs) . . . 71
9.10.3.9 operator[](std::size_t component) . . . 71
9.10.3.10 operator[](std::size_t component) const . . . 71
9.10.3.11 Ptr() . . . 71

9.10.3.12 Ptr() const . . . 71
9.10.4 Member Data Documentation . . . 71
9.10.4.1 a . . . 71
9.10.4.2 b . . . 71
9.10.4.3 components . . . 71
9.10.4.4 g . . . 71
9.10.4.5 r . . . 71
9.11 LLGL::CommandBuffer Class Reference . . . 72
9.11.1 Detailed Description . . . 74
9.11.2 Constructor & Destructor Documentation . . . 74
9.11.2.1 CommandBuffer(const CommandBuffer &)=delete . . . 74
9.11.2.2 ~CommandBuffer() . . . 74
9.11.2.3 CommandBuffer()=default . . . 74
9.11.3 Member Function Documentation . . . 74
9.11.3.1 BeginQuery(Query &query)=0 . . . 74
9.11.3.2 BeginRenderCondition(Query &query, const RenderConditionMode mode)=0 . . . 75
9.11.3.3 BeginStreamOutput(const PrimitiveType primitiveType)=0 . . . 75
9.11.3.4 Clear(long flags)=0 . . . 75
9.11.3.5 Dispatch(unsigned int groupSizeX, unsigned int groupSizeY, unsigned int groupSizeZ)=0 . . . 76
9.11.3.6 Draw(unsigned int numVertices, unsigned int firstVertex)=0 . . . 76
9.11.3.7 DrawIndexed(unsigned int numVertices, unsigned int firstIndex)=0 . . . 76
9.11.3.8 DrawIndexed(unsigned int numVertices, unsigned int firstIndex, int vertexOffset)=0 . . . 76
9.11.3.9 DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex)=0 . . . 77
9.11.3.10 DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset)=0 . . . 77
9.11.3.11 DrawIndexedInstanced(unsigned int numVertices, unsigned int numInstances, unsigned int firstIndex, int vertexOffset, unsigned int instanceOffset)=0 . . . 77
9.11.3.12 DrawInstanced(unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances)=0 . . . 77
9.11.3.13 DrawInstanced(unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances, unsigned int instanceOffset)=0 . . . 77

9.11.3.14 EndQuery(Query &query)=0 . . . 78

9.11.3.15 EndRenderCondition()=0 . . . 78

9.11.3.16 EndStreamOutput()=0 . . . 78

9.11.3.17 operator=(const CommandBuffer &)=delete . . . 78

9.11.3.18 QueryResult(Query &query, std::uint64_t &result)=0 . . . 78

9.11.3.19 SetClearColor(const ColorRGBAf &color)=0 . . . 79

9.11.3.20 SetClearDepth(float depth)=0 . . . 79

9.11.3.21 SetClearStencil(int stencil)=0 . . . 79

9.11.3.22 SetComputePipeline(ComputePipeline &computePipeline)=0 . . . 79

9.11.3.23 SetConstantBuffer(Buffer &buffer, unsigned int slot, long shaderStageFlags=↩ ShaderStageFlags::AllStages)=0 . . . 79

9.11.3.24 SetConstantBufferArray(BufferArray &bufferArray, unsigned int startSlot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . 80

9.11.3.25 SetGraphicsAPIDependentState(const GraphicsAPIDependentStateDescriptor &state)=0 . . . 80

9.11.3.26 SetGraphicsPipeline(GraphicsPipeline &graphicsPipeline)=0 . . . 80

9.11.3.27 SetIndexBuffer(Buffer &buffer)=0 . . . 81

9.11.3.28 SetRenderTarget(RenderTarget &renderTarget)=0 . . . 81

9.11.3.29 SetRenderTarget(RenderContext &renderContext)=0 . . . 81

9.11.3.30 SetSampler(Sampler &sampler, unsigned int slot, long shaderStageFlags=↩ ShaderStageFlags::AllStages)=0 . . . 82

9.11.3.31 SetSamplerArray(SamplerArray &samplerArray, unsigned int startSlot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . 82

9.11.3.32 SetScissor(const Scissor &scissor)=0 . . . 82

9.11.3.33 SetScissorArray(unsigned int numScissors, const Scissor *scissorArray)=0 . . . 82

9.11.3.34 SetStorageBuffer(Buffer &buffer, unsigned int slot, long shaderStageFlags=↩ ShaderStageFlags::AllStages)=0 . . . 83

9.11.3.35 SetStorageBufferArray(BufferArray &bufferArray, unsigned int startSlot, long shaderStageFlags=ShaderStageFlags::AllStages)=0 . . . 83

9.11.3.36 SetStreamOutputBuffer(Buffer &buffer)=0 . . . 84

9.11.3.37 SetStreamOutputBufferArray(BufferArray &bufferArray)=0 . . . 84

9.11.3.38 SetTexture(Texture &texture, unsigned int slot, long shaderStageFlags=Shader↩ StageFlags::AllStages)=0 . . . 84

9.11.3.39 SetTextureArray(TextureArray &textureArray, unsigned int startSlot, long shader↩ StageFlags=ShaderStageFlags::AllStages)=0 . . . 84
9.11.3.40 SetVertexBuffer(Buffer &buffer)=0 . . . 85
9.11.3.41 SetVertexBufferArray(BufferArray &bufferArray)=0 . . . 85
9.11.3.42 SetViewport(const Viewport &viewport)=0 . . . 85
9.11.3.43 SetViewportArray(unsigned int numViewports, const Viewport *viewportArray)=0 85
9.11.3.44 SyncGPU()=0 . . . 86
9.12 LLGL::ComputePipeline Class Reference . . . 86
9.12.1 Detailed Description . . . 86
9.12.2 Constructor & Destructor Documentation . . . 86
9.12.2.1 ~ComputePipeline() . . . 86
9.13 LLGL::ComputePipelineDescriptor Struct Reference . . . 87
9.13.1 Detailed Description . . . 87
9.13.2 Constructor & Destructor Documentation . . . 87
9.13.2.1 ComputePipelineDescriptor()=default . . . 87
9.13.2.2 ComputePipelineDescriptor(ShaderProgram *shaderProgram) . . . 87
9.13.3 Member Data Documentation . . . 87
9.13.3.1 shaderProgram . . . 87
9.14 LLGL::ConstantBufferViewDescriptor Struct Reference . . . 88
9.14.1 Detailed Description . . . 88
9.14.2 Member Data Documentation . . . 88
9.14.2.1 index . . . 88
9.14.2.2 name . . . 88
9.14.2.3 size . . . 88
9.15 LLGL::RenderingProfiler::Counter Class Reference . . . 88
9.15.1 Detailed Description . . . 89
9.15.2 Member Typedef Documentation . . . 89
9.15.2.1 ValueType . . . 89
9.15.3 Member Function Documentation . . . 89
9.15.3.1 Count() const . . . 89
9.15.3.2 Inc() . . . 89

9.15.3.3 Inc(ValueType value) . . . 89
9.15.3.4 operator unsigned int() const . . . 89
9.15.3.5 Reset() . . . 90
9.16 LLGL::DepthDescriptor Struct Reference . . . 90
9.16.1 Detailed Description . . . 90
9.16.2 Member Data Documentation . . . 90
9.16.2.1 compareOp . . . 90
9.16.2.2 testEnabled . . . 90
9.16.2.3 writeEnabled . . . 91
9.17 LLGL::Window::EventListener Class Reference . . . 91
9.17.1 Constructor & Destructor Documentation . . . 92
9.17.1.1 ~EventListener() . . . 92
9.17.2 Member Function Documentation . . . 92
9.17.2.1 OnChar(Window &sender, wchar_t chr) . . . 92
9.17.2.2 OnDoubleClick(Window &sender, Key keyCode) . . . 92
9.17.2.3 OnGlobalMotion(Window &sender, const Point &motion) . . . 92
9.17.2.4 OnKeyDown(Window &sender, Key keyCode) . . . 92
9.17.2.5 OnKeyUp(Window &sender, Key keyCode) . . . 92
9.17.2.6 OnLocalMotion(Window &sender, const Point &position) . . . 92
9.17.2.7 OnProcessEvents(Window &sender) . . . 92
9.17.2.8 OnQuit(Window &sender) . . . 92
9.17.2.9 OnResize(Window &sender, const Size &clientAreaSize) . . . 92
9.17.2.10 OnWheelMotion(Window &sender, int motion) . . . 92
9.17.3 Friends And Related Function Documentation . . . 92
9.17.3.1 Window . . . 92
9.18 LLGL::GraphicsAPIDependentStateDescriptor Union Reference . . . 93
9.18.1 Detailed Description . . . 93
9.18.2 Constructor & Destructor Documentation . . . 93
9.18.2.1 GraphicsAPIDependentStateDescriptor() . . . 93
9.18.3 Member Data Documentation . . . 93

9.18.3.1 stateOpenGL . . . 93
9.19 LLGL::GraphicsPipeline Class Reference . . . 94
9.19.1 Detailed Description . . . 94
9.19.2 Constructor & Destructor Documentation . . . 94
9.19.2.1 ~GraphicsPipeline() . . . 94
9.20 LLGL::GraphicsPipelineDescriptor Struct Reference . . . 94
9.20.1 Detailed Description . . . 95
9.20.2 Member Data Documentation . . . 95
9.20.2.1 blend . . . 95
9.20.2.2 depth . . . 95
9.20.2.3 primitiveTopology . . . 95
9.20.2.4 rasterizer . . . 95
9.20.2.5 shaderProgram . . . 95
9.20.2.6 stencil . . . 96
9.21 LLGL::ImageDescriptor Struct Reference . . . 96
9.21.1 Detailed Description . . . 96
9.21.2 Constructor & Destructor Documentation . . . 97
9.21.2.1 ImageDescriptor()=default . . . 97
9.21.2.2 ImageDescriptor(ImageFormat format, DataType dataType, const void *buffer) . . . 97
9.21.2.3 ImageDescriptor(ImageFormat format, const void *buffer, unsigned int compressedSize) . . . 97
9.21.3 Member Function Documentation . . . 97
9.21.3.1 GetElementSize() const . . . 97
9.21.4 Member Data Documentation . . . 97
9.21.4.1 buffer . . . 97
9.21.4.2 compressedSize . . . 97
9.21.4.3 dataType . . . 97
9.21.4.4 format . . . 97
9.22 LLGL::BufferDescriptor::IndexBufferDescriptor Struct Reference . . . 98
9.22.1 Member Data Documentation . . . 98
9.22.1.1 format . . . 98

9.23 LLGL::IndexFormat Class Reference . . . 98
9.23.1 Constructor & Destructor Documentation . . . 99
9.23.1.1 IndexFormat()=default . . . 99
9.23.1.2 IndexFormat(const DataType dataType) . . . 99
9.23.2 Member Function Documentation . . . 99
9.23.2.1 GetDataType() const . . . 99
9.23.2.2 GetFormatSize() const . . . 99
9.24 LLGL::Input Class Reference . . . 99
9.24.1 Constructor & Destructor Documentation . . . 100
9.24.1.1 Input() . . . 100
9.24.2 Member Function Documentation . . . 100
9.24.2.1 GetEnteredChars() const . . . 100
9.24.2.2 GetMouseMotion() const . . . 100
9.24.2.3 GetMousePosition() const . . . 100
9.24.2.4 GetWheelMotion() const . . . 100
9.24.2.5 KeyDoubleClick(Key keyCode) const . . . 100
9.24.2.6 KeyDown(Key keyCode) const . . . 100
9.24.2.7 KeyPressed(Key keyCode) const . . . 100
9.24.2.8 KeyUp(Key keyCode) const . . . 101
9.25 LLGL::RenderingDebugger::Message Class Reference . . . 101
9.25.1 Detailed Description . . . 101
9.25.2 Constructor & Destructor Documentation . . . 102
9.25.2.1 Message()=default . . . 102
9.25.2.2 Message(const Message &)=default . . . 102
9.25.2.3 Message(const std::string &text, const std::string &source) . . . 102
9.25.3 Member Function Documentation . . . 102
9.25.3.1 Block() . . . 102
9.25.3.2 BlockAfter(std::size_t occurrences) . . . 102
9.25.3.3 GetOccurrences() const . . . 102
9.25.3.4 GetSource() const . . . 102

9.25.3.5 GetText() const . . . 102
9.25.3.6 IncOccurrence() . . . 102
9.25.3.7 IsBlocked() const . . . 102
9.25.3.8 operator=(const Message &)=default . . . 102
9.25.4 Friends And Related Function Documentation . . . 102
9.25.4.1 RenderingDebugger . . . 102
9.26 LLGL::MultiSamplingDescriptor Struct Reference . . . 103
9.26.1 Detailed Description . . . 103
9.26.2 Constructor & Destructor Documentation . . . 103
9.26.2.1 MultiSamplingDescriptor()=default . . . 103
9.26.2.2 MultiSamplingDescriptor(unsigned int samples) . . . 103
9.26.3 Member Function Documentation . . . 103
9.26.3.1 SampleCount() const . . . 103
9.26.4 Member Data Documentation . . . 103
9.26.4.1 enabled . . . 103
9.26.4.2 samples . . . 104
9.27 LLGL::NativeContextHandle Struct Reference . . . 104
9.27.1 Detailed Description . . . 104
9.27.2 Member Data Documentation . . . 104
9.27.2.1 colorMap . . . 104
9.27.2.2 display . . . 104
9.27.2.3 parentWindow . . . 104
9.27.2.4 parentWindow . . . 104
9.27.2.5 parentWindow . . . 104
9.27.2.6 screen . . . 104
9.27.2.7 visual . . . 104
9.28 LLGL::NativeHandle Struct Reference . . . 105
9.28.1 Detailed Description . . . 105
9.28.2 Member Data Documentation . . . 105
9.28.2.1 display . . . 105

9.28.2.2 visual . . . 105
9.28.2.3 window . . . 105
9.28.2.4 window . . . 105
9.28.2.5 window . . . 105
9.29 LLGL::ProfileOpenGLDescriptor Struct Reference . . . 105
9.29.1 Detailed Description . . . 106
9.29.2 Member Data Documentation . . . 106
9.29.2.1 coreProfile . . . 106
9.29.2.2 debugDump . . . 106
9.29.2.3 extProfile . . . 106
9.29.2.4 version . . . 106
9.30 LLGL::Query Class Reference . . . 107
9.30.1 Detailed Description . . . 107
9.30.2 Constructor & Destructor Documentation . . . 107
9.30.2.1 Query(const Query &)=delete . . . 107
9.30.2.2 ~Query() . . . 107
9.30.2.3 Query(const QueryType type) . . . 107
9.30.3 Member Function Documentation . . . 107
9.30.3.1 GetType() const . . . 107
9.30.3.2 operator=(const Query &)=delete . . . 107
9.31 LLGL::QueryDescriptor Struct Reference . . . 108
9.31.1 Detailed Description . . . 108
9.31.2 Constructor & Destructor Documentation . . . 108
9.31.2.1 QueryDescriptor()=default . . . 108
9.31.2.2 QueryDescriptor(QueryType type, bool renderCondition=false) . . . 108
9.31.3 Member Data Documentation . . . 108
9.31.3.1 renderCondition . . . 108
9.31.3.2 type . . . 108
9.32 LLGL::RasterizerDescriptor Struct Reference . . . 109
9.32.1 Detailed Description . . . 109

9.32.2 Member Data Documentation . . . 109
9.32.2.1 antiAliasedLineEnabled . . . 109
9.32.2.2 conservativeRasterization . . . 109
9.32.2.3 cullMode . . . 110
9.32.2.4 depthBias . . . 110
9.32.2.5 depthBiasClamp . . . 110
9.32.2.6 depthClampEnabled . . . 110
9.32.2.7 frontCCW . . . 110
9.32.2.8 multiSampling . . . 110
9.32.2.9 polygonMode . . . 110
9.32.2.10 scissorTestEnabled . . . 110
9.32.2.11 slopeScaledDepthBias . . . 110
9.33 LLGL::RenderContext Class Reference . . . 110
9.33.1 Detailed Description . . . 111
9.33.2 Constructor & Destructor Documentation . . . 111
9.33.2.1 RenderContext(const RenderContext &)=delete . . . 111
9.33.2.2 ~RenderContext() . . . 111
9.33.2.3 RenderContext()=default . . . 111
9.33.3 Member Function Documentation . . . 111
9.33.3.1 GetVideoMode() const . . . 111
9.33.3.2 GetWindow() const . . . 112
9.33.3.3 operator=(const RenderContext &)=delete . . . 112
9.33.3.4 Present()=0 . . . 112
9.33.3.5 SetOrCreateWindow(const std::shared_ptr< Window > &window, VideoModeDescriptor &videoModeDesc, const void *windowContext) . . . 112
9.33.3.6 SetVideoMode(const VideoModeDescriptor &videoModeDesc) . . . 112
9.33.3.7 SetVsync(const VsyncDescriptor &vsyncDesc)=0 . . . 112
9.33.3.8 ShareWindowAndVideoMode(RenderContext &other) . . . 113
9.34 LLGL::RenderContextDescriptor Struct Reference . . . 113
9.34.1 Detailed Description . . . 113
9.34.2 Member Data Documentation . . . 113

9.34.2.1 debugCallback . . . 113
9.34.2.2 multiSampling . . . 113
9.34.2.3 profileOpenGL . . . 114
9.34.2.4 videoMode . . . 114
9.34.2.5 vsync . . . 114
9.35 LLGL::RendererID Struct Reference . . . 114
9.35.1 Detailed Description . . . 115
9.35.2 Member Data Documentation . . . 115
9.35.2.1 Direct3D11 . . . 115
9.35.2.2 Direct3D12 . . . 115
9.35.2.3 OpenGL . . . 115
9.35.2.4 Reserved5 . . . 115
9.35.2.5 Reserved6 . . . 115
9.35.2.6 Reserved7 . . . 115
9.35.2.7 Reserved8 . . . 115
9.35.2.8 Reserved9 . . . 115
9.35.2.9 ReservedA . . . 116
9.35.2.10 ReservedB . . . 116
9.35.2.11 ReservedC . . . 116
9.35.2.12 ReservedD . . . 116
9.35.2.13 ReservedE . . . 116
9.35.2.14 ReservedF . . . 116
9.35.2.15 Vulkan . . . 116
9.36 LLGL::RendererInfo Struct Reference . . . 116
9.36.1 Detailed Description . . . 117
9.36.2 Member Data Documentation . . . 117
9.36.2.1 deviceName . . . 117
9.36.2.2 rendererID . . . 117
9.36.2.3 rendererName . . . 117
9.36.2.4 shadingLanguageName . . . 117

9.36.2.5 vendorName . . . 118
9.37 LLGL::RenderingCaps Struct Reference . . . 118
9.37.1 Detailed Description . . . 119
9.37.2 Member Data Documentation . . . 119
9.37.2.1 clippingRange . . . 119
9.37.2.2 has3DTextures . . . 119
9.37.2.3 hasComputeShaders . . . 120
9.37.2.4 hasConservativeRasterization . . . 120
9.37.2.5 hasConstantBuffers . . . 120
9.37.2.6 hasCubeTextureArrays . . . 120
9.37.2.7 hasCubeTextures . . . 120
9.37.2.8 hasGeometryShaders . . . 120
9.37.2.9 hasInstancing . . . 121
9.37.2.10 hasMultiSampleTextures . . . 121
9.37.2.11 hasOffsetInstancing . . . 121
9.37.2.12 hasRenderTargets . . . 121
9.37.2.13 hasSamplers . . . 121
9.37.2.14 hasStorageBuffers . . . 121
9.37.2.15 hasStreamOutputs . . . 122
9.37.2.16 hasTessellationShaders . . . 122
9.37.2.17 hasTextureArrays . . . 122
9.37.2.18 hasUniforms . . . 122
9.37.2.19 hasViewportArrays . . . 122
9.37.2.20 max1DTextureSize . . . 122
9.37.2.21 max2DTextureSize . . . 122
9.37.2.22 max3DTextureSize . . . 123
9.37.2.23 maxAnisotropy . . . 123
9.37.2.24 maxComputeShaderWorkGroupSize . . . 123
9.37.2.25 maxConstantBufferSize . . . 123
9.37.2.26 maxCubeTextureSize . . . 123

9.37.2.27 maxNumComputeShaderWorkGroups . . . 123
9.37.2.28 maxNumRenderTargetAttachments . . . 123
9.37.2.29 maxNumTextureArrayLayers . . . 123
9.37.2.30 maxPatchVertices . . . 123
9.37.2.31 screenOrigin . . . 124
9.37.2.32 shadingLanguage . . . 124
9.38 LLGL::RenderingDebugger Class Reference . . . 124
9.38.1 Detailed Description . . . 125
9.38.2 Constructor & Destructor Documentation . . . 125
9.38.2.1 ~RenderingDebugger() . . . 125
9.38.2.2 RenderingDebugger()=default . . . 125
9.38.3 Member Function Documentation . . . 125
9.38.3.1 OnError(ErrorType type, Message &message) . . . 125
9.38.3.2 OnWarning(WarningType type, Message &message) . . . 125
9.38.3.3 PostError(const ErrorType type, const std::string &message) . . . 125
9.38.3.4 PostWarning(const WarningType type, const std::string &message) . . . 125
9.38.3.5 SetSource(const char *source) . . . 126
9.39 LLGL::RenderingProfiler Class Reference . . . 126
9.39.1 Detailed Description . . . 127
9.39.2 Member Function Documentation . . . 127
9.39.2.1 RecordDrawCall(const PrimitiveTopology topology, Counter::ValueType numVertices) . . . 127
9.39.2.2 RecordDrawCall(const PrimitiveTopology topology, Counter::ValueType numVertices, Counter::ValueType numInstances) . . . 127
9.39.2.3 ResetCounters() . . . 127
9.39.3 Member Data Documentation . . . 127
9.39.3.1 dispatchComputeCalls . . . 127
9.39.3.2 drawCalls . . . 128
9.39.3.3 mapBuffer . . . 128
9.39.3.4 renderedLines . . . 128
9.39.3.5 renderedPatches . . . 128

9.39.3.6 renderedPoints . . . 128
9.39.3.7 renderedTriangles . . . 128
9.39.3.8 setComputePipeline . . . 128
9.39.3.9 setConstantBuffer . . . 129
9.39.3.10 setGraphicsPipeline . . . 129
9.39.3.11 setIndexBuffer . . . 129
9.39.3.12 setRenderTarget . . . 129
9.39.3.13 setSampler . . . 129
9.39.3.14 setStorageBuffer . . . 129
9.39.3.15 setStreamOutputBuffer . . . 130
9.39.3.16 setTexture . . . 130
9.39.3.17 setVertexBuffer . . . 130
9.39.3.18 writeBuffer . . . 130
9.40 LLGL::RenderSystem Class Reference . . . 130
9.40.1 Detailed Description . . . 133
9.40.2 Constructor & Destructor Documentation . . . 133
9.40.2.1 RenderSystem(const RenderSystem &)=delete . . . 133
9.40.2.2 ∼RenderSystem() . . . 133
9.40.2.3 RenderSystem()=default . . . 133
9.40.3 Member Function Documentation . . . 133
9.40.3.1 AssertCreateBuffer(const BufferDescriptor &desc) . . . 133
9.40.3.2 AssertCreateBufferArray(unsigned int numBuffers, Buffer ∗const ∗bufferArray) . . . 134
9.40.3.3 AssertCreateSamplerArray(unsigned int numSamplers, Sampler ∗const ∗samplerArray) . . . 134
9.40.3.4 AssertCreateTextureArray(unsigned int numTextures, Texture ∗const ∗textureArray) 134
9.40.3.5 CreateBuffer(const BufferDescriptor &desc, const void ∗initialData=nullptr)=0 . . . 134
9.40.3.6 CreateBufferArray(unsigned int numBuffers, Buffer ∗const ∗bufferArray)=0 . . . 134
9.40.3.7 CreateCommandBuffer()=0 . . . 135
9.40.3.8 CreateComputePipeline(const ComputePipelineDescriptor &desc)=0 . . . 135
9.40.3.9 CreateGraphicsPipeline(const GraphicsPipelineDescriptor &desc)=0 . . . 135
9.40.3.10 CreateQuery(const QueryDescriptor &desc)=0 . . . 135

9.40.3.11 CreateRenderContext(const RenderContextDescriptor &desc, const std::shared_ptr< Window > &window=nullptr)=0 . . . 136
9.40.3.12 CreateRenderTarget(const RenderTargetDescriptor &desc)=0 . . . 136
9.40.3.13 CreateSampler(const SamplerDescriptor &desc)=0 . . . 136
9.40.3.14 CreateSamplerArray(unsigned int numSamplers, Sampler *const *samplerArray)=0 136
9.40.3.15 CreateShader(const ShaderType type)=0 . . . 137
9.40.3.16 CreateShaderProgram()=0 . . . 137
9.40.3.17 CreateTexture(const TextureDescriptor &textureDesc, const ImageDescriptor *imageDesc=nullptr)=0 . . . 137
9.40.3.18 CreateTextureArray(unsigned int numTextures, Texture *const *textureArray)=0 . . . 137
9.40.3.19 FindModules() . . . 138
9.40.3.20 GenerateMips(Texture &texture)=0 . . . 138
9.40.3.21 GetConfiguration() const . . . 138
9.40.3.22 GetDefaultTextureImageRGBAub(int numPixels) const . . . 138
9.40.3.23 GetName() const . . . 138
9.40.3.24 GetRendererInfo() const . . . 139
9.40.3.25 GetRenderingCaps() const . . . 139
9.40.3.26 Load(const std::string &moduleName, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr) . . . 139
9.40.3.27 MapBuffer(Buffer &buffer, const BufferCPUAccess access)=0 . . . 140
9.40.3.28 operator=(const RenderSystem &)=delete . . . 140
9.40.3.29 QueryTextureDescriptor(const Texture &texture)=0 . . . 140
9.40.3.30 ReadTexture(const Texture &texture, int mipLevel, ImageFormat imageFormat, DataType dataType, void *buffer)=0 . . . 140
9.40.3.31 Release(RenderContext &renderContext)=0 . . . 141
9.40.3.32 Release(CommandBuffer &commandBuffer)=0 . . . 141
9.40.3.33 Release(Buffer &buffer)=0 . . . 141
9.40.3.34 Release(BufferArray &bufferArray)=0 . . . 141
9.40.3.35 Release(Texture &texture)=0 . . . 141
9.40.3.36 Release(TextureArray &textureArray)=0 . . . 141
9.40.3.37 Release(Sampler &sampler)=0 . . . 141
9.40.3.38 Release(SamplerArray &samplerArray)=0 . . . 141

9.40.3.39 Release(RenderTarget &renderTarget)=0 . . . 142
9.40.3.40 Release(Shader &shader)=0 . . . 142
9.40.3.41 Release(ShaderProgram &shaderProgram)=0 . . . 142
9.40.3.42 Release(GraphicsPipeline &graphicsPipeline)=0 . . . 142
9.40.3.43 Release(ComputePipeline &computePipeline)=0 . . . 142
9.40.3.44 Release(Query &query)=0 . . . 142
9.40.3.45 SetConfiguration(const RenderSystemConfiguration &config) . . . 142
9.40.3.46 SetRendererInfo(const RendererInfo &info) . . . 142
9.40.3.47 SetRenderingCaps(const RenderingCaps &caps) . . . 143
9.40.3.48 UnmapBuffer(Buffer &buffer)=0 . . . 143
9.40.3.49 WriteBuffer(Buffer &buffer, const void *data, std::size_t dataSize, std::size_t offset)=0 . . . 143
9.40.3.50 WriteTexture(Texture &texture, const SubTextureDescriptor &subTextureDesc, const ImageDescriptor &imageDesc)=0 . . . 143
9.41 LLGL::RenderSystemConfiguration Struct Reference . . . 144
9.41.1 Detailed Description . . . 144
9.41.2 Member Data Documentation . . . 144
9.41.2.1 defaultImageColor . . . 144
9.41.2.2 threadCount . . . 144
9.42 LLGL::RenderTarget Class Reference . . . 145
9.42.1 Detailed Description . . . 145
9.42.2 Constructor & Destructor Documentation . . . 145
9.42.2.1 ~RenderTarget() . . . 145
9.42.3 Member Function Documentation . . . 145
9.42.3.1 ApplyMipResolution(Texture &texture, unsigned int mipLevel) . . . 145
9.42.3.2 ApplyResolution(const Gs::Vector2ui &resolution) . . . 145
9.42.3.3 AttachDepthBuffer(const Gs::Vector2ui &size)=0 . . . 145
9.42.3.4 AttachDepthStencilBuffer(const Gs::Vector2ui &size)=0 . . . 146
9.42.3.5 AttachStencilBuffer(const Gs::Vector2ui &size)=0 . . . 146
9.42.3.6 AttachTexture(Texture &texture, const RenderTargetAttachmentDescriptor &attachmentDesc)=0 . . . 146
9.42.3.7 DetachAll()=0 . . . 147

9.42.3.8 GetResolution() const . . . 147
9.42.3.9 ResetResolution() . . . 147
9.43 LLGL::RenderTargetAttachmentDescriptor Struct Reference . . . 147
9.43.1 Detailed Description . . . 147
9.43.2 Member Data Documentation . . . 148
9.43.2.1 cubeFace . . . 148
9.43.2.2 layer . . . 148
9.43.2.3 mipLevel . . . 148
9.44 LLGL::RenderTargetDescriptor Struct Reference . . . 148
9.44.1 Detailed Description . . . 149
9.44.2 Member Data Documentation . . . 149
9.44.2.1 customMultiSampling . . . 149
9.44.2.2 multiSampling . . . 149
9.45 LLGL::Sampler Class Reference . . . 149
9.45.1 Detailed Description . . . 149
9.45.2 Constructor & Destructor Documentation . . . 150
9.45.2.1 Sampler(const Sampler &)=delete . . . 150
9.45.2.2 ~Sampler() . . . 150
9.45.2.3 Sampler()=default . . . 150
9.45.3 Member Function Documentation . . . 150
9.45.3.1 operator=(const Sampler &)=delete . . . 150
9.46 LLGL::SamplerArray Class Reference . . . 150
9.46.1 Detailed Description . . . 150
9.46.2 Constructor & Destructor Documentation . . . 150
9.46.2.1 SamplerArray(const SamplerArray &)=delete . . . 150
9.46.2.2 ~SamplerArray() . . . 150
9.46.2.3 SamplerArray()=default . . . 150
9.46.3 Member Function Documentation . . . 150
9.46.3.1 operator=(const SamplerArray &)=delete . . . 150
9.47 LLGL::SamplerDescriptor Struct Reference . . . 151

9.47.1 Detailed Description . . . 151
9.47.2 Member Data Documentation . . . 151
9.47.2.1 borderColor . . . 151
9.47.2.2 compareOp . . . 152
9.47.2.3 depthCompare . . . 152
9.47.2.4 magFilter . . . 152
9.47.2.5 maxAnisotropy . . . 152
9.47.2.6 maxLOD . . . 152
9.47.2.7 minFilter . . . 152
9.47.2.8 minLOD . . . 152
9.47.2.9 mipMapFilter . . . 152
9.47.2.10 mipMapLODBias . . . 152
9.47.2.11 mipMapping . . . 152
9.47.2.12 textureWrapU . . . 153
9.47.2.13 textureWrapV . . . 153
9.47.2.14 textureWrapW . . . 153
9.48 LLGL::Scissor Struct Reference . . . 153
9.48.1 Detailed Description . . . 153
9.48.2 Constructor & Destructor Documentation . . . 154
9.48.2.1 Scissor()=default . . . 154
9.48.2.2 Scissor(const Scissor &)=default . . . 154
9.48.2.3 Scissor(int x, int y, int width, int height) . . . 154
9.48.3 Member Data Documentation . . . 154
9.48.3.1 height . . . 154
9.48.3.2 width . . . 154
9.48.3.3 x . . . 154
9.48.3.4 y . . . 154
9.49 LLGL::Shader Class Reference . . . 154
9.49.1 Detailed Description . . . 155
9.49.2 Constructor & Destructor Documentation . . . 155

9.49.2.1 Shader(const Shader &)=delete . . . 155
9.49.2.2 ~Shader() . . . 155
9.49.2.3 Shader(const ShaderType type) . . . 155
9.49.3 Member Function Documentation . . . 155
9.49.3.1 Compile(const ShaderSource &shaderSource)=0 . . . 155
9.49.3.2 Disassemble(int flags=0) . . . 155
9.49.3.3 GetType() const . . . 156
9.49.3.4 operator=(const Shader &)=delete . . . 156
9.49.3.5 QueryInfoLog()=0 . . . 156
9.50 LLGL::ShaderCompileFlags Struct Reference . . . 156
9.50.1 Detailed Description . . . 156
9.50.2 Member Enumeration Documentation . . . 156
9.50.2.1 anonymous enum . . . 156
9.51 LLGL::ShaderDisassembleFlags Struct Reference . . . 157
9.51.1 Detailed Description . . . 157
9.51.2 Member Enumeration Documentation . . . 157
9.51.2.1 anonymous enum . . . 157
9.52 LLGL::ShaderProgram Class Reference . . . 157
9.52.1 Detailed Description . . . 158
9.52.2 Constructor & Destructor Documentation . . . 159
9.52.2.1 ShaderProgram(const ShaderProgram &)=delete . . . 159
9.52.2.2 ~ShaderProgram() . . . 159
9.52.2.3 ShaderProgram()=default . . . 159
9.52.3 Member Function Documentation . . . 159
9.52.3.1 AttachShader(Shader &shader)=0 . . . 159
9.52.3.2 BindConstantBuffer(const std::string &name, unsigned int bindingIndex)=0 . . . 160
9.52.3.3 BindStorageBuffer(const std::string &name, unsigned int bindingIndex)=0 . . . 160
9.52.3.4 BuildInputLayout(const VertexFormat &vertexFormat)=0 . . . 161
9.52.3.5 DetachAll()=0 . . . 161
9.52.3.6 LinkShaders()=0 . . . 162

9.52.3.7 LockShaderUniform()=0 . . . 162
9.52.3.8 operator=(const ShaderProgram &)=delete . . . 163
9.52.3.9 QueryConstantBuffers() const =0 . . . 163
9.52.3.10 QueryInfoLog()=0 . . . 163
9.52.3.11 QueryStorageBuffers() const =0 . . . 163
9.52.3.12 QueryStreamOutputAttributes() const =0 . . . 163
9.52.3.13 QueryUniforms() const =0 . . . 163
9.52.3.14 QueryVertexAttributes() const =0 . . . 163
9.52.3.15 UnlockShaderUniform()=0 . . . 164
9.53 LLGL::ShaderSource Struct Reference . . . 164
9.53.1 Detailed Description . . . 165
9.53.2 Constructor & Destructor Documentation . . . 165
9.53.2.1 ShaderSource(const std::string &sourceCode) . . . 165
9.53.2.2 ShaderSource(std::string &&sourceCode) . . . 166
9.53.2.3 ShaderSource(const std::string &sourceCode, const std::string &entryPoint, const std::string &target, long flags=0) . . . 166
9.53.2.4 ShaderSource(std::string &&sourceCode, const std::string &entryPoint, const std::string &target, long flags=0) . . . 166
9.53.3 Member Data Documentation . . . 167
9.53.3.1 sourceCode . . . 167
9.53.3.2 sourceHLSL . . . 167
9.53.3.3 streamOutput . . . 167
9.54 LLGL::ShaderStageFlags Struct Reference . . . 167
9.54.1 Detailed Description . . . 168
9.54.2 Member Enumeration Documentation . . . 168
9.54.2.1 anonymous enum . . . 168
9.55 LLGL::ShaderUniform Class Reference . . . 169
9.55.1 Detailed Description . . . 170
9.55.2 Constructor & Destructor Documentation . . . 170
9.55.2.1 ~ShaderUniform() . . . 170
9.55.3 Member Function Documentation . . . 170

9.55.3.1 SetUniform(int location, const int value)=0 . . . 170
9.55.3.2 SetUniform(int location, const Gs::Vector2i &value)=0 . . . 170
9.55.3.3 SetUniform(int location, const Gs::Vector3i &value)=0 . . . 170
9.55.3.4 SetUniform(int location, const Gs::Vector4i &value)=0 . . . 170
9.55.3.5 SetUniform(int location, const float value)=0 . . . 170
9.55.3.6 SetUniform(int location, const Gs::Vector2f &value)=0 . . . 170
9.55.3.7 SetUniform(int location, const Gs::Vector3f &value)=0 . . . 170
9.55.3.8 SetUniform(int location, const Gs::Vector4f &value)=0 . . . 170
9.55.3.9 SetUniform(int location, const Gs::Matrix2f &value)=0 . . . 170
9.55.3.10 SetUniform(int location, const Gs::Matrix3f &value)=0 . . . 170
9.55.3.11 SetUniform(int location, const Gs::Matrix4f &value)=0 . . . 170
9.55.3.12 SetUniform(const std::string &name, const int value)=0 . . . 170
9.55.3.13 SetUniform(const std::string &name, const Gs::Vector2i &value)=0 . . . 170
9.55.3.14 SetUniform(const std::string &name, const Gs::Vector3i &value)=0 . . . 170
9.55.3.15 SetUniform(const std::string &name, const Gs::Vector4i &value)=0 . . . 170
9.55.3.16 SetUniform(const std::string &name, const float value)=0 . . . 171
9.55.3.17 SetUniform(const std::string &name, const Gs::Vector2f &value)=0 . . . 171
9.55.3.18 SetUniform(const std::string &name, const Gs::Vector3f &value)=0 . . . 171
9.55.3.19 SetUniform(const std::string &name, const Gs::Vector4f &value)=0 . . . 171
9.55.3.20 SetUniform(const std::string &name, const Gs::Matrix2f &value)=0 . . . 171
9.55.3.21 SetUniform(const std::string &name, const Gs::Matrix3f &value)=0 . . . 171
9.55.3.22 SetUniform(const std::string &name, const Gs::Matrix4f &value)=0 . . . 171
9.55.3.23 SetUniformArray(int location, const int *value, std::size_t count)=0 . . . 171
9.55.3.24 SetUniformArray(int location, const Gs::Vector2i *value, std::size_t count)=0 . . . 171
9.55.3.25 SetUniformArray(int location, const Gs::Vector3i *value, std::size_t count)=0 . . . 171
9.55.3.26 SetUniformArray(int location, const Gs::Vector4i *value, std::size_t count)=0 . . . 171
9.55.3.27 SetUniformArray(int location, const float *value, std::size_t count)=0 . . . 171
9.55.3.28 SetUniformArray(int location, const Gs::Vector2f *value, std::size_t count)=0 . . . 171
9.55.3.29 SetUniformArray(int location, const Gs::Vector3f *value, std::size_t count)=0 . . . 171
9.55.3.30 SetUniformArray(int location, const Gs::Vector4f *value, std::size_t count)=0 . . . 171

9.55.3.31 SetUniformArray(int location, const Gs::Matrix2f *value, std::size_t count)=0 . . . 171
9.55.3.32 SetUniformArray(int location, const Gs::Matrix3f *value, std::size_t count)=0 . . . 172
9.55.3.33 SetUniformArray(int location, const Gs::Matrix4f *value, std::size_t count)=0 . . . 172
9.55.3.34 SetUniformArray(const std::string &name, const int *value, std::size_t count)=0 . 172
9.55.3.35 SetUniformArray(const std::string &name, const Gs::Vector2i *value, std::size_t count)=0 . . . 172
9.55.3.36 SetUniformArray(const std::string &name, const Gs::Vector3i *value, std::size_t count)=0 . . . 172
9.55.3.37 SetUniformArray(const std::string &name, const Gs::Vector4i *value, std::size_t count)=0 . . . 172
9.55.3.38 SetUniformArray(const std::string &name, const float *value, std::size_t count)=0 172
9.55.3.39 SetUniformArray(const std::string &name, const Gs::Vector2f *value, std::size_t count)=0 . . . 172
9.55.3.40 SetUniformArray(const std::string &name, const Gs::Vector3f *value, std::size_t count)=0 . . . 172
9.55.3.41 SetUniformArray(const std::string &name, const Gs::Vector4f *value, std::size_t count)=0 . . . 172
9.55.3.42 SetUniformArray(const std::string &name, const Gs::Matrix2f *value, std::size_t count)=0 . . . 172
9.55.3.43 SetUniformArray(const std::string &name, const Gs::Matrix3f *value, std::size_t count)=0 . . . 172
9.55.3.44 SetUniformArray(const std::string &name, const Gs::Matrix4f *value, std::size_t count)=0 . . . 172
9.56 LLGL::ShaderSource::SourceHLSL Struct Reference . . . 172
9.56.1 Detailed Description . . . 173
9.56.2 Member Data Documentation . . . 173
9.56.2.1 entryPoint . . . 173
9.56.2.2 flags . . . 173
9.56.2.3 target . . . 173
9.57 LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor Struct Reference . . . 173
9.57.1 Member Data Documentation . . . 174
9.57.1.1 invertFrontFace . . . 174
9.57.1.2 lineWidth . . . 174
9.57.1.3 logicOp . . . 174

9.57.1.4 screenSpaceOriginLowerLeft . . . 174
9.58 LLGL::StencilDescriptor Struct Reference . . . 175
9.58.1 Detailed Description . . . 175
9.58.2 Member Data Documentation . . . 175
9.58.2.1 back . . . 175
9.58.2.2 front . . . 175
9.58.2.3 testEnabled . . . 175
9.59 LLGL::StencilFaceDescriptor Struct Reference . . . 176
9.59.1 Detailed Description . . . 176
9.59.2 Member Data Documentation . . . 176
9.59.2.1 compareOp . . . 176
9.59.2.2 depthFailOp . . . 176
9.59.2.3 depthPassOp . . . 176
9.59.2.4 readMask . . . 177
9.59.2.5 reference . . . 177
9.59.2.6 stencilFailOp . . . 177
9.59.2.7 writeMask . . . 177
9.60 LLGL::BufferDescriptor::StorageBufferDescriptor Struct Reference . . . 178
9.60.1 Member Data Documentation . . . 178
9.60.1.1 storageType . . . 178
9.60.1.2 stride . . . 178
9.60.1.3 vectorType . . . 178
9.61 LLGL::StorageBufferViewDescriptor Struct Reference . . . 179
9.61.1 Detailed Description . . . 179
9.61.2 Member Data Documentation . . . 179
9.61.2.1 index . . . 179
9.61.2.2 name . . . 179
9.61.2.3 type . . . 179
9.62 LLGL::ShaderSource::StreamOutput Struct Reference . . . 180
9.62.1 Detailed Description . . . 180

9.62.2 Member Data Documentation . . . 180
9.62.2.1 format . . . 180
9.63 LLGL::StreamOutputAttribute Struct Reference . . . 180
9.63.1 Detailed Description . . . 181
9.63.2 Constructor & Destructor Documentation . . . 181
9.63.2.1 StreamOutputAttribute()=default . . . 181
9.63.2.2 StreamOutputAttribute(const StreamOutputAttribute &)=default . . . 181
9.63.3 Member Function Documentation . . . 181
9.63.3.1 operator=(const StreamOutputAttribute &)=default . . . 181
9.63.4 Member Data Documentation . . . 181
9.63.4.1 components . . . 181
9.63.4.2 name . . . 181
9.63.4.3 outputSlot . . . 181
9.63.4.4 semanticIndex . . . 182
9.63.4.5 startComponent . . . 182
9.63.4.6 stream . . . 182
9.64 LLGL::StreamOutputFormat Struct Reference . . . 182
9.64.1 Detailed Description . . . 182
9.64.2 Member Function Documentation . . . 182
9.64.2.1 AppendAttribute(const StreamOutputAttribute &attrib) . . . 182
9.64.2.2 AppendAttributes(const StreamOutputFormat &format) . . . 183
9.64.3 Member Data Documentation . . . 183
9.64.3.1 attributes . . . 183
9.65 LLGL::SubTextureDescriptor Struct Reference . . . 183
9.65.1 Detailed Description . . . 184
9.65.2 Constructor & Destructor Documentation . . . 184
9.65.2.1 SubTextureDescriptor() . . . 184
9.65.2.2 ~SubTextureDescriptor() . . . 184
9.65.3 Member Data Documentation . . . 184
9.65.3.1 "@8 . . . 184

9.65.3.2 mipLevel . . . 184
9.65.3.3 texture1D . . . 184
9.65.3.4 texture2D . . . 184
9.65.3.5 texture3D . . . 185
9.65.3.6 textureCube . . . 185
9.66 LLGL::Texture Class Reference . . . 185
9.66.1 Detailed Description . . . 185
9.66.2 Constructor & Destructor Documentation . . . 185
9.66.2.1 Texture(const Texture &)=delete . . . 185
9.66.2.2 ~Texture() . . . 185
9.66.2.3 Texture(const TextureType type) . . . 185
9.66.3 Member Function Documentation . . . 185
9.66.3.1 GetType() const . . . 185
9.66.3.2 operator=(const Texture &)=delete . . . 185
9.66.3.3 QueryMipLevelSize(unsigned int mipLevel) const =0 . . . 185
9.67 LLGL::SubTextureDescriptor::Texture1DDescriptor Struct Reference . . . 186
9.67.1 Member Data Documentation . . . 186
9.67.1.1 layerOffset . . . 186
9.67.1.2 layers . . . 186
9.67.1.3 width . . . 186
9.67.1.4 x . . . 187
9.68 LLGL::TextureDescriptor::Texture1DDescriptor Struct Reference . . . 187
9.68.1 Member Data Documentation . . . 187
9.68.1.1 layers . . . 187
9.68.1.2 width . . . 187
9.69 LLGL::TextureDescriptor::Texture2DDescriptor Struct Reference . . . 187
9.69.1 Member Data Documentation . . . 188
9.69.1.1 height . . . 188
9.69.1.2 layers . . . 188
9.69.1.3 width . . . 188

9.70 LLGL::SubTextureDescriptor::Texture2DDescriptor Struct Reference . . . 188
9.70.1 Member Data Documentation . . . 188
9.70.1.1 height . . . 188
9.70.1.2 layerOffset . . . 189
9.70.1.3 layers . . . 189
9.70.1.4 width . . . 189
9.70.1.5 x . . . 189
9.70.1.6 y . . . 189
9.71 LLGL::TextureDescriptor::Texture2DMSDescriptor Struct Reference . . . 189
9.71.1 Member Data Documentation . . . 190
9.71.1.1 fixedSamples . . . 190
9.71.1.2 height . . . 190
9.71.1.3 layers . . . 190
9.71.1.4 samples . . . 190
9.71.1.5 width . . . 190
9.72 LLGL::TextureDescriptor::Texture3DDescriptor Struct Reference . . . 190
9.72.1 Member Data Documentation . . . 191
9.72.1.1 depth . . . 191
9.72.1.2 height . . . 191
9.72.1.3 width . . . 191
9.73 LLGL::SubTextureDescriptor::Texture3DDescriptor Struct Reference . . . 191
9.73.1 Member Data Documentation . . . 191
9.73.1.1 depth . . . 191
9.73.1.2 height . . . 192
9.73.1.3 width . . . 192
9.73.1.4 x . . . 192
9.73.1.5 y . . . 192
9.73.1.6 z . . . 192
9.74 LLGL::TextureArray Class Reference . . . 192
9.74.1 Detailed Description . . . 192

9.74.2 Constructor & Destructor Documentation . . . 193
9.74.2.1 TextureArray(const TextureArray &)=delete . . . 193
9.74.2.2 ~TextureArray() . . . 193
9.74.2.3 TextureArray()=default . . . 193
9.74.3 Member Function Documentation . . . 193
9.74.3.1 operator=(const TextureArray &)=delete . . . 193
9.75 LLGL::TextureDescriptor::TextureCubeDescriptor Struct Reference . . . 193
9.75.1 Member Data Documentation . . . 193
9.75.1.1 height . . . 193
9.75.1.2 layers . . . 193
9.75.1.3 width . . . 193
9.76 LLGL::SubTextureDescriptor::TextureCubeDescriptor Struct Reference . . . 194
9.76.1 Member Data Documentation . . . 194
9.76.1.1 cubeFaceOffset . . . 194
9.76.1.2 cubeFaces . . . 194
9.76.1.3 height . . . 194
9.76.1.4 layerOffset . . . 194
9.76.1.5 width . . . 194
9.76.1.6 x . . . 195
9.76.1.7 y . . . 195
9.77 LLGL::TextureDescriptor Struct Reference . . . 195
9.77.1 Detailed Description . . . 196
9.77.2 Constructor & Destructor Documentation . . . 196
9.77.2.1 TextureDescriptor() . . . 196
9.77.2.2 ~TextureDescriptor() . . . 196
9.77.3 Member Data Documentation . . . 196
9.77.3.1 "@6 . . . 196
9.77.3.2 format . . . 196
9.77.3.3 texture1D . . . 196
9.77.3.4 texture2D . . . 196

9.77.3.5 texture2DMS . . . 196
9.77.3.6 texture3D . . . 196
9.77.3.7 textureCube . . . 196
9.77.3.8 type . . . 197
9.78 LLGL::Timer Class Reference . . . 197
9.78.1 Member Typedef Documentation . . . 198
9.78.1.1 FrameCount . . . 198
9.78.2 Constructor & Destructor Documentation . . . 198
9.78.2.1 ~Timer() . . . 198
9.78.3 Member Function Documentation . . . 198
9.78.3.1 Create() . . . 198
9.78.3.2 GetDeltaTime() const . . . 198
9.78.3.3 GetFrameCount() const . . . 198
9.78.3.4 GetFrequency() const =0 . . . 198
9.78.3.5 MeasureTime() . . . 199
9.78.3.6 ResetFrameCounter() . . . 199
9.78.3.7 Start()=0 . . . 199
9.78.3.8 Stop()=0 . . . 199
9.79 LLGL::UniformDescriptor Struct Reference . . . 199
9.79.1 Detailed Description . . . 199
9.79.2 Member Data Documentation . . . 200
9.79.2.1 location . . . 200
9.79.2.2 name . . . 200
9.79.2.3 size . . . 200
9.79.2.4 type . . . 200
9.80 LLGL::VertexAttribute Struct Reference . . . 200
9.80.1 Detailed Description . . . 201
9.80.2 Constructor & Destructor Documentation . . . 201
9.80.2.1 VertexAttribute()=default . . . 201
9.80.2.2 VertexAttribute(const VertexAttribute &)=default . . . 201

9.80.2.3 VertexAttribute(const std::string &name, const VectorType vectorType, unsigned int instanceDivisor=0) . . . 201
9.80.2.4 VertexAttribute(const std::string &semanticName, unsigned int semanticIndex, const VectorType vectorType, unsigned int instanceDivisor=0) . . . 201
9.80.3 Member Function Documentation . . . 202
9.80.3.1 GetSize() const . . . 202
9.80.3.2 operator=(const VertexAttribute &)=default . . . 202
9.80.4 Member Data Documentation . . . 202
9.80.4.1 conversion . . . 202
9.80.4.2 inputSlot . . . 202
9.80.4.3 instanceDivisor . . . 202
9.80.4.4 name . . . 202
9.80.4.5 offset . . . 202
9.80.4.6 semanticIndex . . . 203
9.80.4.7 vectorType . . . 203
9.81 LLGL::BufferDescriptor::VertexBufferDescriptor Struct Reference . . . 203
9.81.1 Detailed Description . . . 203
9.81.2 Member Data Documentation . . . 203
9.81.2.1 format . . . 203
9.82 LLGL::VertexFormat Struct Reference . . . 204
9.82.1 Detailed Description . . . 204
9.82.2 Member Function Documentation . . . 204
9.82.2.1 AppendAttribute(const VertexAttribute &attrib, unsigned int offset=OffsetAppend) 204
9.82.2.2 AppendAttributes(const VertexFormat &format) . . . 205
9.82.3 Member Data Documentation . . . 205
9.82.3.1 attributes . . . 205
9.82.3.2 OffsetAppend . . . 205
9.82.3.3 stride . . . 206
9.83 LLGL::VideoAdapterDescriptor Struct Reference . . . 206
9.83.1 Detailed Description . . . 206
9.83.2 Member Data Documentation . . . 206

9.83.2.1 name . . . 206
9.83.2.2 outputs . . . 206
9.83.2.3 vendor . . . 207
9.83.2.4 videoMemory . . . 207
9.84 LLGL::VideoDisplayMode Struct Reference . . . 207
9.84.1 Detailed Description . . . 207
9.84.2 Member Data Documentation . . . 207
9.84.2.1 height . . . 207
9.84.2.2 refreshRate . . . 207
9.84.2.3 width . . . 208
9.85 LLGL::VideoModeDescriptor Struct Reference . . . 208
9.85.1 Detailed Description . . . 208
9.85.2 Member Data Documentation . . . 208
9.85.2.1 colorDepth . . . 208
9.85.2.2 fullscreen . . . 208
9.85.2.3 resolution . . . 208
9.85.2.4 swapChainMode . . . 209
9.86 LLGL::VideoOutput Struct Reference . . . 209
9.86.1 Detailed Description . . . 209
9.86.2 Member Data Documentation . . . 209
9.86.2.1 displayModes . . . 209
9.87 LLGL::Viewport Struct Reference . . . 209
9.87.1 Detailed Description . . . 210
9.87.2 Constructor & Destructor Documentation . . . 210
9.87.2.1 Viewport()=default . . . 210
9.87.2.2 Viewport(const Viewport &)=default . . . 210
9.87.2.3 Viewport(float x, float y, float width, float height) . . . 210
9.87.2.4 Viewport(float x, float y, float width, float height, float minDepth, float maxDepth) . . . 210
9.87.3 Member Data Documentation . . . 210
9.87.3.1 height . . . 210

9.87.3.2 maxDepth . . . 210
9.87.3.3 minDepth . . . 210
9.87.3.4 width . . . 211
9.87.3.5 x . . . 211
9.87.3.6 y . . . 211
9.88 LLGL::VsyncDescriptor Struct Reference . . . 211
9.88.1 Detailed Description . . . 211
9.88.2 Member Data Documentation . . . 211
9.88.2.1 enabled . . . 211
9.88.2.2 interval . . . 212
9.88.2.3 refreshRate . . . 212
9.89 LLGL::Window Class Reference . . . 212
9.89.1 Constructor & Destructor Documentation . . . 213
9.89.1.1 ∼Window() . . . 213
9.89.2 Member Function Documentation . . . 213
9.89.2.1 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 213
9.89.2.2 Create(const WindowDescriptor &desc) . . . 213
9.89.2.3 GetNativeHandle(void ∗nativeHandle) const =0 . . . 213
9.89.2.4 GetPosition() const =0 . . . 213
9.89.2.5 GetSize(bool useClientArea=true) const =0 . . . 213
9.89.2.6 GetTitle() const =0 . . . 213
9.89.2.7 IsShown() const =0 . . . 213
9.89.2.8 OnProcessEvents()=0 . . . 213
9.89.2.9 PostChar(wchar_t chr) . . . 214
9.89.2.10 PostDoubleClick(Key keyCode) . . . 214
9.89.2.11 PostGlobalMotion(const Point &motion) . . . 214
9.89.2.12 PostKeyDown(Key keyCode) . . . 214
9.89.2.13 PostKeyUp(Key keyCode) . . . 214
9.89.2.14 PostLocalMotion(const Point &position) . . . 214
9.89.2.15 PostQuit() . . . 214

9.89.2.16 PostResize(const Size &clientAreaSize) . . . 214
9.89.2.17 PostWheelMotion(int motion) . . . 214
9.89.2.18 ProcessEvents() . . . 214
9.89.2.19 QueryDesc() const =0 . . . 214
9.89.2.20 Recreate(const WindowDescriptor &desc)=0 . . . 214
9.89.2.21 RemoveEventListener(const EventListener *eventListener) . . . 215
9.89.2.22 SetDesc(const WindowDescriptor &desc)=0 . . . 215
9.89.2.23 SetPosition(const Point &position)=0 . . . 215
9.89.2.24 SetSize(const Size &size, bool useClientArea=true)=0 . . . 215
9.89.2.25 SetTitle(const std::wstring &title)=0 . . . 215
9.89.2.26 Show(bool show=true)=0 . . . 215
9.90 LLGL::WindowDescriptor Struct Reference . . . 215
9.90.1 Detailed Description . . . 215
9.90.2 Member Data Documentation . . . 216
9.90.2.1 acceptDropFiles . . . 216
9.90.2.2 borderless . . . 216
9.90.2.3 centered . . . 216
9.90.2.4 position . . . 216
9.90.2.5 preventForPowerSafe . . . 216
9.90.2.6 resizable . . . 216
9.90.2.7 size . . . 216
9.90.2.8 title . . . 216
9.90.2.9 visible . . . 216
9.90.2.10 windowContext . . . 216

10 File Documentation 217

10.1 Buffer.h File Reference . . . 217

10.2 BufferArray.h File Reference . . . 217

10.3 BufferFlags.h File Reference . . . 218

10.4 Color.h File Reference . . . 218

10.5 ColorRGB.h File Reference . . . 219

10.6 ColorRGBA.h File Reference . . . 220

10.7 CommandBuffer.h File Reference . . . 221

10.8 ComputePipeline.h File Reference . . . 221

10.9 Desktop.h File Reference . . . 222

10.10 Export.h File Reference . . . 222

10.10.1 Macro Definition Documentation . . . 222

10.10.1.1 LLGL_EXPORT . . . 222

10.11 Format.h File Reference . . . 222

10.12 GraphicsPipeline.h File Reference . . . 223

10.13 GraphicsPipelineFlags.h File Reference . . . 223

10.14 Image.h File Reference . . . 225

10.15 IndexFormat.h File Reference . . . 226

10.16 Input.h File Reference . . . 226

10.17 Key.h File Reference . . . 227

10.18 LinuxNativeHandle.h File Reference . . . 228

10.19 LLGL.h File Reference . . . 228

10.20 Log.h File Reference . . . 228

10.21 MacOSNativeHandle.h File Reference . . . 229

10.22 NativeHandle.h File Reference . . . 229

10.23 Query.h File Reference . . . 229

10.24 QueryFlags.h File Reference . . . 230

10.25 RenderContext.h File Reference . . . 230

10.26 RenderContextDescriptor.h File Reference . . . 231

10.27 RenderContextFlags.h File Reference . . . 232

10.28 RenderingDebugger.h File Reference . . . 232
10.29 RenderingProfiler.h File Reference . . . 233
10.30 RenderSystem.h File Reference . . . 234
10.31 RenderSystemFlags.h File Reference . . . 234
10.32 RenderTarget.h File Reference . . . 235
10.33 Sampler.h File Reference . . . 235
10.34 SamplerArray.h File Reference . . . 236
10.35 SamplerFlags.h File Reference . . . 236
10.36 Shader.h File Reference . . . 237
10.37 ShaderFlags.h File Reference . . . 237
10.38 ShaderProgram.h File Reference . . . 238
10.39 ShaderUniform.h File Reference . . . 238
10.40 StreamOutputAttribute.h File Reference . . . 239
10.41 StreamOutputFormat.h File Reference . . . 239
10.42 Texture.h File Reference . . . 240
10.43 TextureArray.h File Reference . . . 240
10.44 TextureFlags.h File Reference . . . 240
10.45 Timer.h File Reference . . . 242
10.46 Types.h File Reference . . . 242
10.47 Utility.h File Reference . . . 243
10.48 Version.h File Reference . . . 244
10.49 VertexAttribute.h File Reference . . . 244
10.50 VertexFormat.h File Reference . . . 245
10.51 VideoAdapter.h File Reference . . . 245
10.52 Win32NativeHandle.h File Reference . . . 245
10.53 Window.h File Reference . . . 246

Index **247**

Chapter 1

LLGL 0.01 Beta Documentation

LLGL (Low Level Graphics Library)

Overview

- **Version**: 0.01 Beta
- **License**: 3-Clause BSD License

Progress

- **OpenGL Renderer**: ~85% done
- **Direct3D 11 Renderer**: ~85% done
- **Direct3D 12 Renderer**: ~5% done
- **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:
CommandBuffer::DrawIndexed(unsigned int numVertices, unsigned int firstIndex);

// OpenGL Implementation:
void GLCommandBuffer::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 D3D11CommandBuffer::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    context_->DrawIndexed(numVertices, 0, firstIndex);
}

// Direct3D 12 Implementation
void D3D12CommandBuffer::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    commandList_->DrawIndexedInstanced(numVertices, 1, firstIndex, 0, 0);
}

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Global utility functions, especially to fill descriptor structures. 15

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

LLGL 19
LLGL::Desktop 52
LLGL::Log 53
LLGL::Version 53

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

LLGL::BlendDescriptor . . . 55
LLGL::BlendTargetDescriptor . . . 56
LLGL::Buffer . . . 57
LLGL::BufferArray . . . 58
LLGL::BufferDescriptor . . . 59
LLGL::BufferFlags . . . 61
LLGL::ClearFlags . . . 62
LLGL::Color< T, N > . . . 63
LLGL::Color< bool > . . . 63
LLGL::Color< float > . . . 63
LLGL::Color< T, 3u > . . . 66
LLGL::Color< T, 4u > . . . 68
LLGL::Color< unsigned char > . . . 63
LLGL::CommandBuffer . . . 72
LLGL::ComputePipeline . . . 86
LLGL::ComputePipelineDescriptor . . . 87
LLGL::ConstantBufferViewDescriptor . . . 88
LLGL::RenderingProfiler::Counter . . . 88
LLGL::DepthDescriptor . . . 90
LLGL::Window::EventListener . . . 91
    LLGL::Input . . . 99
LLGL::GraphicsAPIDependentStateDescriptor . . . 93
LLGL::GraphicsPipeline . . . 94
LLGL::GraphicsPipelineDescriptor . . . 94
LLGL::ImageDescriptor . . . 96
LLGL::BufferDescriptor::IndexBufferDescriptor . . . 98
LLGL::IndexFormat . . . 98
LLGL::RenderingDebugger::Message . . . 101
LLGL::MultiSamplingDescriptor . . . 103
LLGL::NativeContextHandle . . . 104
LLGL::NativeHandle . . . 105
LLGL::ProfileOpenGLDescriptor . . . 105
LLGL::Query . . . 107
LLGL::QueryDescriptor . . . 108
LLGL::RasterizerDescriptor . . . 109

LLGL::RenderContext . . . 110
LLGL::RenderContextDescriptor . . . 113
LLGL::RendererID . . . 114
LLGL::RendererInfo . . . 116
LLGL::RenderingCaps . . . 118
LLGL::RenderingDebugger . . . 124
LLGL::RenderingProfiler . . . 126
LLGL::RenderSystem . . . 130
LLGL::RenderSystemConfiguration . . . 144
LLGL::RenderTarget . . . 145
LLGL::RenderTargetAttachmentDescriptor . . . 147
LLGL::RenderTargetDescriptor . . . 148
LLGL::Sampler . . . 149
LLGL::SamplerArray . . . 150
LLGL::SamplerDescriptor . . . 151
LLGL::Scissor . . . 153
LLGL::Shader . . . 154
LLGL::ShaderCompileFlags . . . 156
LLGL::ShaderDisassembleFlags . . . 157
LLGL::ShaderProgram . . . 157
LLGL::ShaderSource . . . 164
LLGL::ShaderStageFlags . . . 167
LLGL::ShaderUniform . . . 169
LLGL::ShaderSource::SourceHLSL . . . 172
LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor . . . 173
LLGL::StencilDescriptor . . . 175
LLGL::StencilFaceDescriptor . . . 176
LLGL::BufferDescriptor::StorageBufferDescriptor . . . 178
LLGL::StorageBufferViewDescriptor . . . 179
LLGL::ShaderSource::StreamOutput . . . 180
LLGL::StreamOutputAttribute . . . 180
LLGL::StreamOutputFormat . . . 182
LLGL::SubTextureDescriptor . . . 183
LLGL::Texture . . . 185
LLGL::SubTextureDescriptor::Texture1DDescriptor . . . 186
LLGL::TextureDescriptor::Texture1DDescriptor . . . 187
LLGL::TextureDescriptor::Texture2DDescriptor . . . 187
LLGL::SubTextureDescriptor::Texture2DDescriptor . . . 188
LLGL::TextureDescriptor::Texture2DMSDescriptor . . . 189
LLGL::TextureDescriptor::Texture3DDescriptor . . . 190
LLGL::SubTextureDescriptor::Texture3DDescriptor . . . 191
LLGL::TextureArray . . . 192
LLGL::TextureDescriptor::TextureCubeDescriptor . . . 193
LLGL::SubTextureDescriptor::TextureCubeDescriptor . . . 194
LLGL::TextureDescriptor . . . 195
LLGL::Timer . . . 197
LLGL::UniformDescriptor . . . 199
LLGL::VertexAttribute . . . 200
LLGL::BufferDescriptor::VertexBufferDescriptor . . . 203
LLGL::VertexFormat . . . 204
LLGL::VideoAdapterDescriptor . . . 206
LLGL::VideoDisplayMode . . . 207
LLGL::VideoModeDescriptor . . . 208
LLGL::VideoOutput . . . 209
LLGL::Viewport . . . 209
LLGL::VsyncDescriptor . . . 211
LLGL::Window . . . 212
LLGL::WindowDescriptor . . . 215

Chapter 5

Class Index

5.1 Class List

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

LLGL::BlendDescriptor
Blending state descriptor structure . . . 55
LLGL::BlendTargetDescriptor
Blend target state descriptor structure . . . 56
LLGL::Buffer
Hardware buffer interface . . . 57
LLGL::BufferArray
Array of hardware buffers interface . . . 58
LLGL::BufferDescriptor
Hardware buffer descriptor structure . . . 59
LLGL::BufferFlags
Buffer flags enumeration . . . 61
LLGL::ClearFlags
Command buffer clear flags . . . 62
LLGL::Color< T, N >
Base color class with N components . . . 63
LLGL::Color< T, 3u >
RGB color class with components: r, g, and b . . . 66
LLGL::Color< T, 4u >
RGBA color class with components: r, g, b, and a . . . 68
LLGL::CommandBuffer
Command buffer interface . . . 72
LLGL::ComputePipeline
Compute pipeline interface . . . 86
LLGL::ComputePipelineDescriptor
Compute pipeline descriptor structure . . . 87
LLGL::ConstantBufferViewDescriptor
Constant buffer shader-view descriptor structure . . . 88
LLGL::RenderingProfiler::Counter
Profiling counter class . . . 88
LLGL::DepthDescriptor
Depth state descriptor structure . . . 90
LLGL::Window::EventListener . . . 91
LLGL::GraphicsAPIDependentStateDescriptor
Low-level graphics API dependent state descriptor union . . . 93

LLGL::GraphicsPipeline
Graphics pipeline interface . . . 94
LLGL::GraphicsPipelineDescriptor
Graphics pipeline descriptor structure . . . 94
LLGL::ImageDescriptor
Image descriptor structure . . . 96
LLGL::BufferDescriptor::IndexBufferDescriptor . . . 98
LLGL::IndexFormat . . . 98
LLGL::Input . . . 99
LLGL::RenderingDebugger::Message
Rendering debugger message class . . . 101
LLGL::MultiSamplingDescriptor
Multi-sampling descriptor structure . . . 103
LLGL::NativeContextHandle
Linux native context handle structure . . . 104
LLGL::NativeHandle
Linux native handle structure . . . 105
LLGL::ProfileOpenGLDescriptor
OpenGL profile descriptor structure . . . 105
LLGL::Query
Query interface . . . 107
LLGL::QueryDescriptor
Query descriptor structure . . . 108
LLGL::RasterizerDescriptor
Rasterizer state descriptor structure . . . 109
LLGL::RenderContext
Render context interface . . . 110
LLGL::RenderContextDescriptor
Render context descriptor structure . . . 113
LLGL::RendererID
Renderer identification number enumeration . . . 114
LLGL::RendererInfo
Renderer basic information structure . . . 116
LLGL::RenderingCaps
Rendering capabilities structure . . . 118
LLGL::RenderingDebugger
Rendering debugger interface . . . 124
LLGL::RenderingProfiler
Rendering profiler model class . . . 126
LLGL::RenderSystem
Render system interface . . . 130
LLGL::RenderSystemConfiguration
Render system configuration structure . . . 144
LLGL::RenderTarget
Render target interface . . . 145
LLGL::RenderTargetAttachmentDescriptor
Render target attachment descriptor structure . . . 147
LLGL::RenderTargetDescriptor
Render target descriptor structure . . . 148
LLGL::Sampler
Sampler interface . . . 149
LLGL::SamplerArray
Sampler array interface . . . 150
LLGL::SamplerDescriptor
Texture sampler descriptor structure . . . 151
LLGL::Scissor
Scissor dimensions . . . 153

LLGL::Shader
Shader interface . . . 154
LLGL::ShaderCompileFlags
Shader compilation flags enumeration . . . 156
LLGL::ShaderDisassembleFlags
Shader disassemble flags enumeration . . . 157
LLGL::ShaderProgram
Shader program interface . . . 157
LLGL::ShaderSource
Shader source code structure . . . 164
LLGL::ShaderStageFlags
Shader stage flags . . . 167
LLGL::ShaderUniform
Shader uniform setter interface . . . 169
LLGL::ShaderSource::SourceHLSL
Additional descripor for HLSL shader source . . . 172
LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor . . . 173
LLGL::StencilDescriptor
Stencil state descriptor structure . . . 175
LLGL::StencilFaceDescriptor
Stencil face descriptor structure . . . 176
LLGL::BufferDescriptor::StorageBufferDescriptor . . . 178
LLGL::StorageBufferViewDescriptor
Storage buffer shader-view descriptor structure . . . 179
LLGL::ShaderSource::StreamOutput
Additional descriptor for stream outputs . . . 180
LLGL::StreamOutputAttribute
Stream-output attribute structure . . . 180
LLGL::StreamOutputFormat
Stream-output format descriptor structure . . . 182
LLGL::SubTextureDescriptor
Sub-texture descriptor structure . . . 183
LLGL::Texture
Texture interface . . . 185
LLGL::SubTextureDescriptor::Texture1DDescriptor . . . 186
LLGL::TextureDescriptor::Texture1DDescriptor . . . 187
LLGL::TextureDescriptor::Texture2DDescriptor . . . 187
LLGL::SubTextureDescriptor::Texture2DDescriptor . . . 188
LLGL::TextureDescriptor::Texture2DMSDescriptor . . . 189
LLGL::TextureDescriptor::Texture3DDescriptor . . . 190
LLGL::SubTextureDescriptor::Texture3DDescriptor . . . 191
LLGL::TextureArray
Array of textures interface . . . 192
LLGL::TextureDescriptor::TextureCubeDescriptor . . . 193
LLGL::SubTextureDescriptor::TextureCubeDescriptor . . . 194
LLGL::TextureDescriptor
Texture descriptor structure . . . 195
LLGL::Timer . . . 197
LLGL::UniformDescriptor
Shader uniform descriptor structure . . . 199
LLGL::VertexAttribute
Vertex attribute structure . . . 200
LLGL::BufferDescriptor::VertexBufferDescriptor
Vertex buffer descriptor structure . . . 203
LLGL::VertexFormat
Vertex format descriptor structure . . . 204
LLGL::VideoAdapterDescriptor
Video adapter descriptor structure . . . 206

LLGL::VideoDisplayMode
Video display mode structure 207
LLGL::VideoModeDescriptor
Video mode descriptor structure 208
LLGL::VideoOutput
Video output structure 209
LLGL::Viewport
Viewport dimensions 209
LLGL::VsyncDescriptor
Vertical-synchronization (Vsync) descriptor structure 211
LLGL::Window 212
LLGL::WindowDescriptor
Window descriptor structure 215

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Buffer.h . . . 217
BufferArray.h . . . 217
BufferFlags.h . . . 218
Color.h . . . 218
ColorRGB.h . . . 219
ColorRGBA.h . . . 220
CommandBuffer.h . . . 221
ComputePipeline.h . . . 221
Desktop.h . . . 222
Export.h . . . 222
Format.h . . . 222
GraphicsPipeline.h . . . 223
GraphicsPipelineFlags.h . . . 223
Image.h . . . 225
IndexFormat.h . . . 226
Input.h . . . 226
Key.h . . . 227
LinuxNativeHandle.h . . . 228
LLGL.h . . . 228
Log.h . . . 228
MacOSNativeHandle.h . . . 229
NativeHandle.h . . . 229
Query.h . . . 229
QueryFlags.h . . . 230
RenderContext.h . . . 230
RenderContextDescriptor.h . . . 231
RenderContextFlags.h . . . 232
RenderingDebugger.h . . . 232
RenderingProfiler.h . . . 233
RenderSystem.h . . . 234
RenderSystemFlags.h . . . 234
RenderTarget.h . . . 235
Sampler.h . . . 235
SamplerArray.h . . . 236
SamplerFlags.h . . . 236

Shader.h . . . 237
ShaderFlags.h . . . 237
ShaderProgram.h . . . 238
ShaderUniform.h . . . 238
StreamOutputAttribute.h . . . 239
StreamOutputFormat.h . . . 239
Texture.h . . . 240
TextureArray.h . . . 240
TextureFlags.h . . . 240
Timer.h . . . 242
Types.h . . . 242
Utility.h . . . 243
Version.h . . . 244
VertexAttribute.h . . . 244
VertexFormat.h . . . 245
VideoAdapter.h . . . 245
Win32NativeHandle.h . . . 245
Window.h . . . 246

Chapter 7

Module Documentation

7.1 Global utility functions, especially to fill descriptor structures.

Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (TextureFormat format, unsigned int width)

  Returns a TextureDescriptor structure with the TextureType::Texture1D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::Texture2D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int depth)

  Returns a TextureDescriptor structure with the TextureType::Texture3D type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::TextureCube type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (TextureFormat format, unsigned int width, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.
- LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (unsigned int size, const VertexFormat &vertex↩Format, long flags=0)

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (unsigned int size, const IndexFormat &index↩Format, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.

- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (unsigned int size, long flags=BufferFlags::↩ DynamicUsage)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (unsigned int size, const StorageBufferType storageType, unsigned int stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  Returns a BufferDescriptor structure for a storage buffer.

7.1.1 Detailed Description

7.1.2 Function Documentation

7.1.2.1 LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (unsigned int *size*, long *flags* = BufferFlags::DynamicUsage)

Returns a BufferDescriptor structure for a constant buffer.

7.1.2.2 LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (unsigned int *size*, const IndexFormat & *indexFormat*, long *flags* = 0)

Returns a BufferDescriptor structure for an index buffer.

7.1.2.3 LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (unsigned int *size*, const StorageBufferType *storageType*, unsigned int *stride*, long *flags* = BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

Returns a BufferDescriptor structure for a storage buffer.

7.1.2.4 LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (TextureFormat *format*, unsigned int *width*, unsigned int *layers*)

Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.

7.1.2.5 LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (TextureFormat *format*, unsigned int *width*)

Returns a TextureDescriptor structure with the TextureType::Texture1D type.

7.1.2.6 LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*, unsigned int *layers*)

Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.

7.1.2.7 LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*)

Returns a TextureDescriptor structure with the TextureType::Texture2D type.

7.1.2.8 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*, unsigned int *layers*, unsigned int *samples*, bool *fixedSamples* = `true`)

Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.

7.1.2.9 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*, unsigned int *samples*, bool *fixedSamples* = `true`)

Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.

7.1.2.10 LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*, unsigned int *depth*)

Returns a TextureDescriptor structure with the TextureType::Texture3D type.

7.1.2.11 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*, unsigned int *layers*)

Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.

7.1.2.12 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (TextureFormat *format*, unsigned int *width*, unsigned int *height*)

Returns a TextureDescriptor structure with the TextureType::TextureCube type.

7.1.2.13 LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (unsigned int *size*, const VertexFormat & *vertexFormat*, long *flags* = `0`)

Returns a BufferDescriptor structure for a vertex buffer.

Chapter 8

Namespace Documentation

8.1 LLGL Namespace Reference

Namespaces

- Desktop
- Log
- Version

Classes

- struct BlendDescriptor

  Blending state descriptor structure.
- struct BlendTargetDescriptor

  Blend target state descriptor structure.
- class Buffer

  Hardware buffer interface.
- class BufferArray

  Array of hardware buffers interface.
- struct BufferDescriptor

  Hardware buffer descriptor structure.
- struct BufferFlags

  Buffer flags enumeration.
- struct ClearFlags

  Command buffer clear 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 CommandBuffer

  Command buffer interface.
- class ComputePipeline

  Compute pipeline interface.

- struct ComputePipelineDescriptor

  Compute pipeline 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 IndexFormat
- class Input
- struct MultiSamplingDescriptor

  Multi-sampling descriptor structure.
- struct NativeContextHandle

  Linux native context handle structure.
- struct NativeHandle

  Linux native handle structure.
- struct ProfileOpenGLDescriptor

  OpenGL profile descriptor structure.
- class Query

  Query interface.
- struct QueryDescriptor

  Query descriptor structure.
- struct RasterizerDescriptor

  Rasterizer state descriptor structure.
- class RenderContext

  Render context interface.
- struct RenderContextDescriptor

  Render context descriptor structure.
- struct RendererID

  Renderer identification number enumeration.
- struct RendererInfo

  Renderer basic information structure.
- 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.
- struct RenderTargetAttachmentDescriptor

Render target attachment descriptor structure.

- struct RenderTargetDescriptor

  Render target descriptor structure.
- class Sampler

  Sampler interface.
- class SamplerArray

  Sampler array 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.
- struct ShaderSource

  Shader source code structure.
- struct ShaderStageFlags

  Shader stage flags.
- class ShaderUniform

  Shader uniform setter interface.
- struct StencilDescriptor

  Stencil state descriptor structure.
- struct StencilFaceDescriptor

  Stencil face descriptor structure.
- struct StorageBufferViewDescriptor

  Storage buffer shader-view descriptor structure.
- struct StreamOutputAttribute

  Stream-output attribute structure.
- struct StreamOutputFormat

  Stream-output format descriptor structure.
- struct SubTextureDescriptor

  Sub-texture descriptor structure.
- class Texture

  Texture interface.
- class TextureArray

  Array of textures interface.
- struct TextureDescriptor

  Texture descriptor structure.
- class Timer
- struct UniformDescriptor

  Shader uniform descriptor structure.
- struct VertexAttribute

  Vertex attribute structure.
- struct VertexFormat

  Vertex format descriptor structure.
- struct VideoAdapterDescriptor

Video adapter descriptor structure.

- struct VideoDisplayMode

  Video display mode structure.
- struct VideoModeDescriptor

  Video mode descriptor structure.
- struct VideoOutput

  Video output structure.
- struct Viewport

  Viewport dimensions.
- struct VsyncDescriptor

  Vertical-synchronization (Vsync) descriptor structure.
- 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 BufferType {
  BufferType::Vertex, BufferType::Index, BufferType::Constant, BufferType::Storage,
  BufferType::StreamOutput }

  Hardware buffer 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 BufferCPUAccess { BufferCPUAccess::ReadOnly, BufferCPUAccess::WriteOnly, BufferCPUAccess::ReadWrite }

  Hardware buffer CPU acccess enumeration.
- enum DataType {
  DataType::Int8, DataType::UInt8, DataType::Int16, DataType::UInt16,
  DataType::Int32, DataType::UInt32, DataType::Float, DataType::Double }

  Renderer data types enumeration.
- enum VectorType {
  VectorType::Float, VectorType::Float2, VectorType::Float3, VectorType::Float4,
  VectorType::Double, VectorType::Double2, VectorType::Double3, VectorType::Double4,
  VectorType::Int, VectorType::Int2, VectorType::Int3, VectorType::Int4,
  VectorType::UInt, VectorType::UInt2, VectorType::UInt3, VectorType::UInt4 }

  Renderer vector types enumeration.
- enum PrimitiveType { PrimitiveType::Points, PrimitiveType::Lines, PrimitiveType::Triangles }

  Primitive type enumeration.
- enum PrimitiveTopology {
  PrimitiveTopology::PointList, PrimitiveTopology::LineList, PrimitiveTopology::LineStrip, PrimitiveTopology::LineLoop,
  PrimitiveTopology::LineListAdjacency, PrimitiveTopology::LineStripAdjacency, PrimitiveTopology::TriangleList, PrimitiveTopology::TriangleStrip,
  PrimitiveTopology::TriangleFan, PrimitiveTopology::TriangleListAdjacency, PrimitiveTopology::TriangleStripAdjacency, PrimitiveTopology::Patches1,
  PrimitiveTopology::Patches2, PrimitiveTopology::Patches3, PrimitiveTopology::Patches4, PrimitiveTopology::Patches5,
  PrimitiveTopology::Patches6, PrimitiveTopology::Patches7, PrimitiveTopology::Patches8, PrimitiveTopology::Patches9,
  PrimitiveTopology::Patches10, PrimitiveTopology::Patches11, PrimitiveTopology::Patches12, PrimitiveTopology::Patches13,
  PrimitiveTopology::Patches14, PrimitiveTopology::Patches15, PrimitiveTopology::Patches16, PrimitiveTopology::Patches17,
  PrimitiveTopology::Patches18, PrimitiveTopology::Patches19, PrimitiveTopology::Patches20, PrimitiveTopology::Patches21,
  PrimitiveTopology::Patches22, PrimitiveTopology::Patches23, PrimitiveTopology::Patches24, PrimitiveTopology::Patches25,
  PrimitiveTopology::Patches26, PrimitiveTopology::Patches27, PrimitiveTopology::Patches28, PrimitiveTopology::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, BlendOp::SrcAlphaSaturate, BlendOp::BlendFactor,
  BlendOp::InvBlendFactor, BlendOp::Src1Color, BlendOp::InvSrc1Color, BlendOp::Src1Alpha,
  BlendOp::InvSrc1Alpha }

  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 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::BrowserFavorits, 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::ComputeShaderInvocations, 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 RenderConditionMode { RenderConditionMode::Wait, RenderConditionMode::NoWait, RenderConditionMode::ByRegionWait, RenderConditionMode::ByRegionNoWait, RenderConditionMode::WaitInverted, RenderConditionMode::NoWaitInverted, RenderConditionMode::ByRegionWaitInverted, RenderConditionMode::ByRegionNoWaitInverted }

  Render condition mode enumeration.
- enum LogicOp { LogicOp::Keep, LogicOp::Disabled, LogicOp::Clear, LogicOp::Set, LogicOp::Copy, LogicOp::InvertedCopy, LogicOp::Noop, LogicOp::Invert, LogicOp::AND, LogicOp::NAND, LogicOp::OR, LogicOp::NOR, LogicOp::XOR, LogicOp::Equiv, LogicOp::ReverseAND, LogicOp::InvertedAND, LogicOp::ReverseOR, LogicOp::InvertedOR }

  Logical pixel operation enumeration.
- enum ErrorType { ErrorType::InvalidArgument, ErrorType::InvalidState, ErrorType::UnsupportedFeature }

  Rendering debugger error types enumeration.
- enum WarningType { WarningType::ImproperArgument, WarningType::ImproperState, WarningType::PointlessOperation }
- enum ShadingLanguage { ShadingLanguage::Unsupported = 0, ShadingLanguage::GLSL_110 = 110, ShadingLanguage::GLSL_120 = 120, ShadingLanguage::GLSL_130 = 130, ShadingLanguage::GLSL_140 = 140, ShadingLanguage::GLSL_150 = 150, ShadingLanguage::GLSL_330 = 330, ShadingLanguage::GLSL_400 = 400, ShadingLanguage::GLSL_410 = 410, ShadingLanguage::GLSL_420 = 420, ShadingLanguage::GLSL_430 = 430, ShadingLanguage::GLSL_440 = 440, ShadingLanguage::GLSL_450 = 450, ShadingLanguage::HLSL_2_0 = 100200, ShadingLanguage::HLSL_2_0a = 100201, ShadingLanguage::HLSL_2_0b = 100202, ShadingLanguage::HLSL_3_0 = 100300, ShadingLanguage::HLSL_4_0 = 100400, ShadingLanguage::HLSL_4_1 = 100410, ShadingLanguage::HLSL_5_0 = 100500 }

  Shading language version enumation.
- 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 TextureType {
  TextureType::Texture1D, TextureType::Texture2D, TextureType::Texture3D, TextureType::TextureCube,
  TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, TextureType↩
  ::Texture2DMS,
  TextureType::Texture2DMSArray }

  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 unsigned int DataTypeSize (const DataType dataType)

  Returns the size (in bytes) of the specified data type.
- LLGL_EXPORT unsigned int VectorTypeSize (const VectorType vectorType)

  Returns the size (in bytes) of the specified vector type.
- LLGL_EXPORT void VectorTypeFormat (const VectorType vectorType, DataType &dataType, unsigned int &components)

  Retrieves the format of the specified vector type.
- LLGL_EXPORT unsigned int 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 foramt 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 bool operator== (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)
- LLGL_EXPORT bool operator!= (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)
- LLGL_EXPORT unsigned int NumMipLevels (unsigned int width, unsigned int height=1, unsigned int depth=1)

  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 IsArrayTexture (const TextureType type)

  Returns true if the specified texture type is an array texture.
- LLGL_EXPORT bool IsMultiSampleTexture (const TextureType type)

  Returns true if the specified texture type is a multi-sample texture.
- LLGL_EXPORT TextureDescriptor Texture1DDesc (TextureFormat format, unsigned int width)

  Returns a TextureDescriptor structure with the TextureType::Texture1D type.

- LLGL_EXPORT TextureDescriptor Texture2DDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::Texture2D type.
- LLGL_EXPORT TextureDescriptor Texture3DDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int depth)

  Returns a TextureDescriptor structure with the TextureType::Texture3D type.
- LLGL_EXPORT TextureDescriptor TextureCubeDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::TextureCube type.
- LLGL_EXPORT TextureDescriptor Texture1DArrayDesc (TextureFormat format, unsigned int width, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.
- LLGL_EXPORT TextureDescriptor Texture2DArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.
- LLGL_EXPORT TextureDescriptor TextureCubeArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.
- LLGL_EXPORT TextureDescriptor Texture2DMSDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.
- LLGL_EXPORT TextureDescriptor Texture2DMSArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.
- LLGL_EXPORT BufferDescriptor VertexBufferDesc (unsigned int size, const VertexFormat &vertexFormat, long flags=0)

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor IndexBufferDesc (unsigned int size, const IndexFormat &indexFormat, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor ConstantBufferDesc (unsigned int size, long flags=BufferFlags::Dynamic↩Usage)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor StorageBufferDesc (unsigned int size, const StorageBufferType storage↩Type, unsigned int stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  Returns a BufferDescriptor structure for a storage buffer.
- 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.

8.1.1 Typedef Documentation

8.1.1.1 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

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

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

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

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

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

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

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

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

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

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

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

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

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

Debug callback function interface.

Parameters

in	*type*	Descriptive type of the message.
in	*message*	Specifies the debug output message.

Remarks

This output is renderer dependent.

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

2D point (integer)

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

2D size (integer)

8.1.2 Enumeration Type Documentation

8.1.2.1 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.

8.1.2.2 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.

8.1.2.3 enum LLGL::BlendOp [strong]

Blending operations enumeration.

Enumerator

Zero Data source is the color black (0, 0, 0, 0).

One Data source is the color white (1, 1, 1, 1).

SrcColor Data source is color data (RGB) from a fragment shader.

InvSrcColor Data source is inverted color data (1 - RGB) from a fragment shader.

SrcAlpha Data source is alpha data (A) from a fragment shader.

InvSrcAlpha Data source is inverted alpha data (1 - A) from a fragment shader.

DestColor Data source is color data (RGB) from a framebuffer.

InvDestColor Data source is inverted color data (1 - RGB) from a framebuffer.

DestAlpha Data source is alpha data (A) from a framebuffer.

InvDestAlpha Data source is inverted alpha data (1 - A) from a framebuffer.

SrcAlphaSaturate Data source is alpha data (A) from a fragment shader which is clamped to 1 or less.

BlendFactor Data source is the blend factor (RGBA) from the blend state.

See also

BlendDescriptor::blendFactor

InvBlendFactor Data source is the inverted blend factor (1 - RGBA) from the blend state.

See also

BlendDescriptor::blendFactor

Src1Color Data sources are both color data (RGB) from a fragment shader with dual-source color blending.

InvSrc1Color Data sources are both inverted color data (1 - RGB) from a fragment shader with dual-source color blending.

Src1Alpha Data sources are both alpha data (A) from a fragment shader with dual-source color blending.

InvSrc1Alpha Data sources are both inverted alpha data (1 - A) from a fragment shader with dual-source color blending.

8.1.2.4 enum LLGL::BufferCPUAccess [strong]

Hardware buffer CPU acccess enumeration.

See also

RenderSystem::MapBuffer

Enumerator

ReadOnly CPU read access only.

WriteOnly CPU write access only.

ReadWrite CPU read and write access.

8.1.2.5 enum LLGL::BufferType [strong]

Hardware buffer type enumeration.

Enumerator

Vertex Vertex buffer type.

Index Index buffer type.

Constant Constant buffer type (also called "Uniform Buffer Object").

Storage Storage buffer type (also called "Shader Storage Buffer Object" or "Read/Write Buffer").

StreamOutput Stream output buffer type (also called "Transform Feedback Buffer").

8.1.2.6 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).

8.1.2.7 enum LLGL::CompareOp [strong]

Compare operations enumeration.

Remarks

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

Enumerator

Never Comparison never passes.

Less Comparison passes if the source data is less than the destination data.

Equal Comparison passes if the source data is euqal to the right-hand-side.

LessEqual Comparison passes if the source data is less than or equal to the right-hand-side.

Greater Comparison passes if the source data is greater than the right-hand-side.

NotEqual Comparison passes if the source data is not equal to the right-hand-side.

GreaterEqual Comparison passes if the source data is greater than or equal to the right-hand-side.

Ever Comparison always passes. (Can not be called "Always" due to conflict with X11 lib on Linux).

8.1.2.8 enum LLGL::CullMode [strong]

Polygon culling modes enumeration.

Enumerator

Disabled No culling.

Front Front face culling.

Back Back face culling.

8.1.2.9 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 (unsiged int).

Float 32-bit floating-point (float).

Double 64-bit real type (double).

8.1.2.10 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).

8.1.2.11 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.

8.1.2.12 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 Rigth 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

BrowserFavorits

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

8.1.2.13 enum LLGL::LogicOp [strong]

Logical pixel operation enumeration.

Remarks

These logical pixel operations are bitwise operations.

Note

Only supported with: OpenGL.

See also

GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::logicOp
https://www.opengl.org/sdk/docs/man/html/glLogicOp.xhtml

Enumerator

Keep Keep previous logical pixel operation.

Disabled Logical pixel operation is disabled.

Clear Resulting operation: 0.

Set Resulting operation: 1.

Copy Resulting operation: src.

InvertedCopy Resulting operation: ∼src.
Noop Resulting operation: dest.
Invert Resulting operation: ∼dest.
AND Resulting operation: src & dest.
NAND Resulting operation: ∼(src & dest)
OR Resulting operation: src | dest.
NOR Resulting operation: ∼(src | dest)
XOR Resulting operation: src ^ dest.
Equiv Resulting operation: ∼(src ^ dest)
ReverseAND Resulting operation: src & ∼dest.
InvertedAND Resulting operation: ∼src & dest.
ReverseOR Resulting operation: src | ∼dest.
InvertedOR Resulting operation: ∼src | dest.

8.1.2.14 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.

8.1.2.15 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.

8.1.2.16 enum LLGL::PrimitiveTopology [strong]

Primitive topology enumeration.

See also

GraphicsPipelineDescriptor::primitiveTopology

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.

8.1.2.17 enum LLGL::PrimitiveType [strong]

Primitive type enumeration.

Remarks

These entries are generic terms of a primitive topology.

See also

CommandBuffer::BeginStreamOutput

Enumerator

Points Generic term for all point primitives.

Remarks

This term refers to the following primitive topologies: PrimitiveTopology::PointList.

Lines Generic term for all line primitives.

Remarks

This term refers to the following primitive topologies: PrimitiveTopology::LineList, PrimitiveTopology::LineStrip, PrimitiveTopology::LineLoop, PrimitiveTopology::LineListAdjacency, and PrimitiveTopology::LineStripAdjacency.

Triangles Generic term for all triangle primitives.

Remarks

This term refers to the following primitive topologies: PrimitiveTopology::TriangleList, PrimitiveTopology::TriangleStrip, PrimitiveTopology::TriangleFan, PrimitiveTopology::TriangleListAdjacency, and PrimitiveTopology::TriangleStripAdjacency.

8.1.2.18 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.

8.1.2.19 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.

8.1.2.20 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).

8.1.2.21 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.

8.1.2.22 enum LLGL::ShadingLanguage [strong]

Shading language version enumation.

Remarks

These enumeration entries can be casted to an integer to get the respective version number. GLSL versions range from 110 (v.1.10) to 450 (v.4.50), and HLSL version range from 100200 (v.2.0) to 100500 (v.5.0).

Enumerator

Unsupported Enumeration entry if shaders are not supported.

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).

8.1.2.23 enum LLGL::StencilOp [strong]

Stencil operations enumeration.

Enumerator

Keep Keep the existing stencil data.

Zero Set stencil data to 0.

Replace Set the stencil data to the reference value.

See also

StencilFaceDescriptor::reference

IncClamp Increment the stencil value by 1, and clamp the result.

DecClamp Decrement the stencil value by 1, and clamp the result.

Invert Invert the stencil data.

IncWrap Increment the stencil value by 1, and wrap the result if necessary.

DecWrap Decrement the stencil value by 1, and wrap the result if necessary.

8.1.2.24 enum LLGL::StorageBufferType [strong]

Storage buffer type enumeration.

Note

Only supported with: Direct3D 11, Direct3D 12.

Enumerator

Buffer Typed buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

StructuredBuffer Structured buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

ByteAddressBuffer Byte-address buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

RWBuffer Typed read/write buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

RWStructuredBuffer Structured read/write buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

RWByteAddressBuffer Byte-address read/write buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

AppendStructuredBuffer Append structured buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

ConsumeStructuredBuffer Consume structured buffer.

Note

Only supported with: Direct3D 11, Direct3D 12.

8.1.2.25 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.

8.1.2.26 enum LLGL::TextureFilter [strong]

Texture sampling filter enumeration.

Enumerator

Nearest Take the nearest sample.

Linear Interpolate between two samples.

8.1.2.27 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 interger component.

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

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

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

R32SInt Sized format: red 32-bit un-normalized signed interger 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 interger components.

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

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

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

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

Note

Only supported with: OpenGL.

RGB16Sgn Sized format: red, green, blue 16-bit normalized signed interger 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.

8.1.2.28 enum LLGL::TextureType [strong]

Texture type enumeration.

Enumerator

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.

Texture2DMS 2-Dimensional multi-sample texture.

Texture2DMSArray 2-Dimensional multi-sample array texture.

8.1.2.29 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.

8.1.2.30 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.

8.1.2.31 enum LLGL::VectorType [strong]

Renderer vector types enumeration.

Enumerator

Float 1-Dimensional single precision floating-point vector (float in GLSL, float in HLSL).

Float2 2-Dimensional single precision floating-point vector (vec2 in GLSL, float2 in HLSL).

Float3 3-Dimensional single precision floating-point vector (vec3 in GLSL, float3 in HLSL).

Float4 4-Dimensional single precision floating-point vector (vec4 in GLSL, float4 in HLSL).

Double 1-Dimensional double precision floating-point vector (double in GLSL, double in HLSL).

Double2 2-Dimensional double precision floating-point vector (dvec2 in GLSL, double2 in HLSL).

Double3 3-Dimensional double precision floating-point vector (dvec3 in GLSL, double3 in HLSL).

Double4 4-Dimensional double precision floating-point vector (dvec4 in GLSL, double4 in HLSL).

Int 1-Dimensional signed integer vector (int in GLSL, int in HLSL).

Int2 2-Dimensional signed integer vector (ivec2 in GLSL, int2 in HLSL).

Int3 3-Dimensional signed integer vector (ivec3 in GLSL, int3 in HLSL).

Int4 4-Dimensional signed integer vector (ivec4 in GLSL, int4 in HLSL).

UInt 1-Dimensional unsigned integer vector (uint in GLSL, uint in HLSL).

UInt2 2-Dimensional unsigned integer vector (uvec2 in GLSL, uint2 in HLSL).

UInt3 3-Dimensional unsigned integer vector (uvec3 in GLSL, uint3 in HLSL).

UInt4 4-Dimensional unsigned integer vector (uvec4 in GLSL, uint4 in HLSL).

8.1.2.32 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).

8.1.3 Function Documentation

8.1.3.1 LLGL_EXPORT bool LLGL::CompareSWO (const VideoDisplayMode & *lhs*, const VideoDisplayMode & *rhs*)

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

8.1.3.2 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	*srcFormat*	Specifies the source image format.
in	*srcDataType*	Specifies the source data type.
in	*srcBuffer*	Pointer to the source image buffer which is to be converted.
in	*srcBufferSize*	Specifies the size (in bytes) of the source image buffer.
in	*dstFormat*	Specifies the destination image format.
in	*dstDataType*	Specifies the destination data type.
in	*threadCount*	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

std::invalid_argument	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

8.1.3.3 LLGL_EXPORT unsigned int LLGL::DataTypeSize (const DataType *dataType*)

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

8.1.3.4 LLGL_EXPORT unsigned int LLGL::ImageFormatSize (const ImageFormat *imageFormat*)

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

Parameters

in	*imageFormat*	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)

8.1.3.5 LLGL_EXPORT bool LLGL::IsArrayTexture (const TextureType *type*)

Returns true if the specified texture type is an array texture.

Returns

True if 'type' is . either TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, or TextureType::Texture2DMSArray.

8.1.3.6 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

8.1.3.7 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

8.1.3.8 LLGL_EXPORT bool LLGL::IsDepthStencilFormat (const ImageFormat *format*)

Returns true if the specified color foramt is a depth-stencil format, i.e. either ImageFormat::Depth or ImageFormat::DepthStencil.

8.1.3.9 LLGL_EXPORT bool LLGL::IsMultiSampleTexture (const TextureType *type*)

Returns true if the specified texture type is a multi-sample texture.

Returns

True if 'type' is either TextureType::Texture2DMS, or TextureType::Texture2DMSArray.

8.1.3.10 template<typename T > T LLGL::MaxColorValue ( ) [inline]

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

8.1.3.11 template<> bool LLGL::MaxColorValue< bool > ( ) [inline]

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

8.1.3.12 template<> unsigned char LLGL::MaxColorValue< unsigned char > ( ) [inline]

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

8.1.3.13 LLGL_EXPORT unsigned int LLGL::NumMipLevels (unsigned int *width*, unsigned int *height* = 1, unsigned int *depth* = 1)

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

Parameters

in	*width*	Specifies the texture width.
in	*height*	Specifies the texture height or number of layers for 1D array textures. By default 1 (if 1D textures are used).
in	*depth*	Specifies the texture depth or number of layers for 2D array textures. By default 1 (if 1D or 2D textures are used).

Remarks

The height and depth are optional parameters, so this function can be easily used for 1D, 2D, and 3D textures.

Returns

1 + floor(log2(max{ x, y, z })).

8.1.3.14 **LLGL_EXPORT** bool LLGL::operator!= (const **StreamOutputAttribute** & *lhs,* const **StreamOutputAttribute** & *rhs*)

8.1.3.15 **LLGL_EXPORT** bool LLGL::operator!= (const **VertexAttribute** & *lhs,* const **VertexAttribute** & *rhs*)

8.1.3.16 **LLGL_EXPORT** bool LLGL::operator!= (const **VsyncDescriptor** & *lhs,* const **VsyncDescriptor** & *rhs*)

8.1.3.17 **LLGL_EXPORT** bool LLGL::operator!= (const **VideoModeDescriptor** & *lhs,* const **VideoModeDescriptor** & *rhs*)

8.1.3.18 template<typename T , std::size_t N> bool LLGL::operator!= (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.19 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator* (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.20 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator* (const **Color**< T, N > & *lhs,* const T & *rhs*)

8.1.3.21 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator* (const T & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.22 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator+ (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.23 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator- (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.24 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator/ (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.25 template<typename T , std::size_t N> **Color**<T, N> LLGL::operator/ (const **Color**< T, N > & *lhs,* const T & *rhs*)

8.1.3.26 **LLGL_EXPORT** bool LLGL::operator== (const **VideoDisplayMode** & *lhs,* const **VideoDisplayMode** & *rhs*)

8.1.3.27 **LLGL_EXPORT** bool LLGL::operator== (const **StreamOutputAttribute** & *lhs,* const **StreamOutputAttribute** & *rhs*)

8.1.3.28 **LLGL_EXPORT** bool LLGL::operator== (const **VertexAttribute** & *lhs,* const **VertexAttribute** & *rhs*)

8.1.3.29 **LLGL_EXPORT** bool LLGL::operator== (const **VsyncDescriptor** & *lhs,* const **VsyncDescriptor** & *rhs*)

8.1.3.30 **LLGL_EXPORT** bool LLGL::operator== (const **VideoModeDescriptor** & *lhs,* const **VideoModeDescriptor** & *rhs*)

8.1.3.31 template<typename T , std::size_t N> bool LLGL::operator== (const **Color**< T, N > & *lhs,* const **Color**< T, N > & *rhs*)

8.1.3.32 **LLGL_EXPORT** void LLGL::VectorTypeFormat (const **VectorType** *vectorType,* **DataType** & *dataType,* unsigned int & *components*)

Retrieves the format of the specified vector type.

Parameters

in	*vectorType*	Specifies the vector type whose format is to be retrieved.
out	*dataType*	Specifies the output parameter for the resulting data type.
out	*components*	Specifiefs the output parameter for the resulting number of vector components.

8.1.3.33 LLGL_EXPORT unsigned int LLGL::VectorTypeSize (const VectorType *vectorType*)

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

8.2 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.

8.2.1 Function Documentation

8.2.1.1 LLGL_EXPORT int LLGL::Desktop::GetColorDepth ( )

Returns the desktop color depth (bits per pixel).

8.2.1.2 LLGL_EXPORT Size LLGL::Desktop::GetResolution ( )

Returns the desktop resolution.

8.2.1.3 LLGL_EXPORT bool LLGL::Desktop::ResetVideoMode ( )

Restes the standard video mode for the desktop.

8.2.1.4 LLGL_EXPORT bool LLGL::Desktop::SetVideoMode (const VideoModeDescriptor & *videoMode*)

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

8.3 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.

8.3.1 Function Documentation

8.3.1.1 LLGL_EXPORT void LLGL::Log::SetStdErr (std::ostream & *stream*)

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

8.3.1.2 LLGL_EXPORT void LLGL::Log::SetStdOut (std::ostream & *stream*)

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

8.3.1.3 LLGL_EXPORT std::ostream& LLGL::Log::StdErr ( )

Returns the standard output stream for error and warning messages.

8.3.1.4 LLGL_EXPORT std::ostream& LLGL::Log::StdOut ( )

Returns the standard output stream.

8.4 LLGL::Version Namespace Reference

Functions

- LLGL_EXPORT unsigned int GetMajor ()

  Returns the major LLGL version (e.g. 1 stands for "1.00").
- LLGL_EXPORT unsigned int GetMinor ()

  Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.
- LLGL_EXPORT unsigned int GetRevision ()

  Returns the revision version number. Must be less than 100.
- LLGL_EXPORT std::string GetStatus ()

  Returns the LLGL version status (either "Alpha", "Beta", or empty).
- LLGL_EXPORT unsigned int GetID ()

  Returns the full LLGL version as an ID number (e.g. 200317 stands for "2.03 (Rev. 17)").
- LLGL_EXPORT std::string GetString ()

  Returns the full LLGL version as a string (e.g. "0.01 Beta (Rev. 1)").

8.4.1 Function Documentation

8.4.1.1 LLGL_EXPORT unsigned int LLGL::Version::GetID ( )

Returns the full LLGL version as an ID number (e.g. 200317 stands for "2.03 (Rev. 17)").

8.4.1.2 LLGL_EXPORT unsigned int LLGL::Version::GetMajor ( )

Returns the major LLGL version (e.g. 1 stands for "1.00").

8.4.1.3 LLGL_EXPORT unsigned int LLGL::Version::GetMinor ( )

Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.

8.4.1.4 LLGL_EXPORT unsigned int LLGL::Version::GetRevision ( )

Returns the revision version number. Must be less than 100.

8.4.1.5 LLGL_EXPORT std::string LLGL::Version::GetStatus ( )

Returns the LLGL version status (either "Alpha", "Beta", or empty).

8.4.1.6 LLGL_EXPORT std::string LLGL::Version::GetString ( )

Returns the full LLGL version as a string (e.g. "0.01 Beta (Rev. 1)").

Chapter 9

Class Documentation

9.1 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.
- ColorRGBAf blendFactor { 0.0f, 0.0f, 0.0f, 0.0f }

  Specifies the blending color factor. By default (0, 0, 0, 0).
- std::vector< BlendTargetDescriptor > targets

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

9.1.1 Detailed Description

Blending state descriptor structure.

9.1.2 Member Data Documentation

9.1.2.1 bool LLGL::BlendDescriptor::blendEnabled = false

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

9.1.2.2 ColorRGBAf LLGL::BlendDescriptor::blendFactor { 0.0f, 0.0f, 0.0f, 0.0f }

Specifies the blending color factor. By default (0, 0, 0, 0).

Remarks

This is only used if any blending operations of any blending target is either BlendOp::BlendFactor or BlendOp::InvBlendFactor.

See also

BlendOp::BlendFactor
BlendOp::InvBlendFactor

9.1.2.3 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

9.2 LLGL::BlendTargetDescriptor Struct Reference

Blend target state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

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

9.2.1 Detailed Description

Blend target state descriptor structure.

9.2.2 Member Data Documentation

9.2.2.1 BlendArithmetic LLGL::BlendTargetDescriptor::alphaArithmetic = BlendArithmetic::Add

Alpha blending arithmetic. By default BlendArithmetic::Add.

9.2.2.2 BlendArithmetic LLGL::BlendTargetDescriptor::colorArithmetic = BlendArithmetic::Add

Color blending arithmetic. By default BlendArithmetic::Add.

9.2.2.3 ColorRGBAb LLGL::BlendTargetDescriptor::colorMask

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

9.2.2.4 BlendOp LLGL::BlendTargetDescriptor::destAlpha = BlendOp::InvSrcAlpha

Destination alpha blending operation. By default BlendOp::InvSrcAlpha.

9.2.2.5 BlendOp LLGL::BlendTargetDescriptor::destColor = BlendOp::InvSrcAlpha

Destination color blending operation. By default BlendOp::InvSrcAlpha.

9.2.2.6 BlendOp LLGL::BlendTargetDescriptor::srcAlpha = BlendOp::SrcAlpha

Source alpha blending operation. By default BlendOp::SrcAlpha.

9.2.2.7 BlendOp LLGL::BlendTargetDescriptor::srcColor = BlendOp::SrcAlpha

Source color blending operation. By default BlendOp::SrcAlpha.

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

- GraphicsPipelineFlags.h

9.3 LLGL::Buffer Class Reference

Hardware buffer interface.

`#include <Buffer.h>`

Public Member Functions

- Buffer (const Buffer &)=delete
- Buffer & operator= (const Buffer &)=delete
- virtual ~Buffer ()
- BufferType GetType () const

  Returns the type of this buffer.

Protected Member Functions

- Buffer (const BufferType type)

9.3.1 Detailed Description

Hardware buffer interface.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 LLGL::Buffer::Buffer (const Buffer &) [delete]

9.3.2.2 virtual LLGL::Buffer::~Buffer ( ) [virtual]

9.3.2.3 LLGL::Buffer::Buffer (const BufferType *type*) [protected]

9.3.3 Member Function Documentation

9.3.3.1 BufferType LLGL::Buffer::GetType ( ) const [inline]

Returns the type of this buffer.

9.3.3.2 Buffer& LLGL::Buffer::operator= (const Buffer &) [delete]

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

- Buffer.h

9.4 LLGL::BufferArray Class Reference

Array of hardware buffers interface.

```
#include <BufferArray.h>
```

Public Member Functions

- BufferArray (const BufferArray &)=delete
- BufferArray & operator= (const BufferArray &)=delete
- virtual ~BufferArray ()
- BufferType GetType () const

  Returns the type of buffers this array contains.

Protected Member Functions

- BufferArray (const BufferType type)

9.4.1 Detailed Description

Array of hardware buffers interface.

Remarks

This array can only contain buffers which are all from the same type, like an array of vertex buffers for instance.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 LLGL::BufferArray::BufferArray (const BufferArray &) [delete]

9.4.2.2 virtual LLGL::BufferArray::~BufferArray ( ) [virtual]

9.4.2.3 LLGL::BufferArray::BufferArray (const BufferType *type*) [protected]

9.4.3 Member Function Documentation

9.4.3.1 BufferType LLGL::BufferArray::GetType ( ) const [inline]

Returns the type of buffers this array contains.

9.4.3.2 BufferArray& LLGL::BufferArray::operator= (const BufferArray &) [delete]

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

- BufferArray.h

9.5 LLGL::BufferDescriptor Struct Reference

Hardware buffer descriptor structure.

`#include <BufferFlags.h>`

Classes

- struct IndexBufferDescriptor
- struct StorageBufferDescriptor
- struct VertexBufferDescriptor

  Vertex buffer descriptor structure.

Public Attributes

- BufferType type = BufferType::Vertex

  Hardware buffer type. By default BufferType::Vertex.
- unsigned int size = 0

  Buffer size (in bytes). By default 0.
- long flags = 0

  Specifies the buffer creation flags. By default 0.
- VertexBufferDescriptor vertexBuffer

  Vertex buffer type descriptor appendix.
- IndexBufferDescriptor indexBuffer

  Index buffer type descriptor appendix.
- StorageBufferDescriptor storageBuffer

  Storage buffer type descriptor appendix.

9.5.1 Detailed Description

Hardware buffer descriptor structure.

9.5.2 Member Data Documentation

9.5.2.1 long LLGL::BufferDescriptor::flags = 0

Specifies the buffer creation flags. By default 0.

Remarks

This can be bitwise OR combination of the entries of the BufferFlags enumeration.

See also

BufferFlags

9.5.2.2 IndexBufferDescriptor LLGL::BufferDescriptor::indexBuffer

Index buffer type descriptor appendix.

9.5.2.3 unsigned int LLGL::BufferDescriptor::size = 0

Buffer size (in bytes). By default 0.

Remarks

If the buffer type is a storage buffer (i.e. from the type BufferType::Storage), 'size' must be a multiple of 'storageBuffer.stride'.

9.5.2.4 StorageBufferDescriptor LLGL::BufferDescriptor::storageBuffer

Storage buffer type descriptor appendix.

9.5.2.5 BufferType LLGL::BufferDescriptor::type = BufferType::Vertex

Hardware buffer type. By default BufferType::Vertex.

9.5.2.6 VertexBufferDescriptor LLGL::BufferDescriptor::vertexBuffer

Vertex buffer type descriptor appendix.

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

- BufferFlags.h

9.6 LLGL::BufferFlags Struct Reference

Buffer flags enumeration.

```
#include <BufferFlags.h>
```

Public Types

- enum { MapReadAccess = (1 << 0), MapWriteAccess = (1 << 1), DynamicUsage = (1 << 2) }

9.6.1 Detailed Description

Buffer flags enumeration.

9.6.2 Member Enumeration Documentation

9.6.2.1 anonymous enum

Enumerator

MapReadAccess Buffer mapping with CPU read access is required.

See also

RenderSystem::MapBuffer

MapWriteAccess Buffer mapping with CPU write access is required.

See also

RenderSystem::MapBuffer

DynamicUsage Hint to the renderer that the buffer will be frequently updated from the CPU. This is useful for a constant buffer for instance, that is updated by the host program every frame.

See also

RenderSystem::WriteBuffer

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

- BufferFlags.h

9.7 LLGL::ClearFlags Struct Reference

Command buffer clear flags.

`#include <RenderContextFlags.h>`

Public Types

- enum {
  Color = (1 << 0), Depth = (1 << 1), Stencil = (1 << 2), ColorDepth = (Color | Depth),
  DepthStencil = (Depth | Stencil), All = (Color | Depth | Stencil) }

9.7.1 Detailed Description

Command buffer clear flags.

See also

CommandBuffer::Clear

9.7.2 Member Enumeration Documentation

9.7.2.1 anonymous enum

Enumerator

Color Clears the color buffer.

Depth Clears the depth buffer.

Stencil Clears the stencil buffer.

ColorDepth Clears the color and depth buffers.

DepthStencil Clears the depth and stencil buffers.

All Clears the all buffers (color, depth, and stencil).

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

- RenderContextFlags.h

9.8 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.

9.8.1 Detailed Description

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

Base color class with N components.

Template Parameters

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

9.8.2 Constructor & Destructor Documentation

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

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

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

9.8.3 Member Function Documentation

9.8.3.1 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

C	Specifies the static cast type.

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

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

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

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

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

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

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

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

Returns the specified vector component.

Parameters

in	*component*	Specifies the vector component index. This must be in the range [0, N).

9.8.3.10 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	*component*	Specifies the vector component index. This must be in the range [0, N).

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

Returns a pointer to the first element of this vector.

9.8.3.12 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.

9.8.4 Member Data Documentation

9.8.4.1 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

9.9 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.

9.9.1 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.

9.9.2 Constructor & Destructor Documentation

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

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

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

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

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

9.9.3 Member Function Documentation

9.9.3.1 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

C	Specifies the static cast type.

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

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

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

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

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

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

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

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

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, or 2.

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

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, or 2.

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

Returns a pointer to the first element of this color.

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

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

9.9.4 Member Data Documentation

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

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

Specifies the number of color components.

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

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

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

- ColorRGB.h

9.10 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.

9.10.1 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.

9.10.2 Constructor & Destructor Documentation

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

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

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

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

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

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

9.10.3 Member Function Documentation

9.10.3.1 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

C	Specifies the static cast type.

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

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

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

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

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

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

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

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

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, 2, or 3.

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

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, 2, or 3.

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

Returns a pointer to the first element of this color.

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

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

9.10.4 Member Data Documentation

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

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

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

Specifies the number of color components.

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

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

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

- ColorRGBA.h

9.11 LLGL::CommandBuffer Class Reference

Command buffer interface.

`#include <CommandBuffer.h>`

Public Member Functions

- CommandBuffer (const CommandBuffer &)=delete
- CommandBuffer & operator= (const CommandBuffer &)=delete
- virtual ~CommandBuffer ()
- virtual void SetGraphicsAPIDependentState (const GraphicsAPIDependentStateDescriptor &state)=0

  Sets a few low-level graphics API dependent states.
- virtual void SetViewport (const Viewport &viewport)=0

  Sets a single viewport.
- virtual void SetViewportArray (unsigned int numViewports, const Viewport *viewportArray)=0

  Sets an array of viewports.
- virtual void SetScissor (const Scissor &scissor)=0

  Sets a single scissor rectangle.
- virtual void SetScissorArray (unsigned int numScissors, const Scissor *scissorArray)=0

  Sets an array of 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 Clear (long flags)=0

  Clears the specified frame buffers of the active render target.
- virtual void SetVertexBuffer (Buffer &buffer)=0

  Sets the specified vertex buffer for subsequent drawing operations.
- virtual void SetVertexBufferArray (BufferArray &bufferArray)=0

  Sets the specified array of vertex buffers for subsequent drawing operations.
- virtual void SetIndexBuffer (Buffer &buffer)=0

  Sets the active index buffer for subsequent drawing operations.
- virtual void SetConstantBuffer (Buffer &buffer, unsigned int slot, long shaderStageFlags=ShaderStageFlags::AllStages)=0

  Sets the active constant buffer at the specified slot index for subsequent drawing and compute operations.
- virtual void SetConstantBufferArray (BufferArray &bufferArray, unsigned int startSlot, long shaderStageFlags=ShaderStageFlags::AllStages)=0

  Sets the active array of constant buffers at the specified start slot index.
- virtual void SetStorageBuffer (Buffer &buffer, unsigned int slot, long shaderStageFlags=ShaderStageFlags::AllStages)=0

  Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.
- virtual void SetStorageBufferArray (BufferArray &bufferArray, unsigned int startSlot, long shaderStageFlags=ShaderStageFlags::AllStages)=0

  Sets the active array of storage buffers at the specified start slot index.
- virtual void SetStreamOutputBuffer (Buffer &buffer)=0

  Sets the active stream-output buffer to the stream-output stage.
- virtual void SetStreamOutputBufferArray (BufferArray &bufferArray)=0

Sets the active array of stream-output buffers.

- virtual void BeginStreamOutput (const PrimitiveType primitiveType)=0

  Begins with stream-output for subsequent draw calls.

- virtual void EndStreamOutput ()=0

  Ends the current stream-output.

- virtual void SetTexture (Texture &texture, unsigned int slot, long shaderStageFlags=ShaderStageFlags::All↩ Stages)=0

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

- virtual void SetTextureArray (TextureArray &textureArray, unsigned int startSlot, long shaderStage↩ Flags=ShaderStageFlags::AllStages)=0

  Sets the active array of textures at the specified start slot index.

- 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 SetSamplerArray (SamplerArray &samplerArray, unsigned int startSlot, long shaderStage↩ Flags=ShaderStageFlags::AllStages)=0

  Sets the active array of samplers at the specified start slot index.

- virtual void SetRenderTarget (RenderTarget &renderTarget)=0

  Sets the specified render target as the new target for subsequent rendering commands.

- virtual void SetRenderTarget (RenderContext &renderContext)=0

  Sets the back buffer (or rather swap-chain) of the specified render context as the new target for subsequent rendering commands.

- 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 first↩ Index)=0
- virtual void DrawIndexedInstanced (unsigned int numVertices, unsigned int numInstances, unsigned int first↩ Index, int vertexOffset)=0
- virtual void DrawIndexedInstanced (unsigned int numVertices, unsigned int numInstances, unsigned int first↩ Index, int vertexOffset, unsigned int instanceOffset)=0

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

- virtual void Dispatch (unsigned int groupSizeX, unsigned int groupSizeY, unsigned int groupSizeZ)=0

  Dispachtes a compute command.
- virtual void SyncGPU ()=0

  Synchronizes the GPU, i.e. waits until the GPU has completed all pending commands.

Protected Member Functions

- CommandBuffer ()=default

9.11.1 Detailed Description

Command buffer interface.

Remarks

This is the main interface to commit graphics and compute commands to the GPU.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 LLGL::CommandBuffer::CommandBuffer (const CommandBuffer &) [delete]

9.11.2.2 virtual LLGL::CommandBuffer::~CommandBuffer ( ) [inline], [virtual]

9.11.2.3 LLGL::CommandBuffer::CommandBuffer ( ) [protected], [default]

9.11.3 Member Function Documentation

9.11.3.1 virtual void LLGL::CommandBuffer::BeginQuery (Query & *query*) [pure virtual]

Begins the specified query.

Parameters

in	*query*	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

9.11.3.2 virtual void LLGL::CommandBuffer::BeginRenderCondition (Query & *query,* const RenderConditionMode *mode*) [pure virtual]

Begins conditional rendering with the specified query object.

Parameters

in	*query*	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 QueryType::SamplesPassed, QueryType::AnySamplesPassed, or QueryType::AnySamplesPassedConservative.
in	*mode*	Specifies the mode of the render conidition.

Remarks

Here is a usage example:

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

See also

QueryType
RenderConditionMode

9.11.3.3 virtual void LLGL::CommandBuffer::BeginStreamOutput (const PrimitiveType *primitiveType*) [pure virtual]

Begins with stream-output for subsequent draw calls.

Parameters

in	*primitiveType*	Specifies the primitive output type of the last vertex processing shader stage (e.g. vertex- or geometry shader).

See also

EndStreamOutput

9.11.3.4 virtual void LLGL::CommandBuffer::Clear (long *flags*) [pure virtual]

Clears the specified frame buffers of the active render target.

Parameters

in	*flags*	Specifies the clear buffer flags. This can be a bitwise OR combination of the "ClearFlags" enumeration entries.

Remarks

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

See also

ClearFlags
SetClearColor
SetClearDepth
SetClearStencil

9.11.3.5 virtual void LLGL::CommandBuffer::Dispatch (unsigned int *groupSizeX*, unsigned int *groupSizeY*, unsigned int *groupSizeZ*) [pure virtual]

Dispachtes a compute command.

Parameters

in	*groupSizeX*	Specifies the number of thread groups in the X-dimension.
in	*groupSizeY*	Specifies the number of thread groups in the Y-dimension.
in	*groupSizeZ*	Specifies the number of thread groups in the Z-dimension.

See also

SetComputePipeline
RenderingCaps::maxNumComputeShaderWorkGroups

9.11.3.6 virtual void LLGL::CommandBuffer::Draw (unsigned int *numVertices*, unsigned int *firstVertex*) [pure virtual]

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

Parameters

in	*numVertices*	Specifies the number of vertices to generate.
in	*firstVertex*	Specifies the zero-based offset of the first vertex from the vertex buffer.

9.11.3.7 virtual void LLGL::CommandBuffer::DrawIndexed (unsigned int *numVertices*, unsigned int *firstIndex*) [pure virtual]

See also

DrawIndexed(unsigned int, unsigned int, int)

9.11.3.8 virtual void LLGL::CommandBuffer::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	*numVertices*	Specifies the number of vertices to generate.
in	*firstIndex*	Specifies the zero-based offset of the first index from the index buffer.
in	*vertexOffset*	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.

9.11.3.9 virtual void LLGL::CommandBuffer::DrawIndexedInstanced (unsigned int *numVertices*, unsigned int *numInstances*, unsigned int *firstIndex*) [pure virtual]

See also

DrawIndexedInstanced(unsigned int, unsigned int, unsigned int, int, unsigned int)

9.11.3.10 virtual void LLGL::CommandBuffer::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)

9.11.3.11 virtual void LLGL::CommandBuffer::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	*numVertices*	Specifies the number of vertices to generate.
in	*numInstances*	Specifies the number of instances to generate.
in	*firstIndex*	Specifies the zero-based offset of the first index from the index buffer.
in	*vertexOffset*	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.
in	*instanceOffset*	Specifies the zero-based instance offset which is added to each instance ID.

9.11.3.12 virtual void LLGL::CommandBuffer::DrawInstanced (unsigned int *numVertices*, unsigned int *firstVertex*, unsigned int *numInstances*) [pure virtual]

See also

DrawInstanced(unsigned int, unsigned int, unsigned int, unsigned int)

9.11.3.13 virtual void LLGL::CommandBuffer::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	*numVertices*	Specifies the number of vertices to generate.
in	*firstVertex*	Specifies the zero-based offset of the first vertex from the vertex buffer.
in	*numInstances*	Specifies the number of instances to generate.
in	*instanceOffset*	Specifies the zero-based instance offset which is added to each instance ID.

9.11.3.14 virtual void LLGL::CommandBuffer::EndQuery (Query & *query*) [pure virtual]

Ends the specified query.

See also

RenderSystem::CreateQuery
BeginQuery
QueryResult

9.11.3.15 virtual void LLGL::CommandBuffer::EndRenderCondition ( ) [pure virtual]

Ends the current render condition.

See also

BeginRenderCondition

9.11.3.16 virtual void LLGL::CommandBuffer::EndStreamOutput ( ) [pure virtual]

Ends the current stream-output.

See also

BeginStreamOutput

9.11.3.17 CommandBuffer& LLGL::CommandBuffer::operator= (const CommandBuffer &) [delete]

9.11.3.18 virtual bool LLGL::CommandBuffer::QueryResult (Query & *query*, std::uint64_t & *result*) [pure virtual]

Queries the result of the specified Query object.

Parameters

in	*query*	Specifies the Query object whose result is to be queried.
out	*result*	Specifies the output result.

Returns

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

9.11.3.19 virtual void LLGL::CommandBuffer::SetClearColor (const ColorRGBAf & *color*) [pure virtual]

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

9.11.3.20 virtual void LLGL::CommandBuffer::SetClearDepth (float *depth*) [pure virtual]

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

9.11.3.21 virtual void LLGL::CommandBuffer::SetClearStencil (int *stencil*) [pure virtual]

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

9.11.3.22 virtual void LLGL::CommandBuffer::SetComputePipeline (ComputePipeline & *computePipeline*) [pure virtual]

Sets the active compute pipeline state.

Parameters

in	*computePipeline*	Specifies the compuite 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

9.11.3.23 virtual void LLGL::CommandBuffer::SetConstantBuffer (Buffer & *buffer,* unsigned int *slot,* long *shaderStageFlags* = ShaderStageFlags::AllStages) [pure virtual]

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

Parameters

in	*buffer*	Specifies the constant buffer to set. This buffer must have been created with the buffer type: BufferType::Constant. This must not be an unspecified constant buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateBuffer" function or with the "RenderSystem::WriteBuffer" function.
in	*slot*	Specifies the slot index where to put the constant buffer.
in	*shaderStageFlags*	Specifies at which shader stages the constant buffer is to be set. By default all shader stages are affected.

See also

RenderSystem::WriteBuffer
ShaderStageFlags

9.11.3.24 virtual void LLGL::CommandBuffer::SetConstantBufferArray (BufferArray & *bufferArray*, unsigned int *startSlot*, long *shaderStageFlags* = ShaderStageFlags::AllStages) [pure virtual]

Sets the active array of constant buffers at the specified start slot index.

Parameters

in	*bufferArray*	Specifies the constant buffer array to set.

See also

RenderSystem::CreateBufferArray
SetConstantBuffer

9.11.3.25 virtual void LLGL::CommandBuffer::SetGraphicsAPIDependentState (const GraphicsAPIDependentStateDescriptor & *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.

9.11.3.26 virtual void LLGL::CommandBuffer::SetGraphicsPipeline (GraphicsPipeline & *graphicsPipeline*) [pure virtual]

Sets the active graphics pipeline state.

Parameters

in	*graphicsPipeline*	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

9.11.3.27 virtual void LLGL::CommandBuffer::SetIndexBuffer (Buffer & *buffer*) [pure virtual]

Sets the active index buffer for subsequent drawing operations.

Parameters

in	*buffer*	Specifies the index buffer to set. This buffer must have been created with the buffer type: BufferType::Index. This must not be an unspecified index buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateBuffer" function or with the "RenderSystem::WriteBuffer" function.

Remarks

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

See also

RenderSystem::WriteIndexBuffer

9.11.3.28 virtual void LLGL::CommandBuffer::SetRenderTarget (RenderTarget & *renderTarget*) [pure virtual]

Sets the specified render target as the new target for subsequent rendering commands.

Parameters

in	*renderTarget*	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

SetRenderTarget(RenderContext&)

9.11.3.29 virtual void LLGL::CommandBuffer::SetRenderTarget (RenderContext & *renderContext*) [pure virtual]

Sets the back buffer (or rather swap-chain) of the specified render context as the new target for subsequent rendering commands.

Remarks

Subsequent drawing operations will be rendered into the main framebuffer, which can then be presented onto the screen.

See also

SetRenderTarget(RenderTarget&)

9.11.3.30 virtual void LLGL::CommandBuffer::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	*sampler*	Specifies the sampler to set.
in	*slot*	Specifies the slot index where to put the sampler.

See also

RenderSystem::CreateSampler

9.11.3.31 virtual void LLGL::CommandBuffer::SetSamplerArray (SamplerArray & *samplerArray,* unsigned int *startSlot,* long *shaderStageFlags =* ShaderStageFlags::AllStages) [pure virtual]

Sets the active array of samplers at the specified start slot index.

See also

SetSampler

9.11.3.32 virtual void LLGL::CommandBuffer::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

9.11.3.33 virtual void LLGL::CommandBuffer::SetScissorArray (unsigned int *numScissors,* const Scissor * *scissorArray*) [pure virtual]

Sets an array of scissor rectangles.

Parameters

in	*numScissors*	Specifies the number of scissor rectangles to set.
in	*scissorArray*	Pointer to the array of scissor rectangles. This must not be null!

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.screen↩SpaceOriginLowerLeft' is true, the origin of each scissor rectangle is on the lower-left.

See also

SetGraphicsAPIDependentState

9.11.3.34 virtual void LLGL::CommandBuffer::SetStorageBuffer (Buffer & *buffer,* unsigned int *slot,* long *shaderStageFlags =* ShaderStageFlags::AllStages) [pure virtual]

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

Parameters

in	*buffer*	Specifies the storage buffer to set. This buffer must have been created with the buffer type: BufferType::Storage.
in	*slot*	Specifies the slot index where to put the storage buffer.
in	*shaderStageFlags*	Specifies at which shader stages the storage buffer is to be set and which resource views are to be set. By default all shader stages and all resource views are affected.

See also

RenderSystem::MapBuffer
RenderSystem::UnmapBuffer

9.11.3.35 virtual void LLGL::CommandBuffer::SetStorageBufferArray (BufferArray & *bufferArray,* unsigned int *startSlot,* long *shaderStageFlags =* ShaderStageFlags::AllStages) [pure virtual]

Sets the active array of storage buffers at the specified start slot index.

Parameters

in	*bufferArray*	Specifies the storage buffer array to set.

See also

RenderSystem::CreateBufferArray
SetStorageBuffer

9.11.3.36 virtual void LLGL::CommandBuffer::SetStreamOutputBuffer (Buffer & *buffer*) [pure virtual]

Sets the active stream-output buffer to the stream-output stage.

Parameters

in	*buffer*	Specifies the stream-output buffer to set. This buffer must have been created with the buffer type: BufferType::StreamOutput.

See also

RenderSystem::MapBuffer
RenderSystem::UnmapBuffer

9.11.3.37 virtual void LLGL::CommandBuffer::SetStreamOutputBufferArray (BufferArray & *bufferArray*) [pure virtual]

Sets the active array of stream-output buffers.

Parameters

in	*bufferArray*	Specifies the stream-output buffer array to set.

See also

RenderSystem::CreateBufferArray
SetStreamOutputBuffer

9.11.3.38 virtual void LLGL::CommandBuffer::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	*texture*	Specifies the texture to set.
in	*slot*	Specifies the slot index where to put the texture.

9.11.3.39 virtual void LLGL::CommandBuffer::SetTextureArray (TextureArray & *textureArray*, unsigned int *startSlot*, long *shaderStageFlags* = ShaderStageFlags::AllStages) [pure virtual]

Sets the active array of textures at the specified start slot index.

See also

SetTexture

9.11.3.40 virtual void LLGL::CommandBuffer::SetVertexBuffer (Buffer & *buffer*) [pure virtual]

Sets the specified vertex buffer for subsequent drawing operations.

Parameters

in	*buffer*	Specifies the vertex buffer to set. This buffer must have been created with the buffer type: BufferType::Vertex. This must not be an unspecified vertex buffer, i.e. it must be initialized with either the initial data in the "RenderSystem::CreateBuffer" function or with the "RenderSystem::WriteBuffer" function.

See also

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
SetVertexBufferArray

9.11.3.41 virtual void LLGL::CommandBuffer::SetVertexBufferArray (BufferArray & *bufferArray*) [pure virtual]

Sets the specified array of vertex buffers for subsequent drawing operations.

Parameters

in	*bufferArray*	Specifies the vertex buffer array to set.

See also

RenderSystem::CreateBufferArray
SetVertexBuffer

9.11.3.42 virtual void LLGL::CommandBuffer::SetViewport (const Viewport & *viewport*) [pure virtual]

Sets a single viewport.

Remarks

Similar to SetViewportArray but only a single viewport is set.

See also

SetViewportArray

9.11.3.43 virtual void LLGL::CommandBuffer::SetViewportArray (unsigned int *numViewports*, const Viewport * *viewportArray*) [pure virtual]

Sets an array of viewports.

Parameters

in	*numViewports*	Specifies the number of viewports to set.
in	*viewportArray*	Pointer to the array of viewports. This must not be null!

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.screenSpaceOrigin↩ LowerLeft' is true, the origin of each viewport is on the lower-left.

See also

SetGraphicsAPIDependentState

9.11.3.44 virtual void LLGL::CommandBuffer::SyncGPU ( ) [pure virtual]

Synchronizes the GPU, i.e. waits until the GPU has completed all pending commands.

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

- CommandBuffer.h

9.12 LLGL::ComputePipeline Class Reference

Compute pipeline interface.

`#include <ComputePipeline.h>`

Public Member Functions

- virtual ~ComputePipeline ()

9.12.1 Detailed Description

Compute pipeline interface.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 virtual LLGL::ComputePipeline::~ComputePipeline ( ) [inline], [virtual]

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

- ComputePipeline.h

9.13 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.

9.13.1 Detailed Description

Compute pipeline descriptor structure.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor ( ) [default]

9.13.2.2 LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor (ShaderProgram * *shaderProgram*) [inline]

9.13.3 Member Data Documentation

9.13.3.1 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

9.14 LLGL::ConstantBufferViewDescriptor Struct Reference

Constant buffer shader-view descriptor structure.

`#include <BufferFlags.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).

9.14.1 Detailed Description

Constant buffer shader-view descriptor structure.

Remarks

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

9.14.2 Member Data Documentation

9.14.2.1 unsigned int LLGL::ConstantBufferViewDescriptor::index = 0

Index of the constant buffer within the respective shader.

9.14.2.2 std::string LLGL::ConstantBufferViewDescriptor::name

Constant buffer name.

9.14.2.3 unsigned int LLGL::ConstantBufferViewDescriptor::size = 0

Buffer size (in bytes).

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

- BufferFlags.h

9.15 LLGL::RenderingProfiler::Counter Class Reference

Profiling counter class.

`#include <RenderingProfiler.h>`

Public Types

- using ValueType = unsigned int

Public Member Functions

- void Inc ()

  Increment internal counter by one.
- void Inc (ValueType value)

  Increment internal counter by the specified value.
- void Reset ()

  Reset internal counter to zero.
- ValueType Count () const

  Returns the internal counter value.
- operator unsigned int () const

  Returns the internal counter value (same as "Count()" function).

9.15.1 Detailed Description

Profiling counter class.

9.15.2 Member Typedef Documentation

9.15.2.1 using LLGL::RenderingProfiler::Counter::ValueType = unsigned int

9.15.3 Member Function Documentation

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

Returns the internal counter value.

9.15.3.2 void LLGL::RenderingProfiler::Counter::Inc ( ) [inline]

Increment internal counter by one.

9.15.3.3 void LLGL::RenderingProfiler::Counter::Inc (ValueType *value*) [inline]

Increment internal counter by the specified value.

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

Returns the internal counter value (same as "Count()" function).

9.15.3.5 void LLGL::RenderingProfiler::Counter::Reset ( ) [inline]

Reset internal counter to zero.

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

- RenderingProfiler.h

9.16 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.

9.16.1 Detailed Description

Depth state descriptor structure.

9.16.2 Member Data Documentation

9.16.2.1 CompareOp LLGL::DepthDescriptor::compareOp = CompareOp::Less

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

9.16.2.2 bool LLGL::DepthDescriptor::testEnabled = false

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

Remarks

If no pixel shader is used in the graphics pipeline, the depth test must be disabled.

9.16.2.3 bool LLGL::DepthDescriptor::writeEnabled = false

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

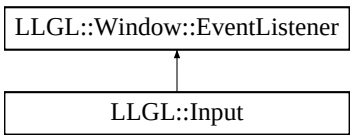
The documentation for this struct was generated from the following file:

- GraphicsPipelineFlags.h

9.17 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

9.17.1 Constructor & Destructor Documentation

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

9.17.2 Member Function Documentation

9.17.2.1 virtual void LLGL::Window::EventListener::OnChar (Window & *sender,* wchar_t *chr*) [protected], [virtual]

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

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

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

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

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

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

9.17.2.8 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.

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

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

9.17.3 Friends And Related Function Documentation

9.17.3.1 friend class Window [friend]

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

- Window.h

9.18 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

9.18.1 Detailed Description

Low-level graphics API dependent state descriptor union.

Remarks

This descriptor is used to compensate a few differences between OpenGL and Direct3D.

See also

RenderContext::SetGraphicsAPIDependentState

9.18.2 Constructor & Destructor Documentation

9.18.2.1 LLGL::GraphicsAPIDependentStateDescriptor::GraphicsAPIDependentStateDescriptor ( ) [inline]

9.18.3 Member Data Documentation

9.18.3.1 struct LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor LLGL::GraphicsAPIDependentStateDescriptor::stateOpenGL

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

- RenderContextFlags.h

9.19 LLGL::GraphicsPipeline Class Reference

Graphics pipeline interface.

`#include <GraphicsPipeline.h>`

Public Member Functions

- virtual ~GraphicsPipeline ()

9.19.1 Detailed Description

Graphics pipeline interface.

9.19.2 Constructor & Destructor Documentation

9.19.2.1 virtual LLGL::GraphicsPipeline::~GraphicsPipeline ( ) [inline], [virtual]

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

- GraphicsPipeline.h

9.20 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.

9.20.1 Detailed Description

Graphics pipeline descriptor structure.

Remarks

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

9.20.2 Member Data Documentation

9.20.2.1 BlendDescriptor LLGL::GraphicsPipelineDescriptor::blend

Specifies the blending state descriptor.

9.20.2.2 DepthDescriptor LLGL::GraphicsPipelineDescriptor::depth

Specifies the depth state descriptor.

9.20.2.3 PrimitiveTopology LLGL::GraphicsPipelineDescriptor::primitiveTopology = PrimitiveTopology::TriangleList

Specifies the primitive topology and ordering of the primitive data. By default PrimitiveTopology::TriangleList.

See also

PrimitiveTopology

9.20.2.4 RasterizerDescriptor LLGL::GraphicsPipelineDescriptor::rasterizer

Specifies the rasterizer state descriptor.

9.20.2.5 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

9.20.2.6 StencilDescriptor LLGL::GraphicsPipelineDescriptor::stencil

Specifies the stencil state descriptor.

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

- GraphicsPipelineFlags.h

9.21 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.
- unsigned int GetElementSize () const

  Returns the size (in bytes) for each image element (i.e. per "texel" or "pixel")

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.

9.21.1 Detailed Description

Image descriptor structure.

Remarks

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

9.21.2 Constructor & Destructor Documentation

9.21.2.1 LLGL::ImageDescriptor::ImageDescriptor ( ) [default]

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

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

Constructor for compressed image data.

9.21.3 Member Function Documentation

9.21.3.1 unsigned int LLGL::ImageDescriptor::GetElementSize ( ) const

Returns the size (in bytes) for each image element (i.e. per "texel" or "pixel")

Returns

```
ImageFormatSize(format) * DataTypeSize(dataType);
```

9.21.4 Member Data Documentation

9.21.4.1 const void* LLGL::ImageDescriptor::buffer = nullptr

Pointer to the image buffer.

9.21.4.2 unsigned int LLGL::ImageDescriptor::compressedSize = 0

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

9.21.4.3 DataType LLGL::ImageDescriptor::dataType = DataType::UInt8

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

9.21.4.4 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

9.22 LLGL::BufferDescriptor::IndexBufferDescriptor Struct Reference

`#include <BufferFlags.h>`

Public Attributes

- IndexFormat format

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

9.22.1 Member Data Documentation

9.22.1.1 IndexFormat LLGL::BufferDescriptor::IndexBufferDescriptor::format

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

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

- BufferFlags.h

9.23 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).

9.23.1 Constructor & Destructor Documentation

9.23.1.1 LLGL::IndexFormat::IndexFormat ( ) [default]

9.23.1.2 LLGL::IndexFormat::IndexFormat (const DataType *dataType*)

9.23.2 Member Function Documentation

9.23.2.1 DataType LLGL::IndexFormat::GetDataType ( ) const [inline]

Returns the data type of this index format.

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

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

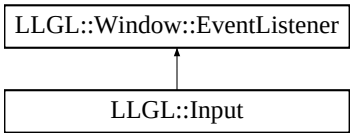
The documentation for this class was generated from the following file:

- IndexFormat.h

9.24 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

9.24.1 Constructor & Destructor Documentation

9.24.1.1 LLGL::Input::Input ( )

9.24.2 Member Function Documentation

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

Returns the entered characters.

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

Returns the global mouse motion.

9.24.2.3 const Point& LLGL::Input::GetMousePosition ( ) const [inline]

Returns the local mouse position.

9.24.2.4 int LLGL::Input::GetWheelMotion ( ) const [inline]

Returns the mouse wheel motion.

9.24.2.5 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.

9.24.2.6 bool LLGL::Input::KeyDown (Key *keyCode*) const

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

9.24.2.7 bool LLGL::Input::KeyPressed (Key *keyCode*) const

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

9.24.2.8 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

9.25 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 ()

  Blocks further occurrences of this message.
- void BlockAfter (std::size_t occurrences)

  Blocks further occurrences of this message after the specified amount of messages have been occurred.
- const std::string & GetText () const

  Returns the message text.
- const std::string & GetSource () const

  Returns the source function where this message occurred.
- std::size_t GetOccurrences () const

  Returns the number of occurrences of this message.
- bool IsBlocked () const

  Returns true if this message has already been blocked.

Protected Member Functions

- void IncOccurrence ()

Friends

- class RenderingDebugger

9.25.1 Detailed Description

Rendering debugger message class.

9.25.2 Constructor & Destructor Documentation

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

9.25.2.2 LLGL::RenderingDebugger::Message::Message (const Message &) [default]

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

9.25.3 Member Function Documentation

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

Blocks further occurrences of this message.

9.25.3.2 void LLGL::RenderingDebugger::Message::BlockAfter (std::size_t *occurrences*)

Blocks further occurrences of this message after the specified amount of messages have been occurred.

9.25.3.3 std::size_t LLGL::RenderingDebugger::Message::GetOccurrences ( ) const [inline]

Returns the number of occurrences of this message.

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

Returns the source function where this message occurred.

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

Returns the message text.

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

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

Returns true if this message has already been blocked.

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

9.25.4 Friends And Related Function Documentation

9.25.4.1 friend class RenderingDebugger [friend]

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

- RenderingDebugger.h

9.26 LLGL::MultiSamplingDescriptor Struct Reference

Multi-sampling descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Member Functions

- MultiSamplingDescriptor ()=default
- MultiSamplingDescriptor (unsigned int samples)
- unsigned int SampleCount () const

  Returns the sample count for the state of this multi-sampling descriptor.

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.

9.26.1 Detailed Description

Multi-sampling descriptor structure.

9.26.2 Constructor & Destructor Documentation

9.26.2.1 LLGL::MultiSamplingDescriptor::MultiSamplingDescriptor ( ) [default]

9.26.2.2 LLGL::MultiSamplingDescriptor::MultiSamplingDescriptor (unsigned int *samples*) [inline]

9.26.3 Member Function Documentation

9.26.3.1 unsigned int LLGL::MultiSamplingDescriptor::SampleCount ( ) const [inline]

Returns the sample count for the state of this multi-sampling descriptor.

Returns

max{ 1, samples } if multi-sampling is enabled, otherwise 1.

9.26.4 Member Data Documentation

9.26.4.1 bool LLGL::MultiSamplingDescriptor::enabled = false

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

9.26.4.2 unsigned int LLGL::MultiSamplingDescriptor::samples = 1

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

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

- GraphicsPipelineFlags.h

9.27 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

9.27.1 Detailed Description

Linux native context handle structure.

Win32 native context handle structure.

MacOS native context handle structure.

9.27.2 Member Data Documentation

9.27.2.1 ::Colormap LLGL::NativeContextHandle::colorMap

9.27.2.2 ::Display* LLGL::NativeContextHandle::display

9.27.2.3 HWND LLGL::NativeContextHandle::parentWindow

9.27.2.4 NSWindow* LLGL::NativeContextHandle::parentWindow

9.27.2.5 ::Window LLGL::NativeContextHandle::parentWindow

9.27.2.6 int LLGL::NativeContextHandle::screen

9.27.2.7 ::XVisualInfo* LLGL::NativeContextHandle::visual

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

- LinuxNativeHandle.h
- MacOSNativeHandle.h
- Win32NativeHandle.h

9.28 LLGL::NativeHandle Struct Reference

Linux native handle structure.

`#include <LinuxNativeHandle.h>`

Public Attributes

- ::Display * display
- ::Window window
- ::XVisualInfo * visual
- NSWindow * window
- HWND window

9.28.1 Detailed Description

Linux native handle structure.

Win32 native handle structure.

MacOS native handle structure.

9.28.2 Member Data Documentation

9.28.2.1 ::Display* LLGL::NativeHandle::display

9.28.2.2 ::XVisualInfo* LLGL::NativeHandle::visual

9.28.2.3 NSWindow* LLGL::NativeHandle::window

9.28.2.4 HWND LLGL::NativeHandle::window

9.28.2.5 ::Window LLGL::NativeHandle::window

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

- LinuxNativeHandle.h
- MacOSNativeHandle.h
- Win32NativeHandle.h

9.29 LLGL::ProfileOpenGLDescriptor Struct Reference

OpenGL profile descriptor structure.

`#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 disbaled.
- 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.

9.29.1 Detailed Description

OpenGL profile descriptor structure.

9.29.2 Member Data Documentation

9.29.2.1 bool LLGL::ProfileOpenGLDescriptor::coreProfile = false

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

Remarks

This requires 'extProfile' to be enabled.

9.29.2.2 bool LLGL::ProfileOpenGLDescriptor::debugDump = false

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

9.29.2.3 bool LLGL::ProfileOpenGLDescriptor::extProfile = false

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

9.29.2.4 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

9.30 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 this query.

Protected Member Functions

- Query (const QueryType type)

9.30.1 Detailed Description

Query interface.

9.30.2 Constructor & Destructor Documentation

9.30.2.1 LLGL::Query::Query (const Query &) [delete]

9.30.2.2 virtual LLGL::Query::~Query ( ) [virtual]

9.30.2.3 LLGL::Query::Query (const QueryType *type*) [protected]

9.30.3 Member Function Documentation

9.30.3.1 QueryType LLGL::Query::GetType ( ) const [inline]

Returns the type of this query.

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

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

- Query.h

9.31 LLGL::QueryDescriptor Struct Reference

Query descriptor structure.

`#include <QueryFlags.h>`

Public Member Functions

- QueryDescriptor ()=default
- QueryDescriptor (QueryType type, bool renderCondition=false)

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.

9.31.1 Detailed Description

Query descriptor structure.

9.31.2 Constructor & Destructor Documentation

9.31.2.1 LLGL::QueryDescriptor::QueryDescriptor ( ) [default]

9.31.2.2 LLGL::QueryDescriptor::QueryDescriptor (QueryType *type*, bool *renderCondition* = false) [inline]

9.31.3 Member Data Documentation

9.31.3.1 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.

9.31.3.2 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:

- QueryFlags.h

9.32 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

  Polygon face culling mode. By default CullMode::Disabled.
- int depthBias = 0
- float depthBiasClamp = 0.0f
- float slopeScaledDepthBias = 0.0f
- MultiSamplingDescriptor multiSampling

  (Multi-)sampling descriptor.
- 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

  Specifies whether scissor test is enabled or disabled. By default disabled.
- bool antiAliasedLineEnabled = false

  Specifies whether lines are rendered with or without anti-aliasing. By default disabled.
- bool conservativeRasterization = false

  If ture, conservative rasterization is enabled.

9.32.1 Detailed Description

Rasterizer state descriptor structure.

9.32.2 Member Data Documentation

9.32.2.1 bool LLGL::RasterizerDescriptor::antiAliasedLineEnabled = false

Specifies whether lines are rendered with or without anti-aliasing. By default disabled.

9.32.2.2 bool LLGL::RasterizerDescriptor::conservativeRasterization = false

If ture, conservative rasterization is enabled.

Note

Only supported with: Direct3D 12 (or OpenGL if the extension "GL_NV_conservative_raster" or "GL_INTE↩ L_conservative_rasterization" is supported).

See also

https://www.opengl.org/registry/specs/NV/conservative_raster.txt
https://www.opengl.org/registry/specs/INTEL/conservative_rasterization.↩ txt

9.32.2.3 CullMode LLGL::RasterizerDescriptor::cullMode = CullMode::Disabled

Polygon face culling mode. By default CullMode::Disabled.

9.32.2.4 int LLGL::RasterizerDescriptor::depthBias = 0

9.32.2.5 float LLGL::RasterizerDescriptor::depthBiasClamp = 0.0f

9.32.2.6 bool LLGL::RasterizerDescriptor::depthClampEnabled = false

9.32.2.7 bool LLGL::RasterizerDescriptor::frontCCW = false

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

9.32.2.8 MultiSamplingDescriptor LLGL::RasterizerDescriptor::multiSampling

(Multi-)sampling descriptor.

9.32.2.9 PolygonMode LLGL::RasterizerDescriptor::polygonMode = PolygonMode::Fill

Polygon render mode. By default PolygonMode::Fill.

9.32.2.10 bool LLGL::RasterizerDescriptor::scissorTestEnabled = false

Specifies whether scissor test is enabled or disabled. By default disabled.

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

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

- GraphicsPipelineFlags.h

9.33 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 back buffer on this render context.
- Window & GetWindow () const

  Returns the window which is used to draw all content.
- 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-sychronization (Vsync) configuration for this render context.
- const VideoModeDescriptor & GetVideoMode () const

  Returns the video mode for this render context.

Protected Member Functions

- RenderContext ()=default
- void SetOrCreateWindow (const std::shared_ptr< Window > &window, VideoModeDescriptor &videoMode↩ Desc, const void *windowContext)

  Sets the render context window or creates one if 'window' is null.
- void ShareWindowAndVideoMode (RenderContext &other)

  Shares the window and video mode with another render context.

9.33.1 Detailed Description

Render context interface.

Remarks

Each render context has its own window and back buffer (or rather swap-chain) to draw into.

9.33.2 Constructor & Destructor Documentation

9.33.2.1 LLGL::RenderContext::RenderContext (const RenderContext &) [delete]

9.33.2.2 virtual LLGL::RenderContext::~RenderContext ( ) [virtual]

9.33.2.3 LLGL::RenderContext::RenderContext ( ) [protected], [default]

9.33.3 Member Function Documentation

9.33.3.1 const VideoModeDescriptor& LLGL::RenderContext::GetVideoMode ( ) const [inline]

Returns the video mode for this render context.

9.33.3.2 Window& LLGL::RenderContext::GetWindow ( ) const [inline]

Returns the window which is used to draw all content.

9.33.3.3 RenderContext& LLGL::RenderContext::operator= (const RenderContext &) [delete]

9.33.3.4 virtual void LLGL::RenderContext::Present ( ) [pure virtual]

Presents the back buffer on this render context.

9.33.3.5 void LLGL::RenderContext::SetOrCreateWindow (const std::shared_ptr< Window > & *window*, VideoModeDescriptor & *videoModeDesc*, const void * *windowContext*) [protected]

Sets the render context window or creates one if 'window' is null.

Parameters

in	*window*	Optional shared pointer to a window which will be used as main render target. If this is null, a new window is created for this render context.
in,out	*videoModeDesc*	Specifies the video mode descriptor. This is used for reading only if 'window' is null, otherwise it is used for writing only and the 'resolution' field will be set to the size of the specified window.
in	*Optional*	pointer to a NativeContextHandle structure.

See also

WindowDescriptor::windowContext

9.33.3.6 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 if the back buffer is required, so a subsequent call to "↩ CommandBuffer::SetRenderContext" is required!

See also

CommandBuffer::SetRenderContext(RenderContext&)

9.33.3.7 virtual void LLGL::RenderContext::SetVsync (const VsyncDescriptor & *vsyncDesc*) [pure virtual]

Sets the new vertical-sychronization (Vsync) configuration for this render context.

9.33.3.8 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.

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

- RenderContext.h

9.34 LLGL::RenderContextDescriptor Struct Reference

Render context descriptor structure.

`#include <RenderContextDescriptor.h>`

Public Attributes

- VsyncDescriptor vsync

  Vertical-synchronization (Vsync) descriptor.
- MultiSamplingDescriptor multiSampling

  Sampling descriptor.
- VideoModeDescriptor videoMode

  Video mode descriptor.
- ProfileOpenGLDescriptor profileOpenGL

  OpenGL profile descriptor (to switch between compatability or core profile).
- DebugCallback debugCallback

  Debuging callback descriptor.

9.34.1 Detailed Description

Render context descriptor structure.

9.34.2 Member Data Documentation

9.34.2.1 DebugCallback LLGL::RenderContextDescriptor::debugCallback

Debuging callback descriptor.

9.34.2.2 MultiSamplingDescriptor LLGL::RenderContextDescriptor::multiSampling

Sampling descriptor.

9.34.2.3 ProfileOpenGLDescriptor LLGL::RenderContextDescriptor::profileOpenGL

OpenGL profile descriptor (to switch between compatability or core profile).

9.34.2.4 VideoModeDescriptor LLGL::RenderContextDescriptor::videoMode

Video mode descriptor.

9.34.2.5 VsyncDescriptor LLGL::RenderContextDescriptor::vsync

Vertical-synchronization (Vsync) descriptor.

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

- RenderContextDescriptor.h

9.35 LLGL::RendererID Struct Reference

Renderer identification number enumeration.

`#include <RenderSystemFlags.h>`

Static Public Attributes

- static const unsigned int OpenGL = 0x00000001
  ID number for the OpenGL renderer.
- static const unsigned int Direct3D11 = 0x00000002
  ID number for the Direct3D 11 renderer.
- static const unsigned int Direct3D12 = 0x00000003
  ID number for the Direct3D 12 renderer.
- static const unsigned int Vulkan = 0x00000004
  ID number for the Vulkan renderer.
- static const unsigned int Reserved5 = 0x00000005
  ID number for a reserved future renderer.
- static const unsigned int Reserved6 = 0x00000006
  ID number for a reserved future renderer.
- static const unsigned int Reserved7 = 0x00000007
  ID number for a reserved future renderer.
- static const unsigned int Reserved8 = 0x00000008
  ID number for a reserved future renderer.
- static const unsigned int Reserved9 = 0x00000009
  ID number for a reserved future renderer.
- static const unsigned int ReservedA = 0x0000000a
  ID number for a reserved future renderer.
- static const unsigned int ReservedB = 0x0000000b
  ID number for a reserved future renderer.
- static const unsigned int ReservedC = 0x0000000c
  ID number for a reserved future renderer.
- static const unsigned int ReservedD = 0x0000000d
  ID number for a reserved future renderer.
- static const unsigned int ReservedE = 0x0000000e
  ID number for a reserved future renderer.
- static const unsigned int ReservedF = 0x0000000f
  ID number for a reserved future renderer.

9.35.1 Detailed Description

Renderer identification number enumeration.

See also

RendererInfo::rendererID

9.35.2 Member Data Documentation

9.35.2.1 const unsigned int LLGL::RendererID::Direct3D11 = 0x00000002 [static]

ID number for the Direct3D 11 renderer.

9.35.2.2 const unsigned int LLGL::RendererID::Direct3D12 = 0x00000003 [static]

ID number for the Direct3D 12 renderer.

9.35.2.3 const unsigned int LLGL::RendererID::OpenGL = 0x00000001 [static]

ID number for the OpenGL renderer.

9.35.2.4 const unsigned int LLGL::RendererID::Reserved5 = 0x00000005 [static]

ID number for a reserved future renderer.

9.35.2.5 const unsigned int LLGL::RendererID::Reserved6 = 0x00000006 [static]

ID number for a reserved future renderer.

9.35.2.6 const unsigned int LLGL::RendererID::Reserved7 = 0x00000007 [static]

ID number for a reserved future renderer.

9.35.2.7 const unsigned int LLGL::RendererID::Reserved8 = 0x00000008 [static]

ID number for a reserved future renderer.

9.35.2.8 const unsigned int LLGL::RendererID::Reserved9 = 0x00000009 [static]

ID number for a reserved future renderer.

9.35.2.9 const unsigned int LLGL::RendererID::ReservedA = 0x0000000a [static]

ID number for a reserved future renderer.

9.35.2.10 const unsigned int LLGL::RendererID::ReservedB = 0x0000000b [static]

ID number for a reserved future renderer.

9.35.2.11 const unsigned int LLGL::RendererID::ReservedC = 0x0000000c [static]

ID number for a reserved future renderer.

9.35.2.12 const unsigned int LLGL::RendererID::ReservedD = 0x0000000d [static]

ID number for a reserved future renderer.

9.35.2.13 const unsigned int LLGL::RendererID::ReservedE = 0x0000000e [static]

ID number for a reserved future renderer.

9.35.2.14 const unsigned int LLGL::RendererID::ReservedF = 0x0000000f [static]

ID number for a reserved future renderer.

9.35.2.15 const unsigned int LLGL::RendererID::Vulkan = 0x00000004 [static]

ID number for the Vulkan renderer.

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

- RenderSystemFlags.h

9.36 LLGL::RendererInfo Struct Reference

Renderer basic information structure.

`#include <RenderSystemFlags.h>`

Public Attributes

- std::string rendererName

  Rendering API name and version (e.g. "OpenGL 4.5.0").
- std::string deviceName

  Renderer device name (e.g. "GeForce GTX 1070/PCIe/SSE2").
- std::string vendorName

  Vendor name of the renderer device (e.g. "NVIDIA Corporation").
- std::string shadingLanguageName

  Shading language version (e.g. "GLSL 4.50").
- unsigned int rendererID = 0

  Rendering API identification number.

9.36.1 Detailed Description

Renderer basic information structure.

9.36.2 Member Data Documentation

9.36.2.1 std::string LLGL::RendererInfo::deviceName

Renderer device name (e.g. "GeForce GTX 1070/PCIe/SSE2").

9.36.2.2 unsigned int LLGL::RendererInfo::rendererID = 0

Rendering API identification number.

Remarks

This can be a value of the RendererID entries. Since the render system is modular, a new render system can use its own ID number.

See also

RendererID

9.36.2.3 std::string LLGL::RendererInfo::rendererName

Rendering API name and version (e.g. "OpenGL 4.5.0").

9.36.2.4 std::string LLGL::RendererInfo::shadingLanguageName

Shading language version (e.g. "GLSL 4.50").

9.36.2.5 std::string LLGL::RendererInfo::vendorName

Vendor name of the renderer device (e.g. "NVIDIA Corporation").

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

- RenderSystemFlags.h

9.37 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.
- ShadingLanguage shadingLanguage = ShadingLanguage::Unsupported

  Latest suppported shading language.
- 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 hasMultiSampleTextures = false

  Specifies whether multi-sample 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

  Speciifes 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.
- bool hasStreamOutputs = false

  Specifies whether stream-output 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.

9.37.1 Detailed Description

Rendering capabilities structure.

9.37.2 Member Data Documentation

9.37.2.1 ClippingRange LLGL::RenderingCaps::clippingRange = ClippingRange::ZeroToOne

Clipping depth range.

9.37.2.2 bool LLGL::RenderingCaps::has3DTextures = false

Specifies whether 3D textures are supported.

See also

TextureType::Texture3D

9.37.2.3 bool LLGL::RenderingCaps::hasComputeShaders = false

Speciifes whether compute shaders are supported.

9.37.2.4 bool LLGL::RenderingCaps::hasConservativeRasterization = false

Specifies whether conservative rasterization is supported.

See also

RasterizerDescriptor::conservativeRasterization

9.37.2.5 bool LLGL::RenderingCaps::hasConstantBuffers = false

Specifies whether constant buffers (also "uniform buffer objects") are supported.

See also

BufferType::Constant

9.37.2.6 bool LLGL::RenderingCaps::hasCubeTextureArrays = false

Specifies whether cube array textures are supported.

See also

TextureType::TextureCubeArray

9.37.2.7 bool LLGL::RenderingCaps::hasCubeTextures = false

Specifies whether cube textures are supported.

See also

TextureType::TextureCube

9.37.2.8 bool LLGL::RenderingCaps::hasGeometryShaders = false

Specifies whether geometry shaders are supported.

9.37.2.9 bool LLGL::RenderingCaps::hasInstancing = false

Specifies whether hardware instancing is supported.

See also

RenderContext::DrawInstanced(unsigned int, unsigned int, unsigned int)
RenderContext::DrawIndexedInstanced(unsigned int, unsigned int, unsigned int)
RenderContext::DrawIndexedInstanced(unsigned int, unsigned int, unsigned int, int)

9.37.2.10 bool LLGL::RenderingCaps::hasMultiSampleTextures = false

Specifies whether multi-sample textures are supported.

See also

TextureType::Texture2DMS
TextureType::Texture2DMSArray

9.37.2.11 bool LLGL::RenderingCaps::hasOffsetInstancing = false

Specifies whether hardware instancing with instance offsets is supported.

See also

RenderContext::DrawInstanced(unsigned int, unsigned int, unsigned int, unsigned int)
RenderContext::DrawIndexedInstanced(unsigned int, unsigned int, unsigned int, int, unsigned int)

9.37.2.12 bool LLGL::RenderingCaps::hasRenderTargets = false

Specifies whether render targets (also "frame buffer objects") are supported.

9.37.2.13 bool LLGL::RenderingCaps::hasSamplers = false

Specifies whether samplers are supported.

9.37.2.14 bool LLGL::RenderingCaps::hasStorageBuffers = false

Specifies whether storage buffers (also "read/write buffers") are supported.

See also

BufferType::Storage

9.37.2.15 bool LLGL::RenderingCaps::hasStreamOutputs = false

Specifies whether stream-output is supported.

See also

ShaderSource::streamOutput
CommandBuffer::BeginStreamOutput

9.37.2.16 bool LLGL::RenderingCaps::hasTessellationShaders = false

Specifies whether tessellation shaders are supported.

9.37.2.17 bool LLGL::RenderingCaps::hasTextureArrays = false

Specifies whether 1D- and 2D array textures are supported.

See also

TextureType::Texture1DArray
TextureType::Texture2DArray

9.37.2.18 bool LLGL::RenderingCaps::hasUniforms = false

Specifies whether individual shader uniforms are supported (typically only for OpenGL 2.0+).

See also

ShaderProgram::LockShaderUniform

9.37.2.19 bool LLGL::RenderingCaps::hasViewportArrays = false

Specifies whether multiple viewports, depth-ranges, and scissors are supported at once.

9.37.2.20 int LLGL::RenderingCaps::max1DTextureSize = 0

Specifies maximum size of each 1D texture.

9.37.2.21 int LLGL::RenderingCaps::max2DTextureSize = 0

Specifies maximum size of each 2D texture (for width and height).

9.37.2.22 int LLGL::RenderingCaps::max3DTextureSize = 0

Specifies maximum size of each 3D texture (for width, height, and depth).

9.37.2.23 int LLGL::RenderingCaps::maxAnisotropy = 0

Specifies maximum anisotropy texture filter.

See also

SamplerDescriptor::maxAnisotropy

9.37.2.24 Gs::Vector3ui LLGL::RenderingCaps::maxComputeShaderWorkGroupSize

Specifies maximum work group size in a compute shader.

9.37.2.25 unsigned int LLGL::RenderingCaps::maxConstantBufferSize = 0

Specifies maximum size (in bytes) of each constant buffer.

9.37.2.26 int LLGL::RenderingCaps::maxCubeTextureSize = 0

Specifies maximum size of each cube texture (for width and height).

9.37.2.27 Gs::Vector3ui LLGL::RenderingCaps::maxNumComputeShaderWorkGroups

Specifies maximum number of work groups in a compute shader.

See also

RenderContext::Dispatch

9.37.2.28 unsigned int LLGL::RenderingCaps::maxNumRenderTargetAttachments = 0

Specifies maximum number of attachment points for each render target.

9.37.2.29 unsigned int LLGL::RenderingCaps::maxNumTextureArrayLayers = 0

Specifies maximum number of texture array layers (for 1D-, 2D-, and cube textures).

9.37.2.30 int LLGL::RenderingCaps::maxPatchVertices = 0

Specifies maximum number of patch control points.

9.37.2.31 ScreenOrigin LLGL::RenderingCaps::screenOrigin = ScreenOrigin::UpperLeft

Screen coordinate system origin.

Remarks

This determines the coordinate space of viewports, scissors, and framebuffers.

9.37.2.32 ShadingLanguage LLGL::RenderingCaps::shadingLanguage = ShadingLanguage::Unsupported

Latest suppported shading language.

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

- RenderSystemFlags.h

9.38 LLGL::RenderingDebugger Class Reference

Rendering debugger interface.

`#include <RenderingDebugger.h>`

Classes

- class Message

  Rendering debugger message class.

Public Member Functions

- virtual ~RenderingDebugger ()
- void SetSource (const char *source)

  Sets the new source function name.
- void PostError (const ErrorType type, const std::string &message)

  Posts an error message.
- void PostWarning (const WarningType type, const std::string &message)

  Posts a warning message.

Protected Member Functions

- RenderingDebugger ()=default
- virtual void OnError (ErrorType type, Message &message)
- virtual void OnWarning (WarningType type, Message &message)

9.38.1 Detailed Description

Rendering debugger interface.

Remarks

This can be used to profile the renderer draw calls and buffer updates.

9.38.2 Constructor & Destructor Documentation

9.38.2.1 virtual LLGL::RenderingDebugger::~RenderingDebugger ( ) [virtual]

9.38.2.2 LLGL::RenderingDebugger::RenderingDebugger ( ) [protected], [default]

9.38.3 Member Function Documentation

9.38.3.1 virtual void LLGL::RenderingDebugger::OnError (ErrorType *type,* Message & *message*) [protected], [virtual]

9.38.3.2 virtual void LLGL::RenderingDebugger::OnWarning (WarningType *type,* Message & *message*) [protected], [virtual]

9.38.3.3 void LLGL::RenderingDebugger::PostError (const ErrorType *type,* const std::string & *message*)

Posts an error message.

Parameters

in	*type*	Specifies the type of error.
in	*message*	Specifies the string which describes the failure.
in	*source*	Specifies the string which describes the source (typically the function where the failure happend).

9.38.3.4 void LLGL::RenderingDebugger::PostWarning (const WarningType *type,* const std::string & *message*)

Posts a warning message.

Parameters

in	*type*	Specifies the type of error.
in	*message*	Specifies the string which describes the warning.
in	*source*	Specifies the string which describes the source (typically the function where the failure happend).

9.38.3.5 void LLGL::RenderingDebugger::SetSource (const char * *source*)

Sets the new source function name.

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

- RenderingDebugger.h

9.39 LLGL::RenderingProfiler Class Reference

Rendering profiler model class.

`#include <RenderingProfiler.h>`

Classes

- class Counter

  Profiling counter class.

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 writeBuffer

  Counter for buffer writings.
- Counter mapBuffer

  Counter for buffer mappings.
- Counter setVertexBuffer

  Counter for vertex buffer bindings.
- Counter setIndexBuffer

  Counter for index buffer bindings.
- Counter setConstantBuffer

  Counter for constant buffer bindings.
- Counter setStorageBuffer

  Counter for storage buffer bindings.
- Counter setStreamOutputBuffer

  Counter for stream-output buffer bindings.
- Counter setGraphicsPipeline

  Counter for graphics pipeline bindings.
- Counter setComputePipeline

  Counter for compute pipeline bindings.

- Counter setTexture
  Counter for texture bindings.
- Counter setSampler
  Counter for sampler bindings.
- Counter setRenderTarget
  Counter for render target bindings.
- Counter drawCalls
  Counter for draw calls.
- Counter dispatchComputeCalls
  Counter for dispatch compute calls.
- Counter renderedPoints
  Counter for rendered point primitives.
- Counter renderedLines
  Counter for rendered line primitives.
- Counter renderedTriangles
  Counter for rendered triangle primitives.
- Counter renderedPatches
  Counter for rendered patch primitives.

9.39.1 Detailed Description

Rendering profiler model class.

Remarks

This can be used to profile the renderer draw calls and buffer updates.

9.39.2 Member Function Documentation

9.39.2.1 void LLGL::RenderingProfiler::RecordDrawCall (const PrimitiveTopology *topology*, Counter::ValueType *numVertices*)

9.39.2.2 void LLGL::RenderingProfiler::RecordDrawCall (const PrimitiveTopology *topology*, Counter::ValueType *numVertices*, Counter::ValueType *numInstances*)

9.39.2.3 void LLGL::RenderingProfiler::ResetCounters ( )

Resets all counters.

See also

Counter::Reset

9.39.3 Member Data Documentation

9.39.3.1 Counter LLGL::RenderingProfiler::dispatchComputeCalls

Counter for dispatch compute calls.

See also

CommandBuffer::Dispatch

9.39.3.2 Counter LLGL::RenderingProfiler::drawCalls

Counter for draw calls.

See also

CommandBuffer.Draw
CommandBuffer.DrawIndexed
CommandBuffer.DrawInstanced
CommandBuffer.DrawIndexedInstanced

9.39.3.3 Counter LLGL::RenderingProfiler::mapBuffer

Counter for buffer mappings.

See also

RenderSystem::MapBuffer

9.39.3.4 Counter LLGL::RenderingProfiler::renderedLines

Counter for rendered line primitives.

9.39.3.5 Counter LLGL::RenderingProfiler::renderedPatches

Counter for rendered patch primitives.

9.39.3.6 Counter LLGL::RenderingProfiler::renderedPoints

Counter for rendered point primitives.

9.39.3.7 Counter LLGL::RenderingProfiler::renderedTriangles

Counter for rendered triangle primitives.

9.39.3.8 Counter LLGL::RenderingProfiler::setComputePipeline

Counter for compute pipeline bindings.

See also

CommandBuffer::SetComputePipeline

9.39.3.9 Counter LLGL::RenderingProfiler::setConstantBuffer

Counter for constant buffer bindings.

See also

CommandBuffer::SetConstantBuffer

9.39.3.10 Counter LLGL::RenderingProfiler::setGraphicsPipeline

Counter for graphics pipeline bindings.

See also

CommandBuffer::SetGraphicsPipeline

9.39.3.11 Counter LLGL::RenderingProfiler::setIndexBuffer

Counter for index buffer bindings.

See also

CommandBuffer::SetIndexBuffer

9.39.3.12 Counter LLGL::RenderingProfiler::setRenderTarget

Counter for render target bindings.

See also

CommandBuffer::SetRenderTarget

9.39.3.13 Counter LLGL::RenderingProfiler::setSampler

Counter for sampler bindings.

See also

CommandBuffer::SetSampler

9.39.3.14 Counter LLGL::RenderingProfiler::setStorageBuffer

Counter for storage buffer bindings.

See also

CommandBuffer::SetStorageBuffer

9.39.3.15 Counter LLGL::RenderingProfiler::setStreamOutputBuffer

Counter for stream-output buffer bindings.

See also

CommandBuffer::SetStreamOutputBuffer

9.39.3.16 Counter LLGL::RenderingProfiler::setTexture

Counter for texture bindings.

See also

CommandBuffer::SetTexture

9.39.3.17 Counter LLGL::RenderingProfiler::setVertexBuffer

Counter for vertex buffer bindings.

See also

CommandBuffer::SetVertexBuffer

9.39.3.18 Counter LLGL::RenderingProfiler::writeBuffer

Counter for buffer writings.

See also

RenderSystem::WriteBuffer

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

- RenderingProfiler.h

9.40 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.
- const RendererInfo & GetRendererInfo () const

  Returns basic renderer information.
- const RenderingCaps & GetRenderingCaps () const

  Returns the rendering capabilities.
- 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.
- virtual CommandBuffer * CreateCommandBuffer ()=0

  Creates a new command buffer.
- virtual void Release (CommandBuffer &commandBuffer)=0

  Releases the specified command buffer. After this call, the specified object must no longer be used.
- virtual Buffer * CreateBuffer (const BufferDescriptor &desc, const void *initialData=nullptr)=0

  Creates a new generic hardware buffer.
- virtual BufferArray * CreateBufferArray (unsigned int numBuffers, Buffer *const *bufferArray)=0

  Creates a new buffer array.
- virtual void Release (Buffer &buffer)=0

  Releases the specified buffer object. After this call, the specified object must no longer be used.
- virtual void Release (BufferArray &bufferArray)=0

  Releases the specified buffer array object. After this call, the specified object must no longer be used.
- virtual void WriteBuffer (Buffer &buffer, const void *data, std::size_t dataSize, std::size_t offset)=0

  Updates the data of the specified buffer.
- virtual void * MapBuffer (Buffer &buffer, const BufferCPUAccess access)=0

  Maps the specified buffer from GPU to CPU memory space.
- virtual void UnmapBuffer (Buffer &buffer)=0

  Unmaps the specified buffer.
- virtual Texture * CreateTexture (const TextureDescriptor &textureDesc, const ImageDescriptor *imageDesc=nullptr)=0

  Creates a new texture.
- virtual TextureArray * CreateTextureArray (unsigned int numTextures, Texture *const *textureArray)=0

  Creates a new texture array.
- virtual void Release (Texture &texture)=0

  Releases the specified texture object. After this call, the specified object must no longer be used.
- virtual void Release (TextureArray &textureArray)=0

  Releases the specified texture array object. After this call, the specified object must no longer be used.
- 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 SamplerArray * CreateSamplerArray (unsigned int numSamplers, Sampler *const *samplerArray)=0

  Creates a new sampler array.

- virtual void Release (Sampler &sampler)=0

  Releases the specified Sampler object. After this call, the specified object must no longer be used.

- virtual void Release (SamplerArray &samplerArray)=0

  Releases the specified sampler array object. After this call, the specified object must no longer be used.

- virtual RenderTarget * CreateRenderTarget (const RenderTargetDescriptor &desc)=0

  Creates a new RenderTarget object.

- 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 object of the specified type.

- virtual ShaderProgram * CreateShaderProgram ()=0

  Creates a new and empty shader program.

- virtual void Release (Shader &shader)=0

  Releases the specified Shader object. After this call, the specified object must no longer be used.

- virtual void Release (ShaderProgram &shaderProgram)=0

  Releases the specified ShaderProgram object. After this call, the specified object must no longer be used.

- 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

  Releases the specified GraphicsPipeline object. After this call, the specified object must no longer be used.

- virtual void Release (ComputePipeline &computePipeline)=0

  Releases the specified ComputePipeline object. After this call, the specified object must no longer be used.

- virtual Query * CreateQuery (const QueryDescriptor &desc)=0

  Creates a new query.

- virtual void Release (Query &query)=0

  Releases the specified Query object. After this call, the specified object must no longer be used.

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", "Direct3D11", "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
- void SetRendererInfo (const RendererInfo &info)

  Sets the renderer information.
- void SetRenderingCaps (const RenderingCaps &caps)

  Sets the rendering capabilities.
- std::vector< ColorRGBAub > GetDefaultTextureImageRGBAub (int numPixels) const

  Creates an RGBA unsigned-byte image buffer for the specified number of pixels.
- void AssertCreateBuffer (const BufferDescriptor &desc)

  Validates the specified buffer descriptor to be used for buffer creation.
- void AssertCreateBufferArray (unsigned int numBuffers, Buffer *const *bufferArray)

  Validates the specified arguments to be used for buffer array creation.
- void AssertCreateTextureArray (unsigned int numTextures, Texture *const *textureArray)

  Validates the specified arguments to be used for texture array creation.
- void AssertCreateSamplerArray (unsigned int numSamplers, Sampler *const *samplerArray)

  Validates the specified arguments to be used for sampler array creation.

9.40.1 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
LLGL::BufferDescriptor bufferDesc;
//fill descriptor ...
auto vertexBuffer = renderSystem->CreateBuffer(*buffer, bufferDesc, initialData);

// Modify data
renderSystem->WriteBuffer(*buffer, modificationData, ...);

// Release object
renderSystem->Release(*buffer);
```

9.40.2 Constructor & Destructor Documentation

9.40.2.1 LLGL::RenderSystem::RenderSystem (const RenderSystem &) [delete]

9.40.2.2 virtual LLGL::RenderSystem::~RenderSystem ( ) [virtual]

9.40.2.3 LLGL::RenderSystem::RenderSystem ( ) [protected], [default]

9.40.3 Member Function Documentation

9.40.3.1 void LLGL::RenderSystem::AssertCreateBuffer (const BufferDescriptor & *desc*) [protected]

Validates the specified buffer descriptor to be used for buffer creation.

9.40.3.2 void LLGL::RenderSystem::AssertCreateBufferArray (unsigned int *numBuffers*, Buffer *const * *bufferArray*) [protected]

Validates the specified arguments to be used for buffer array creation.

9.40.3.3 void LLGL::RenderSystem::AssertCreateSamplerArray (unsigned int *numSamplers*, Sampler *const * *samplerArray*) [protected]

Validates the specified arguments to be used for sampler array creation.

9.40.3.4 void LLGL::RenderSystem::AssertCreateTextureArray (unsigned int *numTextures*, Texture *const * *textureArray*) [protected]

Validates the specified arguments to be used for texture array creation.

9.40.3.5 virtual Buffer* LLGL::RenderSystem::CreateBuffer (const BufferDescriptor & *desc*, const void * *initialData* = nullptr) [pure virtual]

Creates a new generic hardware buffer.

Parameters

in	*desc*	Specifies the vertex buffer descriptor.
in	*initialData*	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 "WriteBuffer" function before it is used for drawing operations. By default null.

See also

WriteBuffer

9.40.3.6 virtual BufferArray* LLGL::RenderSystem::CreateBufferArray (unsigned int *numBuffers*, Buffer *const * *bufferArray*) [pure virtual]

Creates a new buffer array.

Parameters

in	*numBuffers*	Specifies the number of buffers in the array. This must be greater than 0.
in	*bufferArray*	Pointer to an array of Buffer object pointers. This must not be null.

Remarks

This array can only contain buffers which are all from the same type, like an array of vertex buffers for instance. The buffers inside this array must persist as long as this buffer array is used, and the individual buffers are still required to read and write its data from and to the GPU.

Exceptions

std::invalid_argument	If 'numBuffers' is 0, if 'bufferArray' is null, if any of the pointers in the array are null, if not all buffers have the same type, or if the buffer array type is not one of these: BufferType::Vertex, BufferType::Constant, BufferType::Storage, or BufferType::StreamOutput.

9.40.3.7 virtual CommandBuffer* LLGL::RenderSystem::CreateCommandBuffer ( ) [pure virtual]

Creates a new command buffer.

Remarks

Some render systems only support a single command buffer, such as OpenGL and Direct3D 11.

9.40.3.8 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

9.40.3.9 virtual GraphicsPipeline* LLGL::RenderSystem::CreateGraphicsPipeline (const GraphicsPipelineDescriptor & *desc*) [pure virtual]

Creates a new and initialized graphics pipeline state object.

Parameters

in	*desc*	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

9.40.3.10 virtual Query* LLGL::RenderSystem::CreateQuery (const QueryDescriptor & *desc*) [pure virtual]

Creates a new query.

9.40.3.11 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.

9.40.3.12 virtual RenderTarget* LLGL::RenderSystem::CreateRenderTarget (const RenderTargetDescriptor & *desc*) [pure virtual]

Creates a new RenderTarget object.

Exceptions

std::runtime_error	If the renderer does not support RenderTarget objects (e.g. if OpenGL 2.1 or lower is used).

9.40.3.13 virtual Sampler* LLGL::RenderSystem::CreateSampler (const SamplerDescriptor & *desc*) [pure virtual]

Creates a new Sampler object.

Exceptions

std::runtime_error	If the renderer does not support Sampler objects (e.g. if OpenGL 3.1 or lower is used).

See also

RenderContext::QueryRenderingCaps

9.40.3.14 virtual SamplerArray* LLGL::RenderSystem::CreateSamplerArray (unsigned int *numSamplers*, Sampler *const * *samplerArray*) [pure virtual]

Creates a new sampler array.

Parameters

in	*numSamplers*	Specifies the number of samplers in the array. This must be greater than 0.
in	*samplerArray*	Pointer to an array of Sampler object pointers. This must not be null.

Exceptions

std::invalid_argument	If 'numSamplers' is 0, if 'samplerArray' is null, or if any of the pointers in the array are null.

9.40.3.15 virtual Shader* LLGL::RenderSystem::CreateShader (const ShaderType *type*) [pure virtual]

Creates a new and empty Shader object of the specified type.

Parameters

in	*type*	Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like.

See also

Shader

9.40.3.16 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

9.40.3.17 virtual Texture* LLGL::RenderSystem::CreateTexture (const TextureDescriptor & *textureDesc,* const ImageDescriptor * *imageDesc* = nullptr) [pure virtual]

Creates a new texture.

Parameters

in	*textureDesc*	Specifies the texture descriptor.
in	*imageDesc*	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. This parameter will be ignored if the texture type is a multi-sampled texture (i.e. TextureType::Texture2DMS or TextureType::Texture2DMSArray).

See also

WriteTexture
RenderSystemConfiguration::defaultImageColor

9.40.3.18 virtual TextureArray* LLGL::RenderSystem::CreateTextureArray (unsigned int *numTextures,* Texture *const * *textureArray*) [pure virtual]

Creates a new texture array.

Parameters

in	*numTextures*	Specifies the number of textures in the array. This must be greater than 0.
in	*textureArray*	Pointer to an array of Texture object pointers. This must not be null.

Remarks

This texture array is not an "array texture" (like TextureType::Texture2DArray for instance). It is just a container of multiple texture objects, which can be used to bind several hardware textures at once, to improve performance.

Exceptions

std::invalid_argument	If 'numTextures' is 0, if 'textureArray' is null, or if any of the pointers in the array are null.

9.40.3.19 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", "Direct3D11", "Direct3D12" }, but on MacOS it might be only { "OpenGL" }).

9.40.3.20 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

9.40.3.21 const RenderSystemConfiguration& LLGL::RenderSystem::GetConfiguration ( ) const [inline]

Returns the basic configuration.

See also

SetConfiguration

9.40.3.22 std::vector<ColorRGBAub> LLGL::RenderSystem::GetDefaultTextureImageRGBAub (int *numPixels*) const [protected]

Creates an RGBA unsigned-byte image buffer for the specified number of pixels.

9.40.3.23 const std::string& LLGL::RenderSystem::GetName ( ) const [inline]

Returns the name of this render system.

9.40.3.24 const RendererInfo& LLGL::RenderSystem::GetRendererInfo ( ) const [inline]

Returns basic renderer information.

Remarks

The validity of these information is only guaranteed if this function is called after a valid render context has been created. Otherwise the behavior is undefined!

9.40.3.25 const RenderingCaps& LLGL::RenderSystem::GetRenderingCaps ( ) const [inline]

Returns the rendering capabilities.

Remarks

The validity of these information is only guaranteed if this function is called after a valid render context has been created. Otherwise the behavior is undefined!

9.40.3.26 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	*moduleName*	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	*profiler*	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	*debugger*	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 existance 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), and this type can only refer to a std::shared_ptr.

Exceptions

std::runtime_error	If loading the render system from the specified module failed.
std::runtime_error	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!)

9.40.3.27 virtual void* LLGL::RenderSystem::MapBuffer (Buffer & *buffer,* const BufferCPUAccess *access*) [pure virtual]

Maps the specified buffer from GPU to CPU memory space.

Parameters

in	*buffer*	Specifies the buffer which is to be mapped.
in	*access*	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.

See also

UnmapBuffer

9.40.3.28 RenderSystem& LLGL::RenderSystem::operator= (const RenderSystem &) [delete]

9.40.3.29 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

9.40.3.30 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	*texture*	Specifies the texture object to read from.
in	*mipLevel*	Specifies the MIP-level from which to read the image data.
in	*imageFormat*	Specifies the output image format.
in	*dataType*	Specifies the output data type.
out	*buffer*	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

9.40.3.31 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.

9.40.3.32 virtual void LLGL::RenderSystem::Release (CommandBuffer & *commandBuffer*) [pure virtual]

Releases the specified command buffer. After this call, the specified object must no longer be used.

9.40.3.33 virtual void LLGL::RenderSystem::Release (Buffer & *buffer*) [pure virtual]

Releases the specified buffer object. After this call, the specified object must no longer be used.

9.40.3.34 virtual void LLGL::RenderSystem::Release (BufferArray & *bufferArray*) [pure virtual]

Releases the specified buffer array object. After this call, the specified object must no longer be used.

9.40.3.35 virtual void LLGL::RenderSystem::Release (Texture & *texture*) [pure virtual]

Releases the specified texture object. After this call, the specified object must no longer be used.

9.40.3.36 virtual void LLGL::RenderSystem::Release (TextureArray & *textureArray*) [pure virtual]

Releases the specified texture array object. After this call, the specified object must no longer be used.

9.40.3.37 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.

9.40.3.38 virtual void LLGL::RenderSystem::Release (SamplerArray & *samplerArray*) [pure virtual]

Releases the specified sampler array object. After this call, the specified object must no longer be used.

9.40.3.39 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.

9.40.3.40 virtual void LLGL::RenderSystem::Release (Shader & *shader*) [pure virtual]

Releases the specified Shader object. After this call, the specified object must no longer be used.

9.40.3.41 virtual void LLGL::RenderSystem::Release (ShaderProgram & *shaderProgram*) [pure virtual]

Releases the specified ShaderProgram object. After this call, the specified object must no longer be used.

9.40.3.42 virtual void LLGL::RenderSystem::Release (GraphicsPipeline & *graphicsPipeline*) [pure virtual]

Releases the specified GraphicsPipeline object. After this call, the specified object must no longer be used.

9.40.3.43 virtual void LLGL::RenderSystem::Release (ComputePipeline & *computePipeline*) [pure virtual]

Releases the specified ComputePipeline object. After this call, the specified object must no longer be used.

9.40.3.44 virtual void LLGL::RenderSystem::Release (Query & *query*) [pure virtual]

Releases the specified Query object. After this call, the specified object must no longer be used.

9.40.3.45 virtual void LLGL::RenderSystem::SetConfiguration (const RenderSystemConfiguration & *config*) [virtual]

Sets the basic configuration.

Remarks

This can be used to change the behavior of default initializion of textures for instance.

See also

RenderSystemConfiguration

9.40.3.46 void LLGL::RenderSystem::SetRendererInfo (const RendererInfo & *info*) [protected]

Sets the renderer information.

9.40.3.47 void LLGL::RenderSystem::SetRenderingCaps (const RenderingCaps & *caps*) [protected]

Sets the rendering capabilities.

9.40.3.48 virtual void LLGL::RenderSystem::UnmapBuffer (Buffer & *buffer*) [pure virtual]

Unmaps the specified buffer.

See also

MapBuffer

9.40.3.49 virtual void LLGL::RenderSystem::WriteBuffer (Buffer & *buffer,* const void * *data,* std::size_t *dataSize,* std::size_t *offset*) [pure virtual]

Updates the data of the specified buffer.

Parameters

in	*buffer*	Specifies the buffer whose data is to be updated.
in	*data*	Raw pointer to the data with which the buffer is to be updated. This must not be null!
in	*dataSize*	Specifies the size (in bytes) of the data block which is to be updated. This must be less then or equal to the size of the buffer.
in	*offset*	Specifies the offset (in bytes) at which the 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 buffer.

9.40.3.50 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	*texture*	Specifies the texture whose data is to be updated.
in	*subTextureDesc*	Specifies the sub-texture descriptor.
in	*imageDesc*	Specifies the image data descriptor. Its "data" member must not be null!

Remarks

This function can only be used for non-multi-sample textures (i.e. from types other than TextureType::Texture2DMS and TextureType::Texture2DMSArray),

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

- RenderSystem.h

9.41 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.

9.41.1 Detailed Description

Render system configuration structure.

9.41.2 Member Data Documentation

9.41.2.1 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.

9.41.2.2 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

9.42 LLGL::RenderTarget Class Reference

Render target interface.

`#include <RenderTarget.h>`

Public Member Functions

- virtual ~RenderTarget ()
- virtual void AttachDepthBuffer (const Gs::Vector2ui &size)=0

  Attaches an internal depth buffer to this render target.
- virtual void AttachStencilBuffer (const Gs::Vector2ui &size)=0

  Attaches an internal stencil buffer to this render target.
- virtual void AttachDepthStencilBuffer (const Gs::Vector2ui &size)=0

  Attaches an internal depth-stencil buffer to this render target.
- virtual void AttachTexture (Texture &texture, const RenderTargetAttachmentDescriptor &attachmentDesc)=0

  Attaches the specified texture to this render target.
- virtual void DetachAll ()=0

  Detaches all textures and depth-stencil buffers from this render target.
- const Gs::Vector2ui & GetResolution () const

  Returns the frame buffer resolution.

Protected Member Functions

- void ApplyResolution (const Gs::Vector2ui &resolution)
- void ApplyMipResolution (Texture &texture, unsigned int mipLevel)
- void ResetResolution ()

9.42.1 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.

9.42.2 Constructor & Destructor Documentation

9.42.2.1 virtual LLGL::RenderTarget::~RenderTarget ( ) [virtual]

9.42.3 Member Function Documentation

9.42.3.1 void LLGL::RenderTarget::ApplyMipResolution (Texture & *texture,* unsigned int *mipLevel*) [protected]

9.42.3.2 void LLGL::RenderTarget::ApplyResolution (const Gs::Vector2ui & *resolution*) [protected]

9.42.3.3 virtual void LLGL::RenderTarget::AttachDepthBuffer (const Gs::Vector2ui & *size*) [pure virtual]

Attaches an internal depth buffer to this render target.

Parameters

in	*size*	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

9.42.3.4 virtual void LLGL::RenderTarget::AttachDepthStencilBuffer (const Gs::Vector2ui & *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

9.42.3.5 virtual void LLGL::RenderTarget::AttachStencilBuffer (const Gs::Vector2ui & *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

9.42.3.6 virtual void LLGL::RenderTarget::AttachTexture (Texture & *texture,* const RenderTargetAttachmentDescriptor & *attachmentDesc*) [pure virtual]

Attaches the specified texture to this render target.

Parameters

in	*attachmnetDesc*	Specifies the attachment descriptor. Unused members will be ignored, e.g. the 'layer' member is ignored when a non-array texture is passed.

Note

A mixed attachment of multi-sample and non-multi-sample textures to a render-target is currently only supported with: Direct3D 11.

9.42.3.7 virtual void LLGL::RenderTarget::DetachAll ( ) [pure virtual]

Detaches all textures and depth-stencil buffers from this render target.

9.42.3.8 const Gs::Vector2ui& 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.

9.42.3.9 void LLGL::RenderTarget::ResetResolution ( ) [protected]

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

- RenderTarget.h

9.43 LLGL::RenderTargetAttachmentDescriptor Struct Reference

Render target attachment descriptor structure.

`#include <RenderTarget.h>`

Public Attributes

- unsigned int mipLevel = 0

  Specifies the MIP-map level which is to be attached to a render target.
- unsigned int layer = 0

  Array texture layer.
- AxisDirection cubeFace = AxisDirection::XPos

  Cube texture face.

9.43.1 Detailed Description

Render target attachment descriptor structure.

9.43.2 Member Data Documentation

9.43.2.1 AxisDirection LLGL::RenderTargetAttachmentDescriptor::cubeFace = AxisDirection::XPos

Cube texture face.

Remarks

This is only used for cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray).

9.43.2.2 unsigned int LLGL::RenderTargetAttachmentDescriptor::layer = 0

Array texture layer.

Remarks

This is only used for array textures (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray).

9.43.2.3 unsigned int LLGL::RenderTargetAttachmentDescriptor::mipLevel = 0

Specifies the MIP-map level which is to be attached to a render target.

Remarks

This is only used for non-multi-sample textures. All multi-sample textures will always use the first MIP-map level (i.e. TextureType::Texture2DMS and TextureType::Texture2DMSArray).

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

- RenderTarget.h

9.44 LLGL::RenderTargetDescriptor Struct Reference

Render target descriptor structure.

`#include <RenderTarget.h>`

Public Attributes

- MultiSamplingDescriptor multiSampling

  Sampling descriptor.
- bool customMultiSampling = false

  Specifies whether custom multi-sampling is used or not. By default false.

9.44.1 Detailed Description

Render target descriptor structure.

9.44.2 Member Data Documentation

9.44.2.1 bool LLGL::RenderTargetDescriptor::customMultiSampling = false

Specifies whether custom multi-sampling is used or not. By default false.

Remarks

If this is true, only multi-sampled textures can be attached to a render-target, i.e. textures of the following types: Texture2DMS, Texture2DMSArray. If this is false, only non-multi-sampled textures can be attached to a render-target. This field will be ignored if multi-sampling is disabled.

9.44.2.2 MultiSamplingDescriptor LLGL::RenderTargetDescriptor::multiSampling

Sampling descriptor.

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

- RenderTarget.h

9.45 LLGL::Sampler Class Reference

Sampler interface.

`#include <Sampler.h>`

Public Member Functions

- Sampler (const Sampler &)=delete
- Sampler & operator= (const Sampler &)=delete
- virtual ~Sampler ()

Protected Member Functions

- Sampler ()=default

9.45.1 Detailed Description

Sampler interface.

9.45.2 Constructor & Destructor Documentation

9.45.2.1 LLGL::Sampler::Sampler (const Sampler &) [delete]

9.45.2.2 virtual LLGL::Sampler::~Sampler ( ) [inline], [virtual]

9.45.2.3 LLGL::Sampler::Sampler ( ) [protected], [default]

9.45.3 Member Function Documentation

9.45.3.1 Sampler& LLGL::Sampler::operator= (const Sampler &) [delete]

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

- Sampler.h

9.46 LLGL::SamplerArray Class Reference

Sampler array interface.

`#include <SamplerArray.h>`

Public Member Functions

- SamplerArray (const SamplerArray &)=delete
- SamplerArray & operator= (const SamplerArray &)=delete
- virtual ~SamplerArray ()

Protected Member Functions

- SamplerArray ()=default

9.46.1 Detailed Description

Sampler array interface.

9.46.2 Constructor & Destructor Documentation

9.46.2.1 LLGL::SamplerArray::SamplerArray (const SamplerArray &) [delete]

9.46.2.2 virtual LLGL::SamplerArray::~SamplerArray ( ) [inline], [virtual]

9.46.2.3 LLGL::SamplerArray::SamplerArray ( ) [protected], [default]

9.46.3 Member Function Documentation

9.46.3.1 SamplerArray& LLGL::SamplerArray::operator= (const SamplerArray &) [delete]

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

- SamplerArray.h

9.47 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).

9.47.1 Detailed Description

Texture sampler descriptor structure.

9.47.2 Member Data Documentation

9.47.2.1 ColorRGBAf LLGL::SamplerDescriptor::borderColor = { 0.0f, 0.0f, 0.0f, 0.0f }

Border color. By default black (0, 0, 0, 0).

9.47.2.2 CompareOp LLGL::SamplerDescriptor::compareOp = CompareOp::Less

Compare operation for depth textures. By default CompareOp::Less.

9.47.2.3 bool LLGL::SamplerDescriptor::depthCompare = false

Specifies whether the compare operation for depth textures is to be used or not. By default false.

9.47.2.4 TextureFilter LLGL::SamplerDescriptor::magFilter = TextureFilter::Linear

Magnification filter. By default TextureFilter::Linear.

9.47.2.5 unsigned int LLGL::SamplerDescriptor::maxAnisotropy = 1

Maximal anisotropy in the range [1, 16].

9.47.2.6 float LLGL::SamplerDescriptor::maxLOD = 1000.0f

Upper end of the MIP-map range. Must be greater than or equal to "minLOD". By default 1000.

9.47.2.7 TextureFilter LLGL::SamplerDescriptor::minFilter = TextureFilter::Linear

Minification filter. By default TextureFilter::Linear.

9.47.2.8 float LLGL::SamplerDescriptor::minLOD = 0.0f

Lower end of the MIP-map range. By default 0.

9.47.2.9 TextureFilter LLGL::SamplerDescriptor::mipMapFilter = TextureFilter::Linear

MIP-mapping filter. By default TextureFilter::Linear.

9.47.2.10 float LLGL::SamplerDescriptor::mipMapLODBias = 0.0f

MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.

9.47.2.11 bool LLGL::SamplerDescriptor::mipMapping = true

Specifies whether MIP-maps are used or not. By default true.

9.47.2.12 TextureWrap LLGL::SamplerDescriptor::textureWrapU = TextureWrap::Repeat

Texture coordinate wrap mode in U direction. By default TextureWrap::Repeat.

9.47.2.13 TextureWrap LLGL::SamplerDescriptor::textureWrapV = TextureWrap::Repeat

Texture coordinate wrap mode in V direction. By default TextureWrap::Repeat.

9.47.2.14 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

9.48 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

9.48.1 Detailed Description

Scissor dimensions.

Remarks

A scissor is in screen coordinates where the origin is in the left-top corner.

9.48.2 Constructor & Destructor Documentation

9.48.2.1 LLGL::Scissor::Scissor ( ) [default]

9.48.2.2 LLGL::Scissor::Scissor (const Scissor &) [default]

9.48.2.3 LLGL::Scissor::Scissor (int *x*, int *y*, int *width*, int *height*) [inline]

9.48.3 Member Data Documentation

9.48.3.1 int LLGL::Scissor::height = 0

9.48.3.2 int LLGL::Scissor::width = 0

9.48.3.3 int LLGL::Scissor::x = 0

9.48.3.4 int LLGL::Scissor::y = 0

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

- RenderContextFlags.h

9.49 LLGL::Shader Class Reference

Shader interface.

`#include <Shader.h>`

Public Member Functions

- Shader (const Shader &)=delete
- Shader & operator= (const Shader &)=delete
- 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)

9.49.1 Detailed Description

Shader interface.

9.49.2 Constructor & Destructor Documentation

9.49.2.1 LLGL::Shader::Shader (const Shader &) [delete]

9.49.2.2 virtual LLGL::Shader::~Shader ( ) [virtual]

9.49.2.3 LLGL::Shader::Shader (const ShaderType *type*) [protected]

9.49.3 Member Function Documentation

9.49.3.1 virtual bool LLGL::Shader::Compile (const ShaderSource & *shaderSource*) [pure virtual]

Compiles the specified shader source.

Parameters

in	*shaderSource*	Specifies the shader source code.

Returns

True on success, otherwise "QueryInfoLog" can be used to query the reason for failure.

See also

QueryInfoLog

9.49.3.2 virtual std::string LLGL::Shader::Disassemble (int *flags* = 0) [virtual]

Disassembles the previously compiled shader byte code.

Parameters

in	*flags*	Specifies optional disassemble flags. This can be a bitwise OR combination of the 'ShaderDisassembleFlags' 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).

9.49.3.3 ShaderType LLGL::Shader::GetType ( ) const [inline]

Returns the type of this shader.

9.49.3.4 Shader& LLGL::Shader::operator= (const Shader &) [delete]

9.49.3.5 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

9.50 LLGL::ShaderCompileFlags Struct Reference

Shader compilation flags enumeration.

`#include <ShaderFlags.h>`

Public Types

- enum {
  Debug = (1 << 0), O1 = (1 << 1), O2 = (1 << 2), O3 = (1 << 3),
  WarnError = (1 << 4) }

9.50.1 Detailed Description

Shader compilation flags enumeration.

9.50.2 Member Enumeration Documentation

9.50.2.1 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:

- ShaderFlags.h

9.51 LLGL::ShaderDisassembleFlags Struct Reference

Shader disassemble flags enumeration.

`#include <ShaderFlags.h>`

Public Types

- enum { InstructionOnly = (1 << 0) }

9.51.1 Detailed Description

Shader disassemble flags enumeration.

9.51.2 Member Enumeration Documentation

9.51.2.1 anonymous enum

Enumerator

InstructionOnly Show only instructions in disassembly output.

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

- ShaderFlags.h

9.52 LLGL::ShaderProgram Class Reference

Shader program interface.

`#include <ShaderProgram.h>`

Public Member Functions

- ShaderProgram (const ShaderProgram &)=delete
- ShaderProgram & operator= (const ShaderProgram &)=delete
- virtual ~ShaderProgram ()
- virtual void AttachShader (Shader &shader)=0

  Attaches the specified shader to this shader program.
- virtual void DetachAll ()=0

  Detaches all shaders from 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< StreamOutputAttribute > QueryStreamOutputAttributes () const =0

  Returns a list of stream-output attributes, which describes all stream-output 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 BuildInputLayout (const VertexFormat &vertexFormat)=0

  Builds the input layout with the specified vertex format for 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.

Protected Member Functions

- ShaderProgram ()=default

9.52.1 Detailed Description

Shader program interface.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 LLGL::ShaderProgram::ShaderProgram (const ShaderProgram &) [delete]

9.52.2.2 virtual LLGL::ShaderProgram::~ShaderProgram ( ) [inline], [virtual]

9.52.2.3 LLGL::ShaderProgram::ShaderProgram ( ) [protected], [default]

9.52.3 Member Function Documentation

9.52.3.1 virtual void LLGL::ShaderProgram::AttachShader (Shader & *shader*) [pure virtual]

Attaches the specified shader to this shader program.

Parameters

in	*shader*	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

std::invalid_argument	If a shader is attached to this shader program, which is not allowed in the current state. This will happend if a different shader of the same type has already been attached to this shader program for instance.

See also

Shader::GetType

9.52.3.2 virtual void LLGL::ShaderProgram::BindConstantBuffer (const std::string & *name*, unsigned int *bindingIndex*) [pure virtual]

Binds the specified constant buffer to this shader.

Parameters

in	*name*	Specifies the name of the constant buffer within this shader.
in	*bindingIndex*	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

9.52.3.3 virtual void LLGL::ShaderProgram::BindStorageBuffer (const std::string & *name*, unsigned int *bindingIndex*) [pure virtual]

Binds the specified storage buffer to this shader.

Parameters

in	*name*	Specifies the name of the storage buffer within this shader.
in	*bindingIndex*	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

9.52.3.4 virtual void LLGL::ShaderProgram::BuildInputLayout (const VertexFormat & *vertexFormat*) [pure virtual]

Builds the input layout with the specified vertex format for this shader program.

Parameters

in	*vertexFormat*	Specifies the input vertex format.

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 linkage!

See also

AttachShader(VertexShader&)
Shader::Compile
LinkShaders

Exceptions

std::invalid_argument	If the name of an vertex attribute is invalid or the maximal number of available vertex attributes is exceeded.

9.52.3.5 virtual void LLGL::ShaderProgram::DetachAll ( ) [pure virtual]

Detaches all shaders from this shader program.

Remarks

After this call, the link status will be invalid, and the shader program must be linked again.

See also

LinkShaders

9.52.3.6 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

9.52.3.7 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 supported with: OpenGL.

See also

UnlockShaderUniform

9.52.3.8 ShaderProgram& LLGL::ShaderProgram::operator= (const ShaderProgram &) [delete]

9.52.3.9 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".

9.52.3.10 virtual std::string LLGL::ShaderProgram::QueryInfoLog ( ) [pure virtual]

Returns the information log after the shader linkage.

9.52.3.11 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".

9.52.3.12 virtual std::vector<StreamOutputAttribute> LLGL::ShaderProgram::QueryStreamOutputAttributes ( ) const [pure virtual]

Returns a list of stream-output attributes, which describes all stream-output attributes within this shader program.

9.52.3.13 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+.

9.52.3.14 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.

9.52.3.15 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

9.53 LLGL::ShaderSource Struct Reference

Shader source code structure.

`#include <ShaderFlags.h>`

Classes

- struct SourceHLSL
  Additional descripor for HLSL shader source.
- struct StreamOutput
  Additional descriptor for stream outputs.

Public Member Functions

- ShaderSource (const std::string &sourceCode)
  Constructor with shader source code for GLSL.
- ShaderSource (std::string &&sourceCode)
  Constructor with shader source code for GLSL.
- ShaderSource (const std::string &sourceCode, const std::string &entryPoint, const std::string &target, long flags=0)
  Constructor with shader source code for HLSL.
- ShaderSource (std::string &&sourceCode, const std::string &entryPoint, const std::string &target, long flags=0)
  Constructor with shader source code for HLSL.

Public Attributes

- std::string sourceCode
  Shader source code string.
- SourceHLSL sourceHLSL
  Additional HLSL shader source descriptor.
- StreamOutput streamOutput
  Optional stream output for a geometry shader (or a vertex shader when used with OpenGL).

9.53.1 Detailed Description

Shader source code structure.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 LLGL::ShaderSource::ShaderSource (const std::string & *sourceCode*) [inline]

Constructor with shader source code for GLSL.

Parameters

in	*sourceCode*	Specifies the shader source code.

Note

Only supported with: OpenGL.

9.53.2.2 LLGL::ShaderSource::ShaderSource (std::string && *sourceCode*) [inline]

Constructor with shader source code for GLSL.

Parameters

in	*sourceCode*	Specifies the shader source code with move semantic.

Note

Only supported with: OpenGL.

9.53.2.3 LLGL::ShaderSource::ShaderSource (const std::string & *sourceCode,* const std::string & *entryPoint,* const std::string & *target,* long *flags* = 0) [inline]

Constructor with shader source code for HLSL.

Parameters

in	*sourceCode*	Specifies the shader source code.
in	*entryPoint*	Specifies the shader entry point.
in	*target*	Specifies the shader version target (see https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx).
in	*flags*	Specifies optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries. By default 0.

See also

ShaderCompileFlags

Note

Only supported with: Direct3D 11, Direct3D 12.

9.53.2.4 LLGL::ShaderSource::ShaderSource (std::string && *sourceCode,* const std::string & *entryPoint,* const std::string & *target,* long *flags* = 0) [inline]

Constructor with shader source code for HLSL.

Parameters

in	*sourceCode*	Specifies the shader source code with move semantic.
in	*entryPoint*	Specifies the shader entry point.
in	*target*	Specifies the shader version target (see https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx).
in	*flags*	Specifies optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries. By default 0.

See also

ShaderCompileFlags

Note

Only supported with: Direct3D 11, Direct3D 12.

9.53.3 Member Data Documentation

9.53.3.1 std::string LLGL::ShaderSource::sourceCode

Shader source code string.

9.53.3.2 SourceHLSL LLGL::ShaderSource::sourceHLSL

Additional HLSL shader source descriptor.

9.53.3.3 StreamOutput LLGL::ShaderSource::streamOutput

Optional stream output for a geometry shader (or a vertex shader when used with OpenGL).

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

- ShaderFlags.h

9.54 LLGL::ShaderStageFlags Struct Reference

Shader stage flags.

```
#include <ShaderFlags.h>
```

Public Types

- enum {
  VertexStage = (1 << 0), TessControlStage = (1 << 1), TessEvaluationStage = (1 << 2), GeometryStage = (1 << 3),
  FragmentStage = (1 << 4), ComputeStage = (1 << 5), ReadOnlyResource = (1 << 6), AllTessStages = (TessControlStage | TessEvaluationStage),
  AllGraphicsStages = (VertexStage | AllTessStages | GeometryStage | FragmentStage), AllStages = (AllGraphicsStages | ComputeStage) }

9.54.1 Detailed Description

Shader stage flags.

Remarks

Specifies which shader stages are affected by a state change, e.g. to 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, Direct3D 12

9.54.2 Member Enumeration Documentation

9.54.2.1 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.

ReadOnlyResource Specifies whether a resource is bound to the shader stages for reading only.

Remarks

This can be used to set the shader-resource-view (SRV) of a storage buffer to the shader stages instead of the unordered-access-view (UAV), which is the default, if the storage buffer has such a UAV.

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:

- ShaderFlags.h

9.55 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

9.55.1 Detailed Description

Shader uniform setter interface.

Remarks

This is only used by the OpenGL render system.

9.55.2 Constructor & Destructor Documentation

9.55.2.1 virtual LLGL::ShaderUniform::~ShaderUniform ( ) [inline], [virtual]

9.55.3 Member Function Documentation

9.55.3.1 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const int *value*) [pure virtual]

9.55.3.2 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector2i & *value*) [pure virtual]

9.55.3.3 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector3i & *value*) [pure virtual]

9.55.3.4 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector4i & *value*) [pure virtual]

9.55.3.5 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const float *value*) [pure virtual]

9.55.3.6 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector2f & *value*) [pure virtual]

9.55.3.7 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector3f & *value*) [pure virtual]

9.55.3.8 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Vector4f & *value*) [pure virtual]

9.55.3.9 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Matrix2f & *value*) [pure virtual]

9.55.3.10 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Matrix3f & *value*) [pure virtual]

9.55.3.11 virtual void LLGL::ShaderUniform::SetUniform (int *location*, const Gs::Matrix4f & *value*) [pure virtual]

9.55.3.12 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const int *value*) [pure virtual]

9.55.3.13 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector2i & *value*) [pure virtual]

9.55.3.14 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector3i & *value*) [pure virtual]

9.55.3.15 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector4i & *value*) [pure virtual]

9.55.3.16 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const float *value*) [pure virtual]

9.55.3.17 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector2f & *value*) [pure virtual]

9.55.3.18 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector3f & *value*) [pure virtual]

9.55.3.19 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Vector4f & *value*) [pure virtual]

9.55.3.20 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Matrix2f & *value*) [pure virtual]

9.55.3.21 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Matrix3f & *value*) [pure virtual]

9.55.3.22 virtual void LLGL::ShaderUniform::SetUniform (const std::string & *name*, const Gs::Matrix4f & *value*) [pure virtual]

9.55.3.23 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const int * *value*, std::size_t *count*) [pure virtual]

9.55.3.24 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector2i * *value*, std::size_t *count*) [pure virtual]

9.55.3.25 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector3i * *value*, std::size_t *count*) [pure virtual]

9.55.3.26 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector4i * *value*, std::size_t *count*) [pure virtual]

9.55.3.27 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const float * *value*, std::size_t *count*) [pure virtual]

9.55.3.28 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector2f * *value*, std::size_t *count*) [pure virtual]

9.55.3.29 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector3f * *value*, std::size_t *count*) [pure virtual]

9.55.3.30 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Vector4f * *value*, std::size_t *count*) [pure virtual]

9.55.3.31 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Matrix2f * *value*, std::size_t *count*) [pure virtual]

9.55.3.32 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Matrix3f * *value*, std::size_t *count*) [pure virtual]

9.55.3.33 virtual void LLGL::ShaderUniform::SetUniformArray (int *location*, const Gs::Matrix4f * *value*, std::size_t *count*) [pure virtual]

9.55.3.34 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const int * *value*, std::size_t *count*) [pure virtual]

9.55.3.35 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector2i * *value*, std::size_t *count*) [pure virtual]

9.55.3.36 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector3i * *value*, std::size_t *count*) [pure virtual]

9.55.3.37 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector4i * *value*, std::size_t *count*) [pure virtual]

9.55.3.38 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const float * *value*, std::size_t *count*) [pure virtual]

9.55.3.39 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector2f * *value*, std::size_t *count*) [pure virtual]

9.55.3.40 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector3f * *value*, std::size_t *count*) [pure virtual]

9.55.3.41 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Vector4f * *value*, std::size_t *count*) [pure virtual]

9.55.3.42 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Matrix2f * *value*, std::size_t *count*) [pure virtual]

9.55.3.43 virtual void LLGL::ShaderUniform::SetUniformArray (const std::string & *name*, const Gs::Matrix3f * *value*, std::size_t *count*) [pure virtual]

9.55.3.44 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

9.56 LLGL::ShaderSource::SourceHLSL Struct Reference

Additional descripor for HLSL shader source.

`#include <ShaderFlags.h>`

Public Attributes

- 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*).*
- long flags

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

9.56.1 Detailed Description

Additional descripor for HLSL shader source.

9.56.2 Member Data Documentation

9.56.2.1 std::string LLGL::ShaderSource::SourceHLSL::entryPoint

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

9.56.2.2 long LLGL::ShaderSource::SourceHLSL::flags

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

9.56.2.3 std::string LLGL::ShaderSource::SourceHLSL::target

Shader version target (see 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:

- ShaderFlags.h

9.57 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.
- LogicOp logicOp

  Specifies the logical pixel operation for drawing operations. By default LogicOp::Keep.
- float lineWidth

  Specifies the width to rasterize lines. By default 0.

9.57.1 Member Data Documentation

9.57.1.1 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.

9.57.1.2 float LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::lineWidth

Specifies the width to rasterize lines. By default 0.

Remarks

If this is 0, the attribute is ignored and the current line width will not be changed.

See also

https://www.opengl.org/sdk/docs/man/html/glLineWidth.xhtml

9.57.1.3 LogicOp LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::logicOp

Specifies the logical pixel operation for drawing operations. By default LogicOp::Keep.

See also

https://www.opengl.org/sdk/docs/man/html/glLogicOp.xhtml

9.57.1.4 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

9.58 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.

9.58.1 Detailed Description

Stencil state descriptor structure.

9.58.2 Member Data Documentation

9.58.2.1 StencilFaceDescriptor LLGL::StencilDescriptor::back

Specifies the back face settings for the stencil test.

9.58.2.2 StencilFaceDescriptor LLGL::StencilDescriptor::front

Specifies the front face settings for the stencil test.

9.58.2.3 bool LLGL::StencilDescriptor::testEnabled = false

Specifies whether the stencil test is enabled or disabled.

Remarks

If no pixel shader is used in the graphics pipeline, the stencil test must be disabled.

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

- GraphicsPipelineFlags.h

9.59 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 readMask = ~0

  Specifies the portion of the depth-stencil buffer for reading stencil data. By default 0xffffffff.
- std::uint32_t writeMask = ~0

  Specifies the portion of the depth-stencil buffer for writing stencil data. By default 0xffffffff.
- std::uint32_t reference = 0

  Specifies the stencil reference value.

9.59.1 Detailed Description

Stencil face descriptor structure.

9.59.2 Member Data Documentation

9.59.2.1 CompareOp LLGL::StencilFaceDescriptor::compareOp = CompareOp::Less

Specifies the stencil compare operation.

9.59.2.2 StencilOp LLGL::StencilFaceDescriptor::depthFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test passes but the depth test fails.

9.59.2.3 StencilOp LLGL::StencilFaceDescriptor::depthPassOp = StencilOp::Keep

Specifies the operation to take when both the stencil test and the depth test pass.

9.59.2.4 std::uint32_t LLGL::StencilFaceDescriptor::readMask = ~0

Specifies the portion of the depth-stencil buffer for reading stencil data. By default 0xffffffff.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (readMask & 0xff) of the read mask value of the front face will be used.

See also

StencilDescriptor::front

9.59.2.5 std::uint32_t LLGL::StencilFaceDescriptor::reference = 0

Specifies the stencil reference value.

Remarks

This value will be used when the stencil operation is StencilOp::Replace.

Note

For Direct3D 11 and Direct3D 12, only the stencil reference value of the front face will be used.

See also

StencilDescriptor::front

9.59.2.6 StencilOp LLGL::StencilFaceDescriptor::stencilFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test fails.

9.59.2.7 std::uint32_t LLGL::StencilFaceDescriptor::writeMask = ~0

Specifies the portion of the depth-stencil buffer for writing stencil data. By default 0xffffffff.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (writeMask & 0xff) of the write mask value of the front face will be used.

See also

StencilDescriptor::front

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

- GraphicsPipelineFlags.h

9.60 LLGL::BufferDescriptor::StorageBufferDescriptor Struct Reference

`#include <BufferFlags.h>`

Public Attributes

- StorageBufferType storageType = StorageBufferType::Buffer

  Specifies the storage buffer type. By defalut StorageBufferType::Buffer.
- VectorType vectorType = VectorType::Float4

  Specifies the vector type of a typed buffer.
- unsigned int stride = 0

  Specifies the stride (in bytes) of each element in a storage buffer.

9.60.1 Member Data Documentation

9.60.1.1 StorageBufferType LLGL::BufferDescriptor::StorageBufferDescriptor::storageType = StorageBufferType::Buffer

Specifies the storage buffer type. By defalut StorageBufferType::Buffer.

Remarks

In OpenGL there are only generic storage buffers (or rather "Shader Storage Buffer Objects").

See also

vectorType

9.60.1.2 unsigned int LLGL::BufferDescriptor::StorageBufferDescriptor::stride = 0

Specifies the stride (in bytes) of each element in a storage buffer.

Remarks

If this value is zero, the behavior of the buffer creation is undefined.

9.60.1.3 VectorType LLGL::BufferDescriptor::StorageBufferDescriptor::vectorType = VectorType::Float4

Specifies the vector type of a typed buffer.

Remarks

This is only used if the storage type is either StorageBufferType::Buffer or StorageBufferType::RWBuffer.

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

- BufferFlags.h

9.61 LLGL::StorageBufferViewDescriptor Struct Reference

Storage buffer shader-view descriptor structure.

`#include <BufferFlags.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.

9.61.1 Detailed Description

Storage buffer shader-view descriptor structure.

Remarks

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

9.61.2 Member Data Documentation

9.61.2.1 unsigned int LLGL::StorageBufferViewDescriptor::index = 0

Index of the storage buffer within the respective shader.

9.61.2.2 std::string LLGL::StorageBufferViewDescriptor::name

Storage buffer name.

9.61.2.3 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:

- BufferFlags.h

9.62 LLGL::ShaderSource::StreamOutput Struct Reference

Additional descriptor for stream outputs.

`#include <ShaderFlags.h>`

Public Attributes

- StreamOutputFormat format
  Stream-output buffer format.

9.62.1 Detailed Description

Additional descriptor for stream outputs.

9.62.2 Member Data Documentation

9.62.2.1 StreamOutputFormat LLGL::ShaderSource::StreamOutput::format

Stream-output buffer format.

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

- ShaderFlags.h

9.63 LLGL::StreamOutputAttribute Struct Reference

Stream-output attribute structure.

`#include <StreamOutputAttribute.h>`

Public Member Functions

- StreamOutputAttribute ()=default
- StreamOutputAttribute (const StreamOutputAttribute &)=default
- StreamOutputAttribute & operator= (const StreamOutputAttribute &)=default

Public Attributes

- std::string name
  Vertex attribute name (for GLSL) or semantic name (for HLSL).
- unsigned int stream = 0
  Zero-based stream number. By default 0.
- unsigned char startComponent = 0
  Start vector component index, which is to be written. Must be 0, 1, 2, or 3. By default 0.
- unsigned char components = 4
  Number of vector components, which are to be written. Must be 1, 2, 3, or 4.
- unsigned int semanticIndex = 0
  Semantic index.
- unsigned char outputSlot = 0
  Stream-output buffer output slot.

9.63.1 Detailed Description

Stream-output attribute structure.

9.63.2 Constructor & Destructor Documentation

9.63.2.1 LLGL::StreamOutputAttribute::StreamOutputAttribute ( ) [default]

9.63.2.2 LLGL::StreamOutputAttribute::StreamOutputAttribute (const StreamOutputAttribute &) [default]

9.63.3 Member Function Documentation

9.63.3.1 StreamOutputAttribute& LLGL::StreamOutputAttribute::operator= (const StreamOutputAttribute &) [default]

9.63.4 Member Data Documentation

9.63.4.1 unsigned char LLGL::StreamOutputAttribute::components = 4

Number of vector components, which are to be written. Must be 1, 2, 3, or 4.

Remarks

The number of components plus the start component index (see 'startComponent') must not be larger than 4.

See also

startComponent

9.63.4.2 std::string LLGL::StreamOutputAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

9.63.4.3 unsigned char LLGL::StreamOutputAttribute::outputSlot = 0

Stream-output buffer output slot.

Remarks

This is used when multiple stream-output buffers are used simultaneously.

9.63.4.4 unsigned int LLGL::StreamOutputAttribute::semanticIndex = 0

Semantic index.

Note

Only supported with: Direct3D 11, Direct3D 12.

9.63.4.5 unsigned char LLGL::StreamOutputAttribute::startComponent = 0

Start vector component index, which is to be written. Must be 0, 1, 2, or 3. By default 0.

9.63.4.6 unsigned int LLGL::StreamOutputAttribute::stream = 0

Zero-based stream number. By default 0.

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

- StreamOutputAttribute.h

9.64 LLGL::StreamOutputFormat Struct Reference

Stream-output format descriptor structure.

`#include <StreamOutputFormat.h>`

Public Member Functions

- void AppendAttribute (const StreamOutputAttribute &attrib)

  Appends the specified stream-output attribute to this stream-output format.
- void AppendAttributes (const StreamOutputFormat &format)

  Append all attributes of the specified stream-output format.

Public Attributes

- std::vector< StreamOutputAttribute > attributes

  Specifies the list of vertex attributes.

9.64.1 Detailed Description

Stream-output format descriptor structure.

Remarks

A vertex format is required to describe how the vertex attributes are supported inside a vertex buffer.

9.64.2 Member Function Documentation

9.64.2.1 void LLGL::StreamOutputFormat::AppendAttribute (const StreamOutputAttribute & *attrib*)

Appends the specified stream-output attribute to this stream-output format.

Parameters

in	*attrib*	Specifies the new attribute which is appended to this stream-output format.

9.64.2.2 void LLGL::StreamOutputFormat::AppendAttributes (const StreamOutputFormat & *format*)

Append all attributes of the specified stream-output format.

Remarks

This can be used to build a stream-output format for stream-output buffer arrays.

9.64.3 Member Data Documentation

9.64.3.1 std::vector<StreamOutputAttribute> LLGL::StreamOutputFormat::attributes

Specifies the list of vertex attributes.

Remarks

Use "AppendAttribute" or "AppendAttributes" to append new attributes.

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

- StreamOutputFormat.h

9.65 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

- unsigned int mipLevel

  MIP-map level for the sub-texture, where 0 is the base texture, and n > 0 is the n-th MIP-map level.
- union {
  Texture1DDescriptor texture1D
  Descriptor for 1D- and 1D-Array textures.
  Texture2DDescriptor texture2D
  Descriptor for 2D- and 2D-Array textures.
  Texture3DDescriptor texture3D
  Descriptor for 3D textures.
  TextureCubeDescriptor textureCube
  Descriptor for Cube- and Cube-Array textures.
  };

9.65.1 Detailed Description

Sub-texture descriptor structure.

Remarks

This is used to write (or partially write) the image data of a texture MIP-map level.

9.65.2 Constructor & Destructor Documentation

9.65.2.1 LLGL::SubTextureDescriptor::SubTextureDescriptor ( ) [inline]

9.65.2.2 LLGL::SubTextureDescriptor::~SubTextureDescriptor ( ) [inline]

9.65.3 Member Data Documentation

9.65.3.1 union { ... }

9.65.3.2 unsigned int LLGL::SubTextureDescriptor::mipLevel

MIP-map level for the sub-texture, where 0 is the base texture, and n > 0 is the n-th MIP-map level.

9.65.3.3 Texture1DDescriptor LLGL::SubTextureDescriptor::texture1D

Descriptor for 1D- and 1D-Array textures.

9.65.3.4 Texture2DDescriptor LLGL::SubTextureDescriptor::texture2D

Descriptor for 2D- and 2D-Array textures.

9.65.3.5 Texture3DDescriptor LLGL::SubTextureDescriptor::texture3D

Descriptor for 3D textures.

9.65.3.6 TextureCubeDescriptor LLGL::SubTextureDescriptor::textureCube

Descriptor for Cube- and Cube-Array textures.

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

- TextureFlags.h

9.66 LLGL::Texture Class Reference

Texture interface.

`#include <Texture.h>`

Public Member Functions

- Texture (const Texture &)=delete
- Texture & operator= (const Texture &)=delete
- virtual ~Texture ()
- TextureType GetType () const

  Returns the type of this texture.
- virtual Gs::Vector3ui QueryMipLevelSize (unsigned int mipLevel) const =0

  Returns the texture size for the specified MIP-level.

Protected Member Functions

- Texture (const TextureType type)

9.66.1 Detailed Description

Texture interface.

9.66.2 Constructor & Destructor Documentation

9.66.2.1 LLGL::Texture::Texture (const Texture &) [delete]

9.66.2.2 virtual LLGL::Texture::~Texture ( ) [virtual]

9.66.2.3 LLGL::Texture::Texture (const TextureType *type*) [protected]

9.66.3 Member Function Documentation

9.66.3.1 TextureType LLGL::Texture::GetType ( ) const [inline]

Returns the type of this texture.

9.66.3.2 Texture& LLGL::Texture::operator= (const Texture &) [delete]

9.66.3.3 virtual Gs::Vector3ui LLGL::Texture::QueryMipLevelSize (unsigned int *mipLevel*) const [pure virtual]

Returns the texture size for the specified MIP-level.

Parameters

in	*mipLevel*	Specifies the MIP-map level to querey 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

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

- Texture.h

9.67 LLGL::SubTextureDescriptor::Texture1DDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int x

  Sub-texture X-axis offset.
- unsigned int layerOffset

  Zero-based layer offset.
- unsigned int width

  Sub-texture width.
- unsigned int layers

  Number of texture array layers.

9.67.1 Member Data Documentation

9.67.1.1 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::layerOffset

Zero-based layer offset.

9.67.1.2 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::layers

Number of texture array layers.

9.67.1.3 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::width

Sub-texture width.

9.67.1.4 unsigned int LLGL::SubTextureDescriptor::Texture1DDescriptor::x

Sub-texture X-axis offset.

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

- TextureFlags.h

9.68 LLGL::TextureDescriptor::Texture1DDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int width
  Texture width.
- unsigned int layers
  Number of texture array layers.

9.68.1 Member Data Documentation

9.68.1.1 unsigned int LLGL::TextureDescriptor::Texture1DDescriptor::layers

Number of texture array layers.

9.68.1.2 unsigned int LLGL::TextureDescriptor::Texture1DDescriptor::width

Texture width.

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

- TextureFlags.h

9.69 LLGL::TextureDescriptor::Texture2DDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int width
  Texture width.
- unsigned int height
  Texture height.
- unsigned int layers
  Number of texture array layers.

9.69.1 Member Data Documentation

9.69.1.1 unsigned int LLGL::TextureDescriptor::Texture2DDescriptor::height

Texture height.

9.69.1.2 unsigned int LLGL::TextureDescriptor::Texture2DDescriptor::layers

Number of texture array layers.

9.69.1.3 unsigned int LLGL::TextureDescriptor::Texture2DDescriptor::width

Texture width.

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

- TextureFlags.h

9.70 LLGL::SubTextureDescriptor::Texture2DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

Public Attributes

- unsigned int x
  Sub-texture X-axis offset.
- unsigned int y
  Sub-texture Y-axis offset.
- unsigned int layerOffset
  Zero-based layer offset.
- unsigned int width
  Sub-texture width.
- unsigned int height
  Sub-texture height.
- unsigned int layers
  Number of texture array layers.

9.70.1 Member Data Documentation

9.70.1.1 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::height

Sub-texture height.

9.70.1.2 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::layerOffset

Zero-based layer offset.

9.70.1.3 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::layers

Number of texture array layers.

9.70.1.4 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::width

Sub-texture width.

9.70.1.5 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::x

Sub-texture X-axis offset.

9.70.1.6 unsigned int LLGL::SubTextureDescriptor::Texture2DDescriptor::y

Sub-texture Y-axis offset.

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

- TextureFlags.h

9.71 LLGL::TextureDescriptor::Texture2DMSDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int width
  Texture width.
- unsigned int height
  Texture height.
- unsigned int layers
  Number of texture array layers.
- unsigned int samples
  Number of samples.
- bool fixedSamples
  Specifies whether the sample locations are fixed or not. By default true.

9.71.1 Member Data Documentation

9.71.1.1 bool LLGL::TextureDescriptor::Texture2DMSDescriptor::fixedSamples

Specifies whether the sample locations are fixed or not. By default true.

Note

Only supported with: OpenGL.

9.71.1.2 unsigned int LLGL::TextureDescriptor::Texture2DMSDescriptor::height

Texture height.

9.71.1.3 unsigned int LLGL::TextureDescriptor::Texture2DMSDescriptor::layers

Number of texture array layers.

9.71.1.4 unsigned int LLGL::TextureDescriptor::Texture2DMSDescriptor::samples

Number of samples.

9.71.1.5 unsigned int LLGL::TextureDescriptor::Texture2DMSDescriptor::width

Texture width.

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

- TextureFlags.h

9.72 LLGL::TextureDescriptor::Texture3DDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int width
  Texture width.
- unsigned int height
  Texture height.
- unsigned int depth
  Texture depth.

9.72.1 Member Data Documentation

9.72.1.1 unsigned int LLGL::TextureDescriptor::Texture3DDescriptor::depth

Texture depth.

9.72.1.2 unsigned int LLGL::TextureDescriptor::Texture3DDescriptor::height

Texture height.

9.72.1.3 unsigned int LLGL::TextureDescriptor::Texture3DDescriptor::width

Texture width.

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

- TextureFlags.h

9.73 LLGL::SubTextureDescriptor::Texture3DDescriptor Struct Reference

```
#include <TextureFlags.h>
```

Public Attributes

- unsigned int x
  Sub-texture X-axis offset.
- unsigned int y
  Sub-texture Y-axis offset.
- unsigned int z
  Sub-texture Z-axis offset.
- unsigned int width
  Sub-texture width.
- unsigned int height
  Sub-texture height.
- unsigned int depth
  Number of texture array layers.

9.73.1 Member Data Documentation

9.73.1.1 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::depth

Number of texture array layers.

9.73.1.2 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::height

Sub-texture height.

9.73.1.3 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::width

Sub-texture width.

9.73.1.4 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::x

Sub-texture X-axis offset.

9.73.1.5 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::y

Sub-texture Y-axis offset.

9.73.1.6 unsigned int LLGL::SubTextureDescriptor::Texture3DDescriptor::z

Sub-texture Z-axis offset.

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

- TextureFlags.h

9.74 LLGL::TextureArray Class Reference

Array of textures interface.

`#include <TextureArray.h>`

Public Member Functions

- TextureArray (const TextureArray &)=delete
- TextureArray & operator= (const TextureArray &)=delete
- virtual ~TextureArray ()

Protected Member Functions

- TextureArray ()=default

9.74.1 Detailed Description

Array of textures interface.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 LLGL::TextureArray::TextureArray (const TextureArray &) [delete]

9.74.2.2 virtual LLGL::TextureArray::~TextureArray ( ) [inline], [virtual]

9.74.2.3 LLGL::TextureArray::TextureArray ( ) [protected], [default]

9.74.3 Member Function Documentation

9.74.3.1 TextureArray& LLGL::TextureArray::operator= (const TextureArray &) [delete]

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

- TextureArray.h

9.75 LLGL::TextureDescriptor::TextureCubeDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int width
  Texture width.
- unsigned int height
  Texture height.
- unsigned int layers
  Number of texture array layers (internally it will be a multiple of 6).

9.75.1 Member Data Documentation

9.75.1.1 unsigned int LLGL::TextureDescriptor::TextureCubeDescriptor::height

Texture height.

9.75.1.2 unsigned int LLGL::TextureDescriptor::TextureCubeDescriptor::layers

Number of texture array layers (internally it will be a multiple of 6).

9.75.1.3 unsigned int LLGL::TextureDescriptor::TextureCubeDescriptor::width

Texture width.

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

- TextureFlags.h

9.76 LLGL::SubTextureDescriptor::TextureCubeDescriptor Struct Reference

`#include <TextureFlags.h>`

Public Attributes

- unsigned int x
  Sub-texture X-axis offset.
- unsigned int y
  Sub-texture Y-axis offset.
- unsigned int layerOffset
  Zero-based layer offset.
- unsigned int width
  Sub-texture width.
- unsigned 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.

9.76.1 Member Data Documentation

9.76.1.1 AxisDirection LLGL::SubTextureDescriptor::TextureCubeDescriptor::cubeFaceOffset

First cube face in the current layer.

9.76.1.2 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.

9.76.1.3 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::height

Sub-texture height.

9.76.1.4 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::layerOffset

Zero-based layer offset.

9.76.1.5 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::width

Sub-texture width.

9.76.1.6 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::x

Sub-texture X-axis offset.

9.76.1.7 unsigned int LLGL::SubTextureDescriptor::TextureCubeDescriptor::y

Sub-texture Y-axis offset.

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

- TextureFlags.h

9.77 LLGL::TextureDescriptor Struct Reference

Texture descriptor structure.

`#include <TextureFlags.h>`

Classes

- struct Texture1DDescriptor
- struct Texture2DDescriptor
- struct Texture2DMSDescriptor
- struct Texture3DDescriptor
- struct TextureCubeDescriptor

Public Member Functions

- TextureDescriptor ()
- ~TextureDescriptor ()

Public Attributes

- TextureType type

  Texture type. By default TextureType::Texture1D.
- TextureFormat format

  Texture hardware format. By default TextureFormat::RGBA.
- union {
  Texture1DDescriptor texture1D
  Descriptor for 1D- and 1D-Array textures.
  Texture2DDescriptor texture2D
  Descriptor for 2D- and 2D-Array textures.
  Texture3DDescriptor texture3D
  Descriptor for 3D textures.
  TextureCubeDescriptor textureCube
  Descriptor for Cube- and Cube-Array textures.
  Texture2DMSDescriptor texture2DMS
  Descriptor for multi-sampled 2D- and 2D-Array textures.
  };

9.77.1 Detailed Description

Texture descriptor structure.

Remarks

This is used to specifiy the dimensions of a texture which is to be created.

9.77.2 Constructor & Destructor Documentation

9.77.2.1 LLGL::TextureDescriptor::TextureDescriptor ( ) [inline]

9.77.2.2 LLGL::TextureDescriptor::~TextureDescriptor ( ) [inline]

9.77.3 Member Data Documentation

9.77.3.1 union { ... }

9.77.3.2 TextureFormat LLGL::TextureDescriptor::format

Texture hardware format. By default TextureFormat::RGBA.

9.77.3.3 Texture1DDescriptor LLGL::TextureDescriptor::texture1D

Descriptor for 1D- and 1D-Array textures.

9.77.3.4 Texture2DDescriptor LLGL::TextureDescriptor::texture2D

Descriptor for 2D- and 2D-Array textures.

9.77.3.5 Texture2DMSDescriptor LLGL::TextureDescriptor::texture2DMS

Descriptor for multi-sampled 2D- and 2D-Array textures.

9.77.3.6 Texture3DDescriptor LLGL::TextureDescriptor::texture3D

Descriptor for 3D textures.

9.77.3.7 TextureCubeDescriptor LLGL::TextureDescriptor::textureCube

Descriptor for Cube- and Cube-Array textures.

9.77.3.8 TextureType LLGL::TextureDescriptor::type

Texture type. By default TextureType::Texture1D.

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

- TextureFlags.h

9.78 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.

9.78.1 Member Typedef Documentation

9.78.1.1 using LLGL::Timer::FrameCount = unsigned long long

9.78.2 Constructor & Destructor Documentation

9.78.2.1 virtual LLGL::Timer::~Timer ( ) [virtual]

9.78.3 Member Function Documentation

9.78.3.1 static std::unique_ptr<Timer> LLGL::Timer::Create ( ) [static]

Creates a platform specific timer object.

9.78.3.2 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

9.78.3.3 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

9.78.3.4 virtual double LLGL::Timer::GetFrequency ( ) const [pure virtual]

Returns the frequency this timer can measure time (e.g. for milliseconds this is 1000.0).

9.78.3.5 void LLGL::Timer::MeasureTime ( )

Measures the time (elapsed time, and frame count) for each frame.

See also

GetDeltaTime
GetFrameCount()

9.78.3.6 void LLGL::Timer::ResetFrameCounter ( )

Restes the frame counter.

See also

GetFrameCount

9.78.3.7 virtual void LLGL::Timer::Start ( ) [pure virtual]

Starts the timer.

9.78.3.8 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

9.79 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

9.79.1 Detailed Description

Shader uniform descriptor structure.

9.79.2 Member Data Documentation

9.79.2.1 int LLGL::UniformDescriptor::location = 0

9.79.2.2 std::string LLGL::UniformDescriptor::name

9.79.2.3 unsigned int LLGL::UniformDescriptor::size = 0

9.79.2.4 UniformType LLGL::UniformDescriptor::type = UniformType::Float

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

- ShaderUniform.h

9.80 LLGL::VertexAttribute Struct Reference

Vertex attribute structure.

`#include <VertexAttribute.h>`

Public Member Functions

- VertexAttribute ()=default
- VertexAttribute (const VertexAttribute &)=default
- VertexAttribute & operator= (const VertexAttribute &)=default
- VertexAttribute (const std::string &name, const VectorType vectorType, unsigned int instanceDivisor=0)

  Constructs a vertex attribute with a specified name (used for GLSL).
- VertexAttribute (const std::string &semanticName, unsigned int semanticIndex, const VectorType vectorType, unsigned int instanceDivisor=0)

  Constructs a vertex attribute with a specified semantic (used for HLSL).
- unsigned int GetSize () const

  Returns the size (in bytes) which is required for this vertex attribute.

Public Attributes

- std::string name

  Vertex attribute name (for GLSL) or semantic name (for HLSL).
- VectorType vectorType = VectorType::Float4

  Vector type of the vertex attribute. By default VectorType::Float4.
- unsigned int instanceDivisor = 0

  Instance data divosor (or instance data step rate).
- bool conversion = false

  Specifies whether non-floating-point data types are to be converted to floating-points. By default false.
- unsigned int offset = 0

  Byte offset within each vertex. By default 0.
- unsigned int semanticIndex = 0

  Semantic index.
- unsigned int inputSlot = 0

  Vertex buffer input slot.

9.80.1 Detailed Description

Vertex attribute structure.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 LLGL::VertexAttribute::VertexAttribute ( ) [default]

9.80.2.2 LLGL::VertexAttribute::VertexAttribute (const VertexAttribute &) [default]

9.80.2.3 LLGL::VertexAttribute::VertexAttribute (const std::string & *name*, const VertexType *vectorType*, unsigned int *instanceDivisor* = 0)

Constructs a vertex attribute with a specified name (used for GLSL).

Parameters

in	*name*	Specifies the attribute name (for GLSL).
in	*vectorType*	Specifies the vector type of the attribute.
in	*instanceDivisor*	Specifies the divisor (or step rate) for instance data. If this is 0, this vertex attribute is considered to be per-vertex. By default 0.

Remarks

This is equivalent to:

```
VertexAttribute(name, 0, dataType, components,
      instanceDivisor);
```

9.80.2.4 LLGL::VertexAttribute::VertexAttribute (const std::string & *semanticName*, unsigned int *semanticIndex*, const VertexType *vectorType*, unsigned int *instanceDivisor* = 0)

Constructs a vertex attribute with a specified semantic (used for HLSL).

Parameters

in	*semanticName*	Specifies the semantic name (for HLSL).
in	*semanticIndex*	Specifies the semantic index (for HLSL).
in	*vectorType*	Specifies the vector type of the attribute.
in	*instanceDivisor*	Specifies the divisor (or step rate) for instance data. If this is 0, this vertex attribute is considered to be per-vertex. By default 0.

Remarks

This is equivalent to:

```
VertexAttribute(name, 0, dataType, components,
      instanceDivisor);
```

9.80.3 Member Function Documentation

9.80.3.1 unsigned int LLGL::VertexAttribute::GetSize ( ) const

Returns the size (in bytes) which is required for this vertex attribute.

Returns

`VectorTypeSize(vectorType)`.

9.80.3.2 VertexAttribute& LLGL::VertexAttribute::operator= (const VertexAttribute &) [default]

9.80.4 Member Data Documentation

9.80.4.1 bool LLGL::VertexAttribute::conversion = false

Specifies whether non-floating-point data types are to be converted to floating-points. By default false.

9.80.4.2 unsigned int LLGL::VertexAttribute::inputSlot = 0

Vertex buffer input slot.

Remarks

This is used when multiple vertex buffers are used simultaneously.

9.80.4.3 unsigned int LLGL::VertexAttribute::instanceDivisor = 0

Instance data divosor (or instance data step rate).

Remarks

If this is 0, this attribute is considered to be stored per vertex. If this is greater than 0, this attribute is considered to be stored per every instanceDivisor's instance.

9.80.4.4 std::string LLGL::VertexAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

9.80.4.5 unsigned int LLGL::VertexAttribute::offset = 0

Byte offset within each vertex. By default 0.

9.80.4.6 unsigned int LLGL::VertexAttribute::semanticIndex = 0

Semantic index.

Note

Only supported with: Direct3D 11, Direct3D 12.

9.80.4.7 VectorType LLGL::VertexAttribute::vectorType = VectorType::Float4

Vector type of the vertex attribute. By default VectorType::Float4.

Remarks

The double types are only supported with OpenGL, i.e. the vector types: VectorType::Double, VectorType::Double2, VectorType::Double3, and VectorType::Double4.

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

- VertexAttribute.h

9.81 LLGL::BufferDescriptor::VertexBufferDescriptor Struct Reference

Vertex buffer descriptor structure.

`#include <BufferFlags.h>`

Public Attributes

- VertexFormat format

  Specifies the vertex format layout.

9.81.1 Detailed Description

Vertex buffer descriptor structure.

9.81.2 Member Data Documentation

9.81.2.1 VertexFormat LLGL::BufferDescriptor::VertexBufferDescriptor::format

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:

- BufferFlags.h

9.82 LLGL::VertexFormat Struct Reference

Vertex format descriptor structure.

`#include <VertexFormat.h>`

Public Member Functions

- void AppendAttribute (const VertexAttribute &attrib, unsigned int offset=OffsetAppend)

  Appends the specified vertex attribute to this vertex format.
- void AppendAttributes (const VertexFormat &format)

  Append all attributes of the specified vertex format.

Public Attributes

- std::vector< VertexAttribute > attributes

  Specifies the list of vertex attributes.
- unsigned int stride = 0

  Specifies the vertex data stride (or format size) which describes the byte offset between consecutive vertices.

Static Public Attributes

- static const unsigned int OffsetAppend = ~0

  Offset value to determine the offset automatically, so that a vertex attribute is appended at the end of a vertex format.

9.82.1 Detailed Description

Vertex format descriptor structure.

Remarks

A vertex format is required to describe how the vertex attributes are supported inside a vertex buffer.

9.82.2 Member Function Documentation

9.82.2.1 void LLGL::VertexFormat::AppendAttribute (const VertexAttribute & *attrib*, unsigned int *offset* = OffsetAppend)

Appends the specified vertex attribute to this vertex format.

Parameters

in	*attrib*	Specifies the new attribute which is appended to this vertex format.
in	*offset*	Specifies the optional offset (in bytes) for this attribute. If this is 'OffsetAppend', the offset is determined by the previous vertex attribute offset plus its size. If there is no previous vertex attribute, the determined offset is 0. By default OffsetAppend.

Remarks

This function will always overwrite the 'offset' and 'inputSlot' members before the attribute is appended to this vertex format. The 'inputSlot' member will be set to the input slot value of the previous vertex attribute and is increased by one, if the new offset of the new vertex attribute is less than the offset plus size of the previous vertex attribute.

Exceptions

std::invalid_argument	If 'attrib.components' is neither 1, 2, 3, nor 4.

See also

VertexAttribute::offset
VertexAttribute::inputSlot

9.82.2.2 void LLGL::VertexFormat::AppendAttributes (const VertexFormat & *format*)

Append all attributes of the specified vertex format.

Remarks

This can be used to build a vertex format for vertex buffer arrays.

9.82.3 Member Data Documentation

9.82.3.1 std::vector<VertexAttribute> LLGL::VertexFormat::attributes

Specifies the list of vertex attributes.

Remarks

Use "AppendAttribute" or "AppendAttributes" to append new attributes.

9.82.3.2 const unsigned int LLGL::VertexFormat::OffsetAppend = ~0 [static]

Offset value to determine the offset automatically, so that a vertex attribute is appended at the end of a vertex format.

See also

AppendAttribute

9.82.3.3 unsigned int LLGL::VertexFormat::stride = 0

Specifies the vertex data stride (or format size) which describes the byte offset between consecutive vertices.

Remarks

This is updated automatically evertime "AppendAttribute" or "AppendAttributes" is called, but it can also modified manually. It is commonly the size of all vertex attributes.

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

- VertexFormat.h

9.83 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.

9.83.1 Detailed Description

Video adapter descriptor structure.

9.83.2 Member Data Documentation

9.83.2.1 std::wstring LLGL::VideoAdapterDescriptor::name

Hardware adapter name (name of the GPU).

9.83.2.2 std::vector<VideoOutput> LLGL::VideoAdapterDescriptor::outputs

Adapter outputs.

9.83.2.3 std::string LLGL::VideoAdapterDescriptor::vendor

Vendor name.

9.83.2.4 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

9.84 LLGL::VideoDisplayMode Struct Reference

Video display mode structure.

`#include <VideoAdapter.h>`

Public Attributes

- unsigned int width = 0
  Display resolution width (in pixels).
- unsigned int height = 0
  Display resolution width (in height).
- unsigned int refreshRate = 0
  Refresh reate (in Hz).

9.84.1 Detailed Description

Video display mode structure.

9.84.2 Member Data Documentation

9.84.2.1 unsigned int LLGL::VideoDisplayMode::height = 0

Display resolution width (in height).

9.84.2.2 unsigned int LLGL::VideoDisplayMode::refreshRate = 0

Refresh reate (in Hz).

9.84.2.3 unsigned int LLGL::VideoDisplayMode::width = 0

Display resolution width (in pixels).

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

- VideoAdapter.h

9.85 LLGL::VideoModeDescriptor Struct Reference

Video mode descriptor structure.

`#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.

9.85.1 Detailed Description

Video mode descriptor structure.

9.85.2 Member Data Documentation

9.85.2.1 int LLGL::VideoModeDescriptor::colorDepth = 32

Color bit depth. Should be 24 or 32. By default 32.

9.85.2.2 bool LLGL::VideoModeDescriptor::fullscreen = false

Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

9.85.2.3 Size LLGL::VideoModeDescriptor::resolution

Screen resolution.

9.85.2.4 SwapChainMode LLGL::VideoModeDescriptor::swapChainMode = SwapChainMode::DoubleBuffering

Swap chain buffering mode.

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

- RenderContextDescriptor.h

9.86 LLGL::VideoOutput Struct Reference

Video output structure.

`#include <VideoAdapter.h>`

Public Attributes

- std::vector< VideoDisplayMode > displayModes
  Video display mode list.

9.86.1 Detailed Description

Video output structure.

9.86.2 Member Data Documentation

9.86.2.1 std::vector<VideoDisplayMode> LLGL::VideoOutput::displayModes

Video display mode list.

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

- VideoAdapter.h

9.87 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.

9.87.1 Detailed Description

Viewport dimensions.

Remarks

A viewport is in screen coordinates where the origin is in the left-top corner.

9.87.2 Constructor & Destructor Documentation

9.87.2.1 LLGL::Viewport::Viewport ( ) [default]

9.87.2.2 LLGL::Viewport::Viewport (const Viewport &) [default]

9.87.2.3 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*) [inline]

9.87.2.4 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*, float *minDepth*, float *maxDepth*) [inline]

9.87.3 Member Data Documentation

9.87.3.1 float LLGL::Viewport::height = 0.0f

Right-bottom height.

9.87.3.2 float LLGL::Viewport::maxDepth = 1.0f

Maximal depth range.

9.87.3.3 float LLGL::Viewport::minDepth = 0.0f

Minimal depth range.

9.87.3.4 float LLGL::Viewport::width = 0.0f

Right-bottom width.

9.87.3.5 float LLGL::Viewport::x = 0.0f

Left-top X coordinate.

9.87.3.6 float LLGL::Viewport::y = 0.0f

Left-top Y coordinate.

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

- RenderContextFlags.h

9.88 LLGL::VsyncDescriptor Struct Reference

Vertical-synchronization (Vsync) descriptor structure.

`#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.

9.88.1 Detailed Description

Vertical-synchronization (Vsync) descriptor structure.

9.88.2 Member Data Documentation

9.88.2.1 bool LLGL::VsyncDescriptor::enabled = false

Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.

9.88.2.2 unsigned int LLGL::VsyncDescriptor::interval = 1

Synchronisation interval. Can be 1, 2, 3, or 4. If Vsync is disabled, this value is implicit zero.

9.88.2.3 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

9.89 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 "GetNative↩ Handle".
- 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 OnProcessEvents ()=0

9.89.1 Constructor & Destructor Documentation

9.89.1.1 virtual LLGL::Window::~Window ( ) [virtual]

9.89.2 Member Function Documentation

9.89.2.1 void LLGL::Window::AddEventListener (const std::shared_ptr< EventListener > & *eventListener*)

9.89.2.2 static std::unique_ptr<Window> LLGL::Window::Create (const WindowDescriptor & *desc*) [static]

9.89.2.3 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:

```
// Example for a custom Win32 window class
#include <LLGL/Platform/NativeHandle.h>
//...
void YourWindowClass::GetNativeHandle(void* nativeHandle)
{
    auto handle = reinterpret_cast<LLGL::NativeHandle*>(nativeHandle);
    //handle->window = 'some HWND window handle';
}
```

9.89.2.4 virtual Point LLGL::Window::GetPosition ( ) const [pure virtual]

9.89.2.5 virtual Size LLGL::Window::GetSize (bool *useClientArea* = true) const [pure virtual]

9.89.2.6 virtual std::wstring LLGL::Window::GetTitle ( ) const [pure virtual]

9.89.2.7 virtual bool LLGL::Window::IsShown ( ) const [pure virtual]

9.89.2.8 virtual void LLGL::Window::OnProcessEvents ( ) [protected], [pure virtual]

Called inside the "ProcessEvents" function after all event listeners received the same event.

See also

ProcessEvents
EventListener::OnProcessEvents

9.89.2.9 void LLGL::Window::PostChar (wchar_t *chr*)

9.89.2.10 void LLGL::Window::PostDoubleClick (Key *keyCode*)

9.89.2.11 void LLGL::Window::PostGlobalMotion (const Point & *motion*)

9.89.2.12 void LLGL::Window::PostKeyDown (Key *keyCode*)

9.89.2.13 void LLGL::Window::PostKeyUp (Key *keyCode*)

9.89.2.14 void LLGL::Window::PostLocalMotion (const Point & *position*)

9.89.2.15 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.

9.89.2.16 void LLGL::Window::PostResize (const Size & *clientAreaSize*)

9.89.2.17 void LLGL::Window::PostWheelMotion (int *motion*)

9.89.2.18 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 happend, when the user clicks on the close button.

9.89.2.19 virtual WindowDescriptor LLGL::Window::QueryDesc ( ) const [pure virtual]

Query a window descriptor, which describes the current state of this window.

9.89.2.20 virtual void LLGL::Window::Recreate (const WindowDescriptor & *desc*) [pure virtual]

Recreates the internal window object. This may invalidate the native handle previously returned by "GetNative↩ Handle".

See also

GetNativeHandle

9.89.2.21 void LLGL::Window::RemoveEventListener (const EventListener * *eventListener*)

9.89.2.22 virtual void LLGL::Window::SetDesc (const WindowDescriptor & *desc*) [pure virtual]

Sets the new window descriptor.

9.89.2.23 virtual void LLGL::Window::SetPosition (const Point & *position*) [pure virtual]

9.89.2.24 virtual void LLGL::Window::SetSize (const Size & *size*, bool *useClientArea* = true) [pure virtual]

9.89.2.25 virtual void LLGL::Window::SetTitle (const std::wstring & *title*) [pure virtual]

9.89.2.26 virtual void LLGL::Window::Show (bool *show* = true) [pure virtual]

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

- Window.h

9.90 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.

9.90.1 Detailed Description

Window descriptor structure.

9.90.2 Member Data Documentation

9.90.2.1 bool LLGL::WindowDescriptor::acceptDropFiles = false

9.90.2.2 bool LLGL::WindowDescriptor::borderless = false

9.90.2.3 bool LLGL::WindowDescriptor::centered = false

9.90.2.4 Point LLGL::WindowDescriptor::position

Window position (relative to the client area).

9.90.2.5 bool LLGL::WindowDescriptor::preventForPowerSafe = false

9.90.2.6 bool LLGL::WindowDescriptor::resizable = false

9.90.2.7 Size LLGL::WindowDescriptor::size

Client area size.

9.90.2.8 std::wstring LLGL::WindowDescriptor::title

9.90.2.9 bool LLGL::WindowDescriptor::visible = false

9.90.2.10 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 10

File Documentation

10.1 Buffer.h File Reference

```
#include "Export.h"
#include "BufferFlags.h"
```

Classes

- class LLGL::Buffer

  Hardware buffer interface.

Namespaces

- LLGL

10.2 BufferArray.h File Reference

```
#include "Export.h"
#include "BufferFlags.h"
```

Classes

- class LLGL::BufferArray

  Array of hardware buffers interface.

Namespaces

- LLGL

10.3 BufferFlags.h File Reference

```
#include "Export.h"
#include "VertexFormat.h"
#include "IndexFormat.h"
#include "RenderSystemFlags.h"
#include <string>
```

Classes

- struct LLGL::BufferFlags

  Buffer flags enumeration.
- struct LLGL::BufferDescriptor

  Hardware buffer descriptor structure.
- struct LLGL::BufferDescriptor::VertexBufferDescriptor

  Vertex buffer descriptor structure.
- struct LLGL::BufferDescriptor::IndexBufferDescriptor
- struct LLGL::BufferDescriptor::StorageBufferDescriptor
- struct LLGL::ConstantBufferViewDescriptor

  Constant buffer shader-view descriptor structure.
- struct LLGL::StorageBufferViewDescriptor

  Storage buffer shader-view descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::BufferType { LLGL::BufferType::Vertex, LLGL::BufferType::Index, LLGL::BufferType::Constant, LLGL::BufferType::Storage, LLGL::BufferType::StreamOutput }

  Hardware buffer type enumeration.
- 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.
- enum LLGL::BufferCPUAccess { LLGL::BufferCPUAccess::ReadOnly, LLGL::BufferCPUAccess::WriteOnly, LLGL::BufferCPUAccess::ReadWrite }

  Hardware buffer CPU acccess enumeration.

10.4 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)

10.5 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 >

10.6 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::ColorRGBAub = ColorRGBAT< unsigned char >

10.7 CommandBuffer.h File Reference

```
#include "Export.h"
#include "RenderContextFlags.h"
#include "RenderSystemFlags.h"
#include "ColorRGBA.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "Texture.h"
#include "TextureArray.h"
#include "Sampler.h"
#include "SamplerArray.h"
#include "RenderTarget.h"
#include "ShaderProgram.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "Query.h"
```

Classes

- class LLGL::CommandBuffer

  Command buffer interface.

Namespaces

- LLGL

10.8 ComputePipeline.h File Reference

```
#include "Export.h"
```

Classes

- struct LLGL::ComputePipelineDescriptor

  Compute pipeline descriptor structure.
- class LLGL::ComputePipeline

  Compute pipeline interface.

Namespaces

- LLGL

10.9 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.

10.10 Export.h File Reference

Macros

- #define LLGL_EXPORT

10.10.1 Macro Definition Documentation

10.10.1.1 #define LLGL_EXPORT

10.11 Format.h File Reference

```
#include "Export.h"
```

Namespaces

- LLGL

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::VectorType {
  LLGL::VectorType::Float, LLGL::VectorType::Float2, LLGL::VectorType::Float3, LLGL::VectorType::Float4,
  LLGL::VectorType::Double, LLGL::VectorType::Double2, LLGL::VectorType::Double3, LLGL::VectorType::↩
  Double4,
  LLGL::VectorType::Int, LLGL::VectorType::Int2, LLGL::VectorType::Int3, LLGL::VectorType::Int4,
  LLGL::VectorType::UInt, LLGL::VectorType::UInt2, LLGL::VectorType::UInt3, LLGL::VectorType::UInt4 }

  Renderer vector types enumeration.

Functions

- LLGL_EXPORT unsigned int LLGL::DataTypeSize (const DataType dataType)

  Returns the size (in bytes) of the specified data type.
- LLGL_EXPORT unsigned int LLGL::VectorTypeSize (const VectorType vectorType)

  Returns the size (in bytes) of the specified vector type.
- LLGL_EXPORT void LLGL::VectorTypeFormat (const VectorType vectorType, DataType &dataType, unsigned int &components)

  Retrieves the format of the specified vector type.

10.12 GraphicsPipeline.h File Reference

```
#include "Export.h"
#include "GraphicsPipelineFlags.h"
```

Classes

- class LLGL::GraphicsPipeline

  Graphics pipeline interface.

Namespaces

- LLGL

10.13 GraphicsPipelineFlags.h File Reference

```
#include "Export.h"
#include "ColorRGBA.h"
#include <vector>
#include <cstdint>
```

Classes

- struct LLGL::MultiSamplingDescriptor

  Multi-sampling descriptor structure.
- 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::PrimitiveType { LLGL::PrimitiveType::Points, LLGL::PrimitiveType::Lines, LLGL::PrimitiveType::Triangles }

  Primitive type enumeration.
- 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, LLGL::BlendOp::SrcAlphaSaturate, LLGL::BlendOp::BlendFactor, LLGL::BlendOp::InvBlendFactor, LLGL::BlendOp::Src1Color, LLGL::BlendOp::InvSrc1Color, LLGL::BlendOp::Src1Alpha, LLGL::BlendOp::InvSrc1Alpha }

  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.

10.14 Image.h File Reference

```
#include "Export.h"
#include "Format.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::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 unsigned int 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 foramt 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).

10.15 IndexFormat.h File Reference

```
#include "Export.h"
#include "Image.h"
```

Classes

- class LLGL::IndexFormat

Namespaces

- LLGL

10.16 Input.h File Reference

```
#include <LLGL/Window.h>
#include <LLGL/Types.h>
#include <array>
#include <string>
```

Classes

- class LLGL::Input

Namespaces

- LLGL

10.17 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::BrowserFavorits, 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.

10.18 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

10.19 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"
#include "Version.h"
```

10.20 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.

10.21 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

10.22 NativeHandle.h File Reference

10.23 Query.h File Reference

```
#include "Export.h"
#include "QueryFlags.h"
```

Classes

- class LLGL::Query

  Query interface.

Namespaces

- LLGL

10.24 QueryFlags.h File Reference

Classes

- struct LLGL::QueryDescriptor

  Query descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::QueryType {
  LLGL::QueryType::SamplesPassed, LLGL::QueryType::AnySamplesPassed, LLGL::QueryType::AnySamplesPassedConservative, 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.

10.25 RenderContext.h File Reference

```
#include "Export.h"
#include "Window.h"
#include "RenderContextDescriptor.h"
#include "RenderContextFlags.h"
#include "RenderSystemFlags.h"
#include "ColorRGBA.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "ShaderProgram.h"
#include "Texture.h"
#include "TextureArray.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

10.26 RenderContextDescriptor.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "GraphicsPipelineFlags.h"
#include <functional>
```

Classes

- struct LLGL::VsyncDescriptor

  Vertical-synchronization (Vsync) descriptor structure.
- struct LLGL::VideoModeDescriptor

  Video mode descriptor structure.
- struct LLGL::ProfileOpenGLDescriptor

  OpenGL profile descriptor structure.
- struct LLGL::RenderContextDescriptor

  Render context descriptor structure.

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::OpenGL↩
  Version::OpenGL_1_1 = 110, LLGL::OpenGLVersion::OpenGL_1_2 = 120,
  LLGL::OpenGLVersion::OpenGL_1_3 = 130, LLGL::OpenGLVersion::OpenGL_1_4 = 140, LLGL::OpenGL↩
  Version::OpenGL_1_5 = 150, LLGL::OpenGLVersion::OpenGL_2_0 = 200,
  LLGL::OpenGLVersion::OpenGL_2_1 = 210, LLGL::OpenGLVersion::OpenGL_3_0 = 300, LLGL::OpenGL↩
  Version::OpenGL_3_1 = 310, LLGL::OpenGLVersion::OpenGL_3_2 = 320,
  LLGL::OpenGLVersion::OpenGL_3_3 = 330, LLGL::OpenGLVersion::OpenGL_4_0 = 400, LLGL::OpenGL↩
  Version::OpenGL_4_1 = 410, LLGL::OpenGLVersion::OpenGL_4_2 = 420,
  LLGL::OpenGLVersion::OpenGL_4_3 = 430, LLGL::OpenGLVersion::OpenGL_4_4 = 440, LLGL::OpenGL↩
  Version::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)

10.27 RenderContextFlags.h File Reference

Classes

- struct LLGL::ClearFlags

  Command buffer clear 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::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.
- enum LLGL::LogicOp {
  LLGL::LogicOp::Keep, LLGL::LogicOp::Disabled, LLGL::LogicOp::Clear, LLGL::LogicOp::Set,
  LLGL::LogicOp::Copy, LLGL::LogicOp::InvertedCopy, LLGL::LogicOp::Noop, LLGL::LogicOp::Invert,
  LLGL::LogicOp::AND, LLGL::LogicOp::NAND, LLGL::LogicOp::OR, LLGL::LogicOp::NOR,
  LLGL::LogicOp::XOR, LLGL::LogicOp::Equiv, LLGL::LogicOp::ReverseAND, LLGL::LogicOp::InvertedAND,
  LLGL::LogicOp::ReverseOR, LLGL::LogicOp::InvertedOR }

  Logical pixel operation enumeration.

10.28 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::Error↩ Type::UnsupportedFeature }

  Rendering debugger error types enumeration.
- enum LLGL::WarningType { LLGL::WarningType::ImproperArgument, LLGL::WarningType::ImproperState, LLGL::WarningType::PointlessOperation }

10.29 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

  Profiling counter class.

Namespaces

- LLGL

10.30 RenderSystem.h File Reference

```
#include "Export.h"
#include "RenderContext.h"
#include "CommandBuffer.h"
#include "RenderSystemFlags.h"
#include "RenderingProfiler.h"
#include "RenderingDebugger.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "Texture.h"
#include "TextureArray.h"
#include "Sampler.h"
#include "SamplerArray.h"
#include "RenderTarget.h"
#include "ShaderProgram.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "Query.h"
#include <string>
#include <memory>
#include <vector>
```

Classes

- class LLGL::RenderSystem

  Render system interface.

Namespaces

- LLGL

10.31 RenderSystemFlags.h File Reference

```
#include <Gauss/Vector3.h>
#include "ColorRGBA.h"
#include <cstddef>
#include <string>
```

Classes

- struct LLGL::RenderSystemConfiguration

  Render system configuration structure.
- struct LLGL::RendererID

  Renderer identification number enumeration.
- struct LLGL::RendererInfo

  Renderer basic information structure.
- struct LLGL::RenderingCaps

  Rendering capabilities structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::ShadingLanguage {
  LLGL::ShadingLanguage::Unsupported = 0, LLGL::ShadingLanguage::GLSL_110 = 110, LLGL::Shading↩
  Language::GLSL_120 = 120, LLGL::ShadingLanguage::GLSL_130 = 130,
  LLGL::ShadingLanguage::GLSL_140 = 140, LLGL::ShadingLanguage::GLSL_150 = 150, LLGL::Shading↩
  Language::GLSL_330 = 330, LLGL::ShadingLanguage::GLSL_400 = 400,
  LLGL::ShadingLanguage::GLSL_410 = 410, LLGL::ShadingLanguage::GLSL_420 = 420, LLGL::Shading↩
  Language::GLSL_430 = 430, LLGL::ShadingLanguage::GLSL_440 = 440,
  LLGL::ShadingLanguage::GLSL_450 = 450, LLGL::ShadingLanguage::HLSL_2_0 = 100200, LLGL::↩
  ShadingLanguage::HLSL_2_0a = 100201, LLGL::ShadingLanguage::HLSL_2_0b = 100202,
  LLGL::ShadingLanguage::HLSL_3_0 = 100300, LLGL::ShadingLanguage::HLSL_4_0 = 100400, LLGL::↩
  ShadingLanguage::HLSL_4_1 = 100410, LLGL::ShadingLanguage::HLSL_5_0 = 100500 }

  Shading language version enumation.
- 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.

10.32 RenderTarget.h File Reference

```
#include "Export.h"
#include "TextureFlags.h"
#include "GraphicsPipelineFlags.h"
#include <Gauss/Vector2.h>
```

Classes

- struct LLGL::RenderTargetAttachmentDescriptor

  Render target attachment descriptor structure.
- struct LLGL::RenderTargetDescriptor

  Render target descriptor structure.
- class LLGL::RenderTarget

  Render target interface.

Namespaces

- LLGL

10.33 Sampler.h File Reference

```
#include "Export.h"
#include "SamplerFlags.h"
```

Classes

- class LLGL::Sampler

  Sampler interface.

Namespaces

- LLGL

10.34 SamplerArray.h File Reference

```
#include "Export.h"
```

Classes

- class LLGL::SamplerArray

  Sampler array interface.

Namespaces

- LLGL

10.35 SamplerFlags.h File Reference

```
#include "Export.h"
#include "GraphicsPipelineFlags.h"
#include "ColorRGBA.h"
#include <cstddef>
```

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.

10.36 Shader.h File Reference

```
#include "Export.h"
#include "ShaderFlags.h"
```

Classes

- class LLGL::Shader

  Shader interface.

Namespaces

- LLGL

10.37 ShaderFlags.h File Reference

```
#include "Export.h"
#include "StreamOutputFormat.h"
#include <string>
```

Classes

- struct LLGL::ShaderCompileFlags

  Shader compilation flags enumeration.
- struct LLGL::ShaderDisassembleFlags

  Shader disassemble flags enumeration.
- struct LLGL::ShaderStageFlags

  Shader stage flags.
- struct LLGL::ShaderSource

  Shader source code structure.
- struct LLGL::ShaderSource::SourceHLSL

  Additional descripor for HLSL shader source.
- struct LLGL::ShaderSource::StreamOutput

  Additional descriptor for stream outputs.

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.

10.38 ShaderProgram.h File Reference

```
#include "Export.h"
#include "Shader.h"
#include "VertexFormat.h"
#include "StreamOutputFormat.h"
#include "BufferFlags.h"
#include "ShaderUniform.h"
#include <string>
#include <vector>
```

Classes

- class LLGL::ShaderProgram

  Shader program interface.

Namespaces

- LLGL

10.39 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::UniformType::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.

10.40 StreamOutputAttribute.h File Reference

```
#include "Export.h"
#include "Format.h"
#include <string>
```

Classes

- struct LLGL::StreamOutputAttribute

  Stream-output attribute structure.

Namespaces

- LLGL

Functions

- LLGL_EXPORT bool LLGL::operator== (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)
- LLGL_EXPORT bool LLGL::operator!= (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)

10.41 StreamOutputFormat.h File Reference

```
#include "Export.h"
#include "StreamOutputAttribute.h"
#include <vector>
```

Classes

- struct LLGL::StreamOutputFormat

  Stream-output format descriptor structure.

Namespaces

- LLGL

10.42 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

10.43 TextureArray.h File Reference

```
#include "Export.h"
```

Classes

- class LLGL::TextureArray

  Array of textures interface.

Namespaces

- LLGL

10.44 TextureFlags.h File Reference

```
#include "Export.h"
#include <Gauss/Vector3.h>
#include <cstddef>
```

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::TextureDescriptor::Texture2DMSDescriptor
- 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::Texture1D, LLGL::TextureType::Texture2D, LLGL::TextureType::Texture3D, LLGL::TextureType::TextureCube,
  LLGL::TextureType::Texture1DArray, LLGL::TextureType::Texture2DArray, LLGL::TextureType::TextureCubeArray, LLGL::TextureType::Texture2DMS,
  LLGL::TextureType::Texture2DMSArray }

  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::RGBA32Float, 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::AxisDirection::YNeg,
  LLGL::AxisDirection::ZPos, LLGL::AxisDirection::ZNeg }

  Axis direction (also used for texture cube face).

Functions

- LLGL_EXPORT unsigned int LLGL::NumMipLevels (unsigned int width, unsigned int height=1, unsigned int depth=1)

  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, TextureFormat::RGBA_DXT1, TextureFormat::RGBA_DXT3, or TextureFormat::RGBA_DXT5.
- LLGL_EXPORT bool LLGL::IsArrayTexture (const TextureType type)

  Returns true if the specified texture type is an array texture.
- LLGL_EXPORT bool LLGL::IsMultiSampleTexture (const TextureType type)

  Returns true if the specified texture type is a multi-sample texture.

10.45 Timer.h File Reference

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

Classes

- class LLGL::Timer

Namespaces

- LLGL

10.46 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)

10.47 Utility.h File Reference

```
#include "Export.h"
#include "TextureFlags.h"
#include "BufferFlags.h"
```

Namespaces

- LLGL

Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (TextureFormat format, unsigned int width)

  Returns a TextureDescriptor structure with the TextureType::Texture1D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::Texture2D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int depth)

  Returns a TextureDescriptor structure with the TextureType::Texture3D type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (TextureFormat format, unsigned int width, unsigned int height)

  Returns a TextureDescriptor structure with the TextureType::TextureCube type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (TextureFormat format, unsigned int width, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers)

  Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (TextureFormat format, unsigned int width, unsigned int height, unsigned int layers, unsigned int samples, bool fixedSamples=true)

  Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.
- LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (unsigned int size, const VertexFormat &vertex↩Format, long flags=0)

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (unsigned int size, const IndexFormat &index↩Format, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (unsigned int size, long flags=BufferFlags::↩DynamicUsage)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (unsigned int size, const StorageBufferType storageType, unsigned int stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  Returns a BufferDescriptor structure for a storage buffer.

10.48 Version.h File Reference

```
#include "Export.h"
#include <string>
```

Namespaces

- LLGL
- LLGL::Version

Functions

- LLGL_EXPORT unsigned int LLGL::Version::GetMajor ()

  Returns the major LLGL version (e.g. 1 stands for "1.00").
- LLGL_EXPORT unsigned int LLGL::Version::GetMinor ()

  Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.
- LLGL_EXPORT unsigned int LLGL::Version::GetRevision ()

  Returns the revision version number. Must be less than 100.
- LLGL_EXPORT std::string LLGL::Version::GetStatus ()

  Returns the LLGL version status (either "Alpha", "Beta", or empty).
- LLGL_EXPORT unsigned int LLGL::Version::GetID ()

  Returns the full LLGL version as an ID number (e.g. 200317 stands for "2.03 (Rev. 17)").
- LLGL_EXPORT std::string LLGL::Version::GetString ()

  Returns the full LLGL version as a string (e.g. "0.01 Beta (Rev. 1)").

10.49 VertexAttribute.h File Reference

```
#include "Export.h"
#include "Format.h"
#include <string>
```

Classes

- struct LLGL::VertexAttribute

  Vertex attribute structure.

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)

10.50 VertexFormat.h File Reference

```
#include "Export.h"
#include "VertexAttribute.h"
#include <vector>
```

Classes

- struct LLGL::VertexFormat

  Vertex format descriptor structure.

Namespaces

- LLGL

10.51 VideoAdapter.h File Reference

```
#include "Export.h"
#include <vector>
#include <string>
```

Classes

- struct LLGL::VideoDisplayMode

  Video display mode structure.
- struct LLGL::VideoOutput

  Video output structure.
- 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.

10.52 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

10.53 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

~Buffer
LLGL::Buffer, 58
~BufferArray
LLGL::BufferArray, 59
~CommandBuffer
LLGL::CommandBuffer, 74
~ComputePipeline
LLGL::ComputePipeline, 86
~EventListener
LLGL::Window::EventListener, 92
~GraphicsPipeline
LLGL::GraphicsPipeline, 94
~Query
LLGL::Query, 107
~RenderContext
LLGL::RenderContext, 111
~RenderSystem
LLGL::RenderSystem, 133
~RenderTarget
LLGL::RenderTarget, 145
~RenderingDebugger
LLGL::RenderingDebugger, 125
~Sampler
LLGL::Sampler, 150
~SamplerArray
LLGL::SamplerArray, 150
~Shader
LLGL::Shader, 155
~ShaderProgram
LLGL::ShaderProgram, 159
~ShaderUniform
LLGL::ShaderUniform, 170
~SubTextureDescriptor
LLGL::SubTextureDescriptor, 184
~Texture
LLGL::Texture, 185
~TextureArray
LLGL::TextureArray, 193
~TextureDescriptor
LLGL::TextureDescriptor, 196
~Timer
LLGL::Timer, 198
~Window
LLGL::Window, 213

A
LLGL, 35
a
LLGL::Color< T, 4u >, 71
AND
LLGL, 38
acceptDropFiles
LLGL::WindowDescriptor, 216
Add
LLGL, 30
AddEventListener
LLGL::Window, 213
All
LLGL::ClearFlags, 63
AllGraphicsStages
LLGL::ShaderStageFlags, 168
AllStages
LLGL::ShaderStageFlags, 168
AllTessStages
LLGL::ShaderStageFlags, 168
alphaArithmetic
LLGL::BlendTargetDescriptor, 57
antiAliasedLineEnabled
LLGL::RasterizerDescriptor, 109
AnySamplesPassed
LLGL, 41
AnySamplesPassedConservative
LLGL, 41
AppendAttribute
LLGL::StreamOutputFormat, 182
LLGL::VertexFormat, 204
AppendAttributes
LLGL::StreamOutputFormat, 183
LLGL::VertexFormat, 205
AppendStructuredBuffer
LLGL, 43
ApplyMipResolution
LLGL::RenderTarget, 145
ApplyResolution
LLGL::RenderTarget, 145
Apps
LLGL, 35
AssertCreateBuffer
LLGL::RenderSystem, 133
AssertCreateBufferArray
LLGL::RenderSystem, 133
AssertCreateSamplerArray
LLGL::RenderSystem, 134
AssertCreateTextureArray
LLGL::RenderSystem, 134
AttachDepthBuffer
LLGL::RenderTarget, 145
AttachDepthStencilBuffer
LLGL::RenderTarget, 146

AttachShader
LLGL::ShaderProgram, 159
AttachStencilBuffer
LLGL::RenderTarget, 146
AttachTexture
LLGL::RenderTarget, 146
Attn
LLGL, 37
attributes
LLGL::StreamOutputFormat, 183
LLGL::VertexFormat, 205
AxisDirection
LLGL, 30

B
LLGL, 35
b
LLGL::Color< T, 3u >, 68
LLGL::Color< T, 4u >, 71
BGRA
LLGL, 33
BGR
LLGL, 33
Back
LLGL, 32, 34
back
LLGL::StencilDescriptor, 175
BeginQuery
LLGL::CommandBuffer, 74
BeginRenderCondition
LLGL::CommandBuffer, 74
BeginStreamOutput
LLGL::CommandBuffer, 75
BindConstantBuffer
LLGL::ShaderProgram, 160
BindStorageBuffer
LLGL::ShaderProgram, 160
blend
LLGL::GraphicsPipelineDescriptor, 95
BlendArithmetic
LLGL, 30
blendEnabled
LLGL::BlendDescriptor, 55
BlendFactor
LLGL, 31
blendFactor
LLGL::BlendDescriptor, 55
BlendOp
LLGL, 30
Block
LLGL::RenderingDebugger::Message, 102
BlockAfter
LLGL::RenderingDebugger::Message, 102
Border
LLGL, 46
borderColor
LLGL::SamplerDescriptor, 151
borderless
LLGL::WindowDescriptor, 216
BrowserBack
LLGL, 36
BrowserFavorits
LLGL, 36
BrowserForward
LLGL, 36
BrowserHome
LLGL, 36
BrowserRefresh
LLGL, 36
BrowserSearch
LLGL, 36
BrowserStop
LLGL, 36
Buffer
LLGL::Buffer, 58
LLGL, 43
buffer
LLGL::ImageDescriptor, 97
Buffer.h, 217
BufferArray
LLGL::BufferArray, 59
BufferArray.h, 217
BufferCPUAccess
LLGL, 31
BufferFlags.h, 218
BufferType
LLGL, 31
BuildInputLayout
LLGL::ShaderProgram, 161
ByRegionNoWait
LLGL, 41
ByRegionNoWaitInverted
LLGL, 41
ByRegionWait
LLGL, 41
ByRegionWaitInverted
LLGL, 41
ByteAddressBuffer
LLGL, 43
ByteBuffer
LLGL, 28

C
LLGL, 35
Cancel
LLGL, 34
Capital
LLGL, 34
Cast
LLGL::Color, 64
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
centered
LLGL::WindowDescriptor, 216
Clamp
LLGL, 46
Clear
LLGL::CommandBuffer, 75

LLGL, 34, 37
ClippingInputPrimitives
LLGL, 41
ClippingOutputPrimitives
LLGL, 41
ClippingRange
LLGL, 32
clippingRange
LLGL::RenderingCaps, 119
Color
LLGL::ClearFlags, 63
LLGL::Color, 64
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
Color.h, 218
colorArithmetic
LLGL::BlendTargetDescriptor, 57
ColorDepth
LLGL::ClearFlags, 63
colorDepth
LLGL::VideoModeDescriptor, 208
colorMap
LLGL::NativeContextHandle, 104
colorMask
LLGL::BlendTargetDescriptor, 57
ColorRGB.h, 219
ColorRGBA.h, 220
ColorRGBAb
LLGL, 29
ColorRGBAd
LLGL, 29
ColorRGBAf
LLGL, 29
ColorRGBAT
LLGL, 29
ColorRGBAub
LLGL, 29
ColorRGBA
LLGL, 29
ColorRGBb
LLGL, 29
ColorRGBd
LLGL, 29
ColorRGBf
LLGL, 29
ColorRGBT
LLGL, 29
ColorRGBub
LLGL, 29
ColorRGB
LLGL, 29
Comma
LLGL, 37
CommandBuffer
LLGL::CommandBuffer, 74
CommandBuffer.h, 221
CompareOp
LLGL, 32
compareOp
LLGL::DepthDescriptor, 90
LLGL::SamplerDescriptor, 151
LLGL::StencilFaceDescriptor, 176
CompareSWO
LLGL, 48
Compile
LLGL::Shader, 155
components
LLGL::Color, 65
LLGL::Color< T, 3u >, 68
LLGL::Color< T, 4u >, 71
LLGL::StreamOutputAttribute, 181
CompressedRGBA
LLGL, 33
CompressedRGB
LLGL, 33
compressedSize
LLGL::ImageDescriptor, 97
Compute
LLGL, 42
ComputePipeline.h, 221
ComputePipelineDescriptor
LLGL::ComputePipelineDescriptor, 87
ComputeShaderInvocations
LLGL, 41
ComputeStage
LLGL::ShaderStageFlags, 168
conservativeRasterization
LLGL::RasterizerDescriptor, 109
Constant
LLGL, 32
ConstantBufferDesc
Global utility functions, especially to fill descriptor structures., 16
ConsumeStructuredBuffer
LLGL, 43
Control
LLGL, 34
conversion
LLGL::VertexAttribute, 202
ConvertImageBuffer
LLGL, 48
Copy
LLGL, 37
coreProfile
LLGL::ProfileOpenGLDescriptor, 106
Count
LLGL::RenderingProfiler::Counter, 89
CrSel
LLGL, 37
Create
LLGL::Timer, 198
LLGL::Window, 213
CreateBuffer
LLGL::RenderSystem, 134
CreateBufferArray
LLGL::RenderSystem, 134

CreateCommandBuffer
LLGL::RenderSystem, 135
CreateComputePipeline
LLGL::RenderSystem, 135
CreateGraphicsPipeline
LLGL::RenderSystem, 135
CreateQuery
LLGL::RenderSystem, 135
CreateRenderContext
LLGL::RenderSystem, 135
CreateRenderTarget
LLGL::RenderSystem, 136
CreateSampler
LLGL::RenderSystem, 136
CreateSamplerArray
LLGL::RenderSystem, 136
CreateShader
LLGL::RenderSystem, 137
CreateShaderProgram
LLGL::RenderSystem, 137
CreateTexture
LLGL::RenderSystem, 137
CreateTextureArray
LLGL::RenderSystem, 137
cubeFace
LLGL::RenderTargetAttachmentDescriptor, 148
cubeFaceOffset
LLGL::SubTextureDescriptor::TextureCube↩
Descriptor, 194
cubeFaces
LLGL::SubTextureDescriptor::TextureCube↩
Descriptor, 194
CullMode
LLGL, 32
cullMode
LLGL::RasterizerDescriptor, 109
customMultiSampling
LLGL::RenderTargetDescriptor, 149

D
LLGL, 35
D0
LLGL, 34
D1
LLGL, 34
D2
LLGL, 34
D3
LLGL, 34
D4
LLGL, 34
D5
LLGL, 34
D6
LLGL, 34
D7
LLGL, 34
D8
LLGL, 34
D9
LLGL, 35
DataType
LLGL, 32
dataType
LLGL::ImageDescriptor, 97
DataTypeSize
LLGL, 48
Debug
LLGL::ShaderCompileFlags, 156
DebugCallback
LLGL, 29
debugCallback
LLGL::RenderContextDescriptor, 113
debugDump
LLGL::ProfileOpenGLDescriptor, 106
DecClamp
LLGL, 43
DecWrap
LLGL, 43
defaultImageColor
LLGL::RenderSystemConfiguration, 144
Delete
LLGL, 34
Depth
LLGL::ClearFlags, 63
LLGL, 33
depth
LLGL::GraphicsPipelineDescriptor, 95
LLGL::SubTextureDescriptor::Texture3DDescriptor,
191
LLGL::TextureDescriptor::Texture3DDescriptor,
191
depthBias
LLGL::RasterizerDescriptor, 110
depthBiasClamp
LLGL::RasterizerDescriptor, 110
depthClampEnabled
LLGL::RasterizerDescriptor, 110
depthCompare
LLGL::SamplerDescriptor, 152
DepthComponent
LLGL, 44
depthFailOp
LLGL::StencilFaceDescriptor, 176
depthPassOp
LLGL::StencilFaceDescriptor, 176
DepthStencil
LLGL::ClearFlags, 63
LLGL, 33, 44
Desktop.h, 222
DestAlpha
LLGL, 31
destAlpha
LLGL::BlendTargetDescriptor, 57
DestColor
LLGL, 31
destColor

LLGL::BlendTargetDescriptor, 57
DetachAll
LLGL::RenderTarget, 147
LLGL::ShaderProgram, 161
deviceName
LLGL::RendererInfo, 117
Direct3D11
LLGL::RendererID, 115
Direct3D12
LLGL::RendererID, 115
Disabled
LLGL, 32, 37
Disassemble
LLGL::Shader, 155
Dispatch
LLGL::CommandBuffer, 76
dispatchComputeCalls
LLGL::RenderingProfiler, 127
display
LLGL::NativeContextHandle, 104
LLGL::NativeHandle, 105
displayModes
LLGL::VideoOutput, 209
Double
LLGL, 33, 46, 47
Double2
LLGL, 46, 47
Double2x2
LLGL, 47
Double3
LLGL, 46, 47
Double3x3
LLGL, 47
Double4
LLGL, 46, 47
Double4x4
LLGL, 47
DoubleBuffering
LLGL, 44
Down
LLGL, 34
Draw
LLGL::CommandBuffer, 76
drawCalls
LLGL::RenderingProfiler, 127
DrawIndexed
LLGL::CommandBuffer, 76
DrawIndexedInstanced
LLGL::CommandBuffer, 77
DrawInstanced
LLGL::CommandBuffer, 77
DynamicUsage
LLGL::BufferFlags, 62

E
LLGL, 35
enabled
LLGL::MultiSamplingDescriptor, 103
LLGL::VsyncDescriptor, 211
End
LLGL, 34
EndQuery
LLGL::CommandBuffer, 78
EndRenderCondition
LLGL::CommandBuffer, 78
EndStreamOutput
LLGL::CommandBuffer, 78
entryPoint
LLGL::ShaderSource::SourceHLSL, 173
Equal
LLGL, 32
Equiv
LLGL, 38
ErEOF
LLGL, 37
ErrorType
LLGL, 33
Escape
LLGL, 34
Ever
LLGL, 32
ExSel
LLGL, 37
Exe
LLGL, 34
Exponent
LLGL, 37
Export.h, 222
LLGL_EXPORT, 222
extProfile
LLGL::ProfileOpenGLDescriptor, 106

F
LLGL, 35
F1
LLGL, 36
F10
LLGL, 36
F11
LLGL, 36
F12
LLGL, 36
F13
LLGL, 36
F14
LLGL, 36
F15
LLGL, 36
F16
LLGL, 36
F17
LLGL, 36
F18
LLGL, 36
F19
LLGL, 36
F2
LLGL, 36

F20
LLGL, 36
F21
LLGL, 36
F22
LLGL, 36
F23
LLGL, 36
F24
LLGL, 36
F3
LLGL, 36
F4
LLGL, 36
F5
LLGL, 36
F6
LLGL, 36
F7
LLGL, 36
F8
LLGL, 36
F9
LLGL, 36
Fill
LLGL, 38
FindModules
LLGL::RenderSystem, 138
fixedSamples
LLGL::TextureDescriptor::Texture2DMSDescriptor, 190
flags
LLGL::BufferDescriptor, 60
LLGL::ShaderSource::SourceHLSL, 173
Float
LLGL, 33, 46, 47
Float2
LLGL, 46, 47
Float2x2
LLGL, 47
Float3
LLGL, 46, 47
Float3x3
LLGL, 47
Float4
LLGL, 46, 47
Float4x4
LLGL, 47
format
LLGL::BufferDescriptor::IndexBufferDescriptor, 98
LLGL::BufferDescriptor::VertexBufferDescriptor, 203
LLGL::ImageDescriptor, 97
LLGL::ShaderSource::StreamOutput, 180
LLGL::TextureDescriptor, 196
Format.h, 222
Fragment
LLGL, 42
FragmentShaderInvocations
LLGL, 41
FragmentStage
LLGL::ShaderStageFlags, 168
FrameCount
LLGL::Timer, 198
Front
LLGL, 32
front
LLGL::StencilDescriptor, 175
frontCCW
LLGL::RasterizerDescriptor, 110
fullscreen
LLGL::VideoModeDescriptor, 208

G
LLGL, 35
g
LLGL::Color< T, 3u >, 68
LLGL::Color< T, 4u >, 71
GLSL_110
LLGL, 42
GLSL_120
LLGL, 42
GLSL_130
LLGL, 42
GLSL_140
LLGL, 42
GLSL_150
LLGL, 42
GLSL_330
LLGL, 42
GLSL_400
LLGL, 42
GLSL_410
LLGL, 42
GLSL_420
LLGL, 42
GLSL_430
LLGL, 42
GLSL_440
LLGL, 42
GLSL_450
LLGL, 42
GenerateMips
LLGL::RenderSystem, 138
Geometry
LLGL, 42
GeometryPrimitivesGenerated
LLGL, 41
GeometryShaderInvocations
LLGL, 41
GeometryStage
LLGL::ShaderStageFlags, 168
GetColorDepth
LLGL::Desktop, 52
GetConfiguration
LLGL::RenderSystem, 138
GetDataType

LLGL::IndexFormat, 99
GetDefaultTextureImageRGBAub
LLGL::RenderSystem, 138
GetDeltaTime
LLGL::Timer, 198
GetElementSize
LLGL::ImageDescriptor, 97
GetEnteredChars
LLGL::Input, 100
GetFormatSize
LLGL::IndexFormat, 99
GetFrameCount
LLGL::Timer, 198
GetFrequency
LLGL::Timer, 198
GetID
LLGL::Version, 54
GetMajor
LLGL::Version, 54
GetMinor
LLGL::Version, 54
GetMouseMotion
LLGL::Input, 100
GetMousePosition
LLGL::Input, 100
GetName
LLGL::RenderSystem, 138
GetNativeHandle
LLGL::Window, 213
GetOccurrences
LLGL::RenderingDebugger::Message, 102
GetPosition
LLGL::Window, 213
GetRendererInfo
LLGL::RenderSystem, 138
GetRenderingCaps
LLGL::RenderSystem, 139
GetResolution
LLGL::Desktop, 52
LLGL::RenderTarget, 147
GetRevision
LLGL::Version, 54
GetSize
LLGL::VertexAttribute, 202
LLGL::Window, 213
GetSource
LLGL::RenderingDebugger::Message, 102
GetStatus
LLGL::Version, 54
GetString
LLGL::Version, 54
GetText
LLGL::RenderingDebugger::Message, 102
GetTitle
LLGL::Window, 213
GetType
LLGL::Buffer, 58
LLGL::BufferArray, 59
LLGL::Query, 107
LLGL::Shader, 155
LLGL::Texture, 185
GetVideoMode
LLGL::RenderContext, 111
GetWheelMotion
LLGL::Input, 100
GetWindow
LLGL::RenderContext, 111
Global utility functions, especially to fill descriptor structures., 15
ConstantBufferDesc, 16
IndexBufferDesc, 16
StorageBufferDesc, 16
Texture1DArrayDesc, 16
Texture1DDesc, 16
Texture2DArrayDesc, 16
Texture2DDesc, 16
Texture2DMSArrayDesc, 16
Texture2DMSDesc, 17
Texture3DDesc, 17
TextureCubeArrayDesc, 17
TextureCubeDesc, 17
VertexBufferDesc, 17
GraphicsAPIDependentStateDescriptor
LLGL::GraphicsAPIDependentStateDescriptor, 93
GraphicsPipeline.h, 223
GraphicsPipelineFlags.h, 223
Greater
LLGL, 32
GreaterEqual
LLGL, 32

H
LLGL, 35
HLSL_2_0
LLGL, 42
HLSL_2_0a
LLGL, 42
HLSL_2_0b
LLGL, 42
HLSL_3_0
LLGL, 42
HLSL_4_0
LLGL, 42
HLSL_4_1
LLGL, 42
HLSL_5_0
LLGL, 42
has3DTextures
LLGL::RenderingCaps, 119
hasComputeShaders
LLGL::RenderingCaps, 119
hasConservativeRasterization
LLGL::RenderingCaps, 120
hasConstantBuffers
LLGL::RenderingCaps, 120
hasCubeTextureArrays
LLGL::RenderingCaps, 120

hasCubeTextures
LLGL::RenderingCaps, 120
hasGeometryShaders
LLGL::RenderingCaps, 120
hasInstancing
LLGL::RenderingCaps, 120
hasMultiSampleTextures
LLGL::RenderingCaps, 121
hasOffsetInstancing
LLGL::RenderingCaps, 121
hasRenderTargets
LLGL::RenderingCaps, 121
hasSamplers
LLGL::RenderingCaps, 121
hasStorageBuffers
LLGL::RenderingCaps, 121
hasStreamOutputs
LLGL::RenderingCaps, 121
hasTessellationShaders
LLGL::RenderingCaps, 122
hasTextureArrays
LLGL::RenderingCaps, 122
hasUniforms
LLGL::RenderingCaps, 122
hasViewportArrays
LLGL::RenderingCaps, 122
height
LLGL::Scissor, 154
LLGL::SubTextureDescriptor::Texture2DDescriptor, 188
LLGL::SubTextureDescriptor::Texture3DDescriptor, 191
LLGL::SubTextureDescriptor::TextureCube↩ Descriptor, 194
LLGL::TextureDescriptor::Texture2DDescriptor, 188
LLGL::TextureDescriptor::Texture2DMSDescriptor, 190
LLGL::TextureDescriptor::Texture3DDescriptor, 191
LLGL::TextureDescriptor::TextureCubeDescriptor, 193
LLGL::VideoDisplayMode, 207
LLGL::Viewport, 210
Help
LLGL, 34
Home
LLGL, 34

I
LLGL, 35
Image.h, 225
ImageDescriptor
LLGL::ImageDescriptor, 97
ImageFormat
LLGL, 33
ImageFormatSize
LLGL, 48
ImproperArgument
LLGL, 47
ImproperState
LLGL, 47
Inc
LLGL::RenderingProfiler::Counter, 89
IncClamp
LLGL, 43
IncOccurrence
LLGL::RenderingDebugger::Message, 102
IncWrap
LLGL, 43
Index
LLGL, 32
index
LLGL::ConstantBufferViewDescriptor, 88
LLGL::StorageBufferViewDescriptor, 179
indexBuffer
LLGL::BufferDescriptor, 60
IndexBufferDesc
Global utility functions, especially to fill descriptor structures., 16
IndexFormat
LLGL::IndexFormat, 99
IndexFormat.h, 226
Input
LLGL::Input, 100
Input.h, 226
inputSlot
LLGL::VertexAttribute, 202
Insert
LLGL, 34
instanceDivisor
LLGL::VertexAttribute, 202
InstructionOnly
LLGL::ShaderDisassembleFlags, 157
Int
LLGL, 46, 47
Int16
LLGL, 33
Int2
LLGL, 46, 47
Int3
LLGL, 46, 47
Int32
LLGL, 33
Int4
LLGL, 46, 47
Int8
LLGL, 33
interval
LLGL::VsyncDescriptor, 211
InvBlendFactor
LLGL, 31
InvDestAlpha
LLGL, 31
InvDestColor
LLGL, 31
InvSrc1Alpha

LLGL, 31
InvSrc1Color
LLGL, 31
InvSrcAlpha
LLGL, 31
InvSrcColor
LLGL, 31
InvalidArgument
LLGL, 33
InvalidState
LLGL, 33
Invert
LLGL, 38, 43
invertFrontFace
LLGL::GraphicsAPIDependentStateDescriptor::↩
StateOpenGLDescriptor, 174
InvertedAND
LLGL, 38
InvertedCopy
LLGL, 37
InvertedOR
LLGL, 38
IsArrayTexture
LLGL, 49
IsBlocked
LLGL::RenderingDebugger::Message, 102
IsCompressedFormat
LLGL, 49
IsDepthStencilFormat
LLGL, 49
IsMultiSampleTexture
LLGL, 50
IsShown
LLGL::Window, 213

J
LLGL, 35

K
LLGL, 35
Keep
LLGL, 37, 43
Key
LLGL, 33
Key.h, 227
KeyDoubleClick
LLGL::Input, 100
KeyDown
LLGL::Input, 100
KeyPressed
LLGL::Input, 100
KeyUp
LLGL::Input, 100
Keypad0
LLGL, 35
Keypad1
LLGL, 35
Keypad2
LLGL, 35
Keypad3
LLGL, 35
Keypad4
LLGL, 35
Keypad5
LLGL, 35
Keypad6
LLGL, 35
Keypad7
LLGL, 35
Keypad8
LLGL, 35
Keypad9
LLGL, 35
KeypadDecimal
LLGL, 36
KeypadDivide
LLGL, 36
KeypadMinus
LLGL, 35
KeypadMultiply
LLGL, 35
KeypadPlus
LLGL, 35
KeypadSeparator
LLGL, 35

L
LLGL, 35
LButton
LLGL, 34
LControl
LLGL, 36
LLGL.h, 228
LLGL::BlendDescriptor, 55
blendEnabled, 55
blendFactor, 55
targets, 56
LLGL::BlendTargetDescriptor, 56
alphaArithmetic, 57
colorArithmetic, 57
colorMask, 57
destAlpha, 57
destColor, 57
srcAlpha, 57
srcColor, 57
LLGL::Buffer, 57
∼Buffer, 58
Buffer, 58
GetType, 58
operator=, 58
LLGL::BufferArray, 58
∼BufferArray, 59
BufferArray, 59
GetType, 59
operator=, 59
LLGL::BufferDescriptor, 59
flags, 60
indexBuffer, 60

size, 60
storageBuffer, 61
type, 61
vertexBuffer, 61
LLGL::BufferDescriptor::IndexBufferDescriptor, 98
format, 98
LLGL::BufferDescriptor::StorageBufferDescriptor, 178
storageType, 178
stride, 178
vectorType, 178
LLGL::BufferDescriptor::VertexBufferDescriptor, 203
format, 203
LLGL::BufferFlags, 61
DynamicUsage, 62
MapReadAccess, 62
MapWriteAccess, 62
LLGL::ClearFlags, 62
All, 63
Color, 63
ColorDepth, 63
Depth, 63
DepthStencil, 63
Stencil, 63
LLGL::Color
Cast, 64
Color, 64
components, 65
operator*=, 64
operator+=, 64
operator-, 65
operator-=, 65
operator/=, 65
operator[], 65
Ptr, 65
LLGL::Color< T, 3u >, 66
b, 68
Cast, 67
Color, 67
components, 68
g, 68
operator*=, 67
operator+=, 67
operator-, 67
operator-=, 67
operator/=, 67
operator[], 67, 68
Ptr, 68
r, 68
LLGL::Color< T, 4u >, 68
a, 71
b, 71
Cast, 70
Color, 70
components, 71
g, 71
operator*=, 70
operator+=, 70
operator-, 70
operator-=, 70
operator/=, 70
operator[], 71
Ptr, 71
r, 71
LLGL::Color< T, N >, 63
LLGL::CommandBuffer, 72
~CommandBuffer, 74
BeginQuery, 74
BeginRenderCondition, 74
BeginStreamOutput, 75
Clear, 75
CommandBuffer, 74
Dispatch, 76
Draw, 76
DrawIndexed, 76
DrawIndexedInstanced, 77
DrawInstanced, 77
EndQuery, 78
EndRenderCondition, 78
EndStreamOutput, 78
operator=, 78
QueryResult, 78
SetClearColor, 79
SetClearDepth, 79
SetClearStencil, 79
SetComputePipeline, 79
SetConstantBuffer, 79
SetConstantBufferArray, 80
SetGraphicsAPIDependentState, 80
SetGraphicsPipeline, 80
SetIndexBuffer, 80
SetRenderTarget, 81
SetSampler, 82
SetSamplerArray, 82
SetScissor, 82
SetScissorArray, 82
SetStorageBuffer, 83
SetStorageBufferArray, 83
SetStreamOutputBuffer, 83
SetStreamOutputBufferArray, 84
SetTexture, 84
SetTextureArray, 84
SetVertexBuffer, 84
SetVertexBufferArray, 85
SetViewport, 85
SetViewportArray, 85
SyncGPU, 86
LLGL::ComputePipeline, 86
~ComputePipeline, 86
LLGL::ComputePipelineDescriptor, 87
ComputePipelineDescriptor, 87
shaderProgram, 87
LLGL::ConstantBufferViewDescriptor, 88
index, 88
name, 88
size, 88
LLGL::DepthDescriptor, 90

compareOp, 90
testEnabled, 90
writeEnabled, 90
LLGL::Desktop, 52
GetColorDepth, 52
GetResolution, 52
ResetVideoMode, 52
SetVideoMode, 52
LLGL::GraphicsAPIDependentStateDescriptor, 93
GraphicsAPIDependentStateDescriptor, 93
stateOpenGL, 93
LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor, 173
invertFrontFace, 174
lineWidth, 174
logicOp, 174
screenSpaceOriginLowerLeft, 174
LLGL::GraphicsPipeline, 94
~GraphicsPipeline, 94
LLGL::GraphicsPipelineDescriptor, 94
blend, 95
depth, 95
primitiveTopology, 95
rasterizer, 95
shaderProgram, 95
stencil, 95
LLGL::ImageDescriptor, 96
buffer, 97
compressedSize, 97
dataType, 97
format, 97
GetElementSize, 97
ImageDescriptor, 97
LLGL::IndexFormat, 98
GetDataType, 99
GetFormatSize, 99
IndexFormat, 99
LLGL::Input, 99
GetEnteredChars, 100
GetMouseMotion, 100
GetMousePosition, 100
GetWheelMotion, 100
Input, 100
KeyDoubleClick, 100
KeyDown, 100
KeyPressed, 100
KeyUp, 100
LLGL::Log, 53
SetStdErr, 53
SetStdOut, 53
StdErr, 53
StdOut, 53
LLGL::MultiSamplingDescriptor, 103
enabled, 103
MultiSamplingDescriptor, 103
SampleCount, 103
samples, 103
LLGL::NativeContextHandle, 104
colorMap, 104
display, 104
parentWindow, 104
screen, 104
visual, 104
LLGL::NativeHandle, 105
display, 105
visual, 105
window, 105
LLGL::ProfileOpenGLDescriptor, 105
coreProfile, 106
debugDump, 106
extProfile, 106
version, 106
LLGL::Query, 107
~Query, 107
GetType, 107
operator=, 107
Query, 107
LLGL::QueryDescriptor, 108
QueryDescriptor, 108
renderCondition, 108
type, 108
LLGL::RasterizerDescriptor, 109
antiAliasedLineEnabled, 109
conservativeRasterization, 109
cullMode, 109
depthBias, 110
depthBiasClamp, 110
depthClampEnabled, 110
frontCCW, 110
multiSampling, 110
polygonMode, 110
scissorTestEnabled, 110
slopeScaledDepthBias, 110
LLGL::RenderContext, 110
~RenderContext, 111
GetVideoMode, 111
GetWindow, 111
operator=, 112
Present, 112
RenderContext, 111
SetOrCreateWindow, 112
SetVideoMode, 112
SetVsync, 112
ShareWindowAndVideoMode, 112
LLGL::RenderContextDescriptor, 113
debugCallback, 113
multiSampling, 113
profileOpenGL, 113
videoMode, 114
vsync, 114
LLGL::RenderSystem, 130
~RenderSystem, 133
AssertCreateBuffer, 133
AssertCreateBufferArray, 133
AssertCreateSamplerArray, 134
AssertCreateTextureArray, 134

CreateBuffer, 134
CreateBufferArray, 134
CreateCommandBuffer, 135
CreateComputePipeline, 135
CreateGraphicsPipeline, 135
CreateQuery, 135
CreateRenderContext, 135
CreateRenderTarget, 136
CreateSampler, 136
CreateSamplerArray, 136
CreateShader, 137
CreateShaderProgram, 137
CreateTexture, 137
CreateTextureArray, 137
FindModules, 138
GenerateMips, 138
GetConfiguration, 138
GetDefaultTextureImageRGBAub, 138
GetName, 138
GetRendererInfo, 138
GetRenderingCaps, 139
Load, 139
MapBuffer, 140
operator=, 140
QueryTextureDescriptor, 140
ReadTexture, 140
Release, 141, 142
RenderSystem, 133
SetConfiguration, 142
SetRendererInfo, 142
SetRenderingCaps, 142
UnmapBuffer, 143
WriteBuffer, 143
WriteTexture, 143
LLGL::RenderSystemConfiguration, 144
defaultImageColor, 144
threadCount, 144
LLGL::RenderTarget, 145
~RenderTarget, 145
ApplyMipResolution, 145
ApplyResolution, 145
AttachDepthBuffer, 145
AttachDepthStencilBuffer, 146
AttachStencilBuffer, 146
AttachTexture, 146
DetachAll, 147
GetResolution, 147
ResetResolution, 147
LLGL::RenderTargetAttachmentDescriptor, 147
cubeFace, 148
layer, 148
mipLevel, 148
LLGL::RenderTargetDescriptor, 148
customMultiSampling, 149
multiSampling, 149
LLGL::RendererID, 114
Direct3D11, 115
Direct3D12, 115
OpenGL, 115
Reserved5, 115
Reserved6, 115
Reserved7, 115
Reserved8, 115
Reserved9, 115
ReservedA, 115
ReservedB, 116
ReservedC, 116
ReservedD, 116
ReservedE, 116
ReservedF, 116
Vulkan, 116
LLGL::RendererInfo, 116
deviceName, 117
rendererID, 117
rendererName, 117
shadingLanguageName, 117
vendorName, 117
LLGL::RenderingCaps, 118
clippingRange, 119
has3DTextures, 119
hasComputeShaders, 119
hasConservativeRasterization, 120
hasConstantBuffers, 120
hasCubeTextureArrays, 120
hasCubeTextures, 120
hasGeometryShaders, 120
hasInstancing, 120
hasMultiSampleTextures, 121
hasOffsetInstancing, 121
hasRenderTargets, 121
hasSamplers, 121
hasStorageBuffers, 121
hasStreamOutputs, 121
hasTessellationShaders, 122
hasTextureArrays, 122
hasUniforms, 122
hasViewportArrays, 122
max1DTextureSize, 122
max2DTextureSize, 122
max3DTextureSize, 122
maxAnisotropy, 123
maxComputeShaderWorkGroupSize, 123
maxConstantBufferSize, 123
maxCubeTextureSize, 123
maxNumComputeShaderWorkGroups, 123
maxNumRenderTargetAttachments, 123
maxNumTextureArrayLayers, 123
maxPatchVertices, 123
screenOrigin, 123
shadingLanguage, 124
LLGL::RenderingDebugger, 124
~RenderingDebugger, 125
OnError, 125
OnWarning, 125
PostError, 125
PostWarning, 125

RenderingDebugger, 125
SetSource, 125
LLGL::RenderingDebugger::Message, 101
Block, 102
BlockAfter, 102
GetOccurrences, 102
GetSource, 102
GetText, 102
IncOccurrence, 102
IsBlocked, 102
Message, 102
operator=, 102
RenderingDebugger, 102
LLGL::RenderingProfiler, 126
dispatchComputeCalls, 127
drawCalls, 127
mapBuffer, 128
RecordDrawCall, 127
renderedLines, 128
renderedPatches, 128
renderedPoints, 128
renderedTriangles, 128
ResetCounters, 127
setComputePipeline, 128
setConstantBuffer, 128
setGraphicsPipeline, 129
setIndexBuffer, 129
setRenderTarget, 129
setSampler, 129
setStorageBuffer, 129
setStreamOutputBuffer, 129
setTexture, 130
setVertexBuffer, 130
writeBuffer, 130
LLGL::RenderingProfiler::Counter, 88
Count, 89
Inc, 89
operator unsigned int, 89
Reset, 89
ValueType, 89
LLGL::Sampler, 149
∼Sampler, 150
operator=, 150
Sampler, 150
LLGL::SamplerArray, 150
∼SamplerArray, 150
operator=, 150
SamplerArray, 150
LLGL::SamplerDescriptor, 151
borderColor, 151
compareOp, 151
depthCompare, 152
magFilter, 152
maxAnisotropy, 152
maxLOD, 152
minFilter, 152
minLOD, 152
mipMapFilter, 152
mipMapLODBias, 152
mipMapping, 152
textureWrapU, 152
textureWrapV, 153
textureWrapW, 153
LLGL::Scissor, 153
height, 154
Scissor, 154
width, 154
x, 154
y, 154
LLGL::Shader, 154
∼Shader, 155
Compile, 155
Disassemble, 155
GetType, 155
operator=, 156
QueryInfoLog, 156
Shader, 155
LLGL::ShaderCompileFlags, 156
Debug, 156
O1, 156
O2, 156
O3, 156
WarnError, 156
LLGL::ShaderDisassembleFlags, 157
InstructionOnly, 157
LLGL::ShaderProgram, 157
∼ShaderProgram, 159
AttachShader, 159
BindConstantBuffer, 160
BindStorageBuffer, 160
BuildInputLayout, 161
DetachAll, 161
LinkShaders, 162
LockShaderUniform, 162
operator=, 162
QueryConstantBuffers, 163
QueryInfoLog, 163
QueryStorageBuffers, 163
QueryStreamOutputAttributes, 163
QueryUniforms, 163
QueryVertexAttributes, 163
ShaderProgram, 159
UnlockShaderUniform, 163
LLGL::ShaderSource, 164
ShaderSource, 165, 166
sourceCode, 167
sourceHLSL, 167
streamOutput, 167
LLGL::ShaderSource::SourceHLSL, 172
entryPoint, 173
flags, 173
target, 173
LLGL::ShaderSource::StreamOutput, 180
format, 180
LLGL::ShaderStageFlags, 167
AllGraphicsStages, 168

AllStages, 168
AllTessStages, 168
ComputeStage, 168
FragmentStage, 168
GeometryStage, 168
ReadOnlyResource, 168
TessControlStage, 168
TessEvaluationStage, 168
VertexStage, 168
LLGL::ShaderUniform, 169
~ShaderUniform, 170
SetUniform, 170, 171
SetUniformArray, 171, 172
LLGL::StencilDescriptor, 175
back, 175
front, 175
testEnabled, 175
LLGL::StencilFaceDescriptor, 176
compareOp, 176
depthFailOp, 176
depthPassOp, 176
readMask, 176
reference, 177
stencilFailOp, 177
writeMask, 177
LLGL::StorageBufferViewDescriptor, 179
index, 179
name, 179
type, 179
LLGL::StreamOutputAttribute, 180
components, 181
name, 181
operator=, 181
outputSlot, 181
semanticIndex, 181
startComponent, 182
stream, 182
StreamOutputAttribute, 181
LLGL::StreamOutputFormat, 182
AppendAttribute, 182
AppendAttributes, 183
attributes, 183
LLGL::SubTextureDescriptor, 183
~SubTextureDescriptor, 184
mipLevel, 184
SubTextureDescriptor, 184
texture1D, 184
texture2D, 184
texture3D, 184
textureCube, 185
LLGL::SubTextureDescriptor::Texture1DDescriptor, 186
layerOffset, 186
layers, 186
width, 186
x, 186
LLGL::SubTextureDescriptor::Texture2DDescriptor, 188
height, 188
layerOffset, 188
layers, 189
width, 189
x, 189
y, 189
LLGL::SubTextureDescriptor::Texture3DDescriptor, 191
depth, 191
height, 191
width, 192
x, 192
y, 192
z, 192
LLGL::SubTextureDescriptor::TextureCubeDescriptor, 194
cubeFaceOffset, 194
cubeFaces, 194
height, 194
layerOffset, 194
width, 194
x, 194
y, 195
LLGL::Texture, 185
~Texture, 185
GetType, 185
operator=, 185
QueryMipLevelSize, 185
Texture, 185
LLGL::TextureArray, 192
~TextureArray, 193
operator=, 193
TextureArray, 193
LLGL::TextureDescriptor, 195
~TextureDescriptor, 196
format, 196
texture1D, 196
texture2DMS, 196
texture2D, 196
texture3D, 196
textureCube, 196
TextureDescriptor, 196
type, 196
LLGL::TextureDescriptor::Texture1DDescriptor, 187
layers, 187
width, 187
LLGL::TextureDescriptor::Texture2DDescriptor, 187
height, 188
layers, 188
width, 188
LLGL::TextureDescriptor::Texture2DMSDescriptor, 189
fixedSamples, 190
height, 190
layers, 190
samples, 190
width, 190
LLGL::TextureDescriptor::Texture3DDescriptor, 190
depth, 191
height, 191
width, 191
LLGL::TextureDescriptor::TextureCubeDescriptor, 193

height, 193
layers, 193
width, 193
LLGL::Timer, 197
~Timer, 198
Create, 198
FrameCount, 198
GetDeltaTime, 198
GetFrameCount, 198
GetFrequency, 198
MeasureTime, 198
ResetFrameCounter, 199
Start, 199
Stop, 199
LLGL::UniformDescriptor, 199
location, 200
name, 200
size, 200
type, 200
LLGL::Version, 53
GetID, 54
GetMajor, 54
GetMinor, 54
GetRevision, 54
GetStatus, 54
GetString, 54
LLGL::VertexAttribute, 200
conversion, 202
GetSize, 202
inputSlot, 202
instanceDivisor, 202
name, 202
offset, 202
operator=, 202
semanticIndex, 202
vectorType, 203
VertexAttribute, 201
LLGL::VertexFormat, 204
AppendAttribute, 204
AppendAttributes, 205
attributes, 205
OffsetAppend, 205
stride, 205
LLGL::VideoAdapterDescriptor, 206
name, 206
outputs, 206
vendor, 206
videoMemory, 207
LLGL::VideoDisplayMode, 207
height, 207
refreshRate, 207
width, 207
LLGL::VideoModeDescriptor, 208
colorDepth, 208
fullscreen, 208
resolution, 208
swapChainMode, 208
LLGL::VideoOutput, 209
displayModes, 209
LLGL::Viewport, 209
height, 210
maxDepth, 210
minDepth, 210
Viewport, 210
width, 210
x, 211
y, 211
LLGL::VsyncDescriptor, 211
enabled, 211
interval, 211
refreshRate, 212
LLGL::Window, 212
~Window, 213
AddEventListener, 213
Create, 213
GetNativeHandle, 213
GetPosition, 213
GetSize, 213
GetTitle, 213
IsShown, 213
OnProcessEvents, 213
PostChar, 213
PostDoubleClick, 214
PostGlobalMotion, 214
PostKeyDown, 214
PostKeyUp, 214
PostLocalMotion, 214
PostQuit, 214
PostResize, 214
PostWheelMotion, 214
ProcessEvents, 214
QueryDesc, 214
Recreate, 214
RemoveEventListener, 214
SetDesc, 215
SetPosition, 215
SetSize, 215
SetTitle, 215
Show, 215
LLGL::Window::EventListener, 91
~EventListener, 92
OnChar, 92
OnDoubleClick, 92
OnGlobalMotion, 92
OnKeyDown, 92
OnKeyUp, 92
OnLocalMotion, 92
OnProcessEvents, 92
OnQuit, 92
OnResize, 92
OnWheelMotion, 92
Window, 92
LLGL::WindowDescriptor, 215
acceptDropFiles, 216
borderless, 216
centered, 216

position, 216
preventForPowerSafe, 216
resizable, 216
size, 216
title, 216
visible, 216
windowContext, 216
LLGL_EXPORT
Export.h, 222
LLGL, 19
A, 35
AND, 38
Add, 30
AnySamplesPassed, 41
AnySamplesPassedConservative, 41
AppendStructuredBuffer, 43
Apps, 35
Attn, 37
AxisDirection, 30
B, 35
BGRA, 33
BGR, 33
Back, 32, 34
BlendArithmetic, 30
BlendFactor, 31
BlendOp, 30
Border, 46
BrowserBack, 36
BrowserFavorits, 36
BrowserForward, 36
BrowserHome, 36
BrowserRefresh, 36
BrowserSearch, 36
BrowserStop, 36
Buffer, 43
BufferCPUAccess, 31
BufferType, 31
ByRegionNoWait, 41
ByRegionNoWaitInverted, 41
ByRegionWait, 41
ByRegionWaitInverted, 41
ByteAddressBuffer, 43
ByteBuffer, 28
C, 35
Cancel, 34
Capital, 34
Clamp, 46
Clear, 34, 37
ClippingInputPrimitives, 41
ClippingOutputPrimitives, 41
ClippingRange, 32
ColorRGBAb, 29
ColorRGBAd, 29
ColorRGBAf, 29
ColorRGBAT, 29
ColorRGBAub, 29
ColorRGBA, 29
ColorRGBb, 29
ColorRGBd, 29
ColorRGBf, 29
ColorRGBT, 29
ColorRGBub, 29
ColorRGB, 29
Comma, 37
CompareOp, 32
CompareSWO, 48
CompressedRGBA, 33
CompressedRGB, 33
Compute, 42
ComputeShaderInvocations, 41
Constant, 32
ConsumeStructuredBuffer, 43
Control, 34
ConvertImageBuffer, 48
Copy, 37
CrSel, 37
CullMode, 32
D, 35
D0, 34
D1, 34
D2, 34
D3, 34
D4, 34
D5, 34
D6, 34
D7, 34
D8, 34
D9, 35
DataType, 32
DataTypeSize, 48
DebugCallback, 29
DecClamp, 43
DecWrap, 43
Delete, 34
Depth, 33
DepthComponent, 44
DepthStencil, 33, 44
DestAlpha, 31
DestColor, 31
Disabled, 32, 37
Double, 33, 46, 47
Double2, 46, 47
Double2x2, 47
Double3, 46, 47
Double3x3, 47
Double4, 46, 47
Double4x4, 47
DoubleBuffering, 44
Down, 34
E, 35
End, 34
Equal, 32
Equiv, 38
ErEOF, 37
ErrorType, 33
Escape, 34

Ever, 32
ExSel, 37
Exe, 34
Exponent, 37
F, 35
F1, 36
F10, 36
F11, 36
F12, 36
F13, 36
F14, 36
F15, 36
F16, 36
F17, 36
F18, 36
F19, 36
F2, 36
F20, 36
F21, 36
F22, 36
F23, 36
F24, 36
F3, 36
F4, 36
F5, 36
F6, 36
F7, 36
F8, 36
F9, 36
Fill, 38
Float, 33, 46, 47
Float2, 46, 47
Float2x2, 47
Float3, 46, 47
Float3x3, 47
Float4, 46, 47
Float4x4, 47
Fragment, 42
FragmentShaderInvocations, 41
Front, 32
G, 35
GLSL_110, 42
GLSL_120, 42
GLSL_130, 42
GLSL_140, 42
GLSL_150, 42
GLSL_330, 42
GLSL_400, 42
GLSL_410, 42
GLSL_420, 42
GLSL_430, 42
GLSL_440, 42
GLSL_450, 42
Geometry, 42
GeometryPrimitivesGenerated, 41
GeometryShaderInvocations, 41
Greater, 32
GreaterEqual, 32
H, 35
HLSL_2_0, 42
HLSL_2_0a, 42
HLSL_2_0b, 42
HLSL_3_0, 42
HLSL_4_0, 42
HLSL_4_1, 42
HLSL_5_0, 42
Help, 34
Home, 34
I, 35
ImageFormat, 33
ImageFormatSize, 48
ImproperArgument, 47
ImproperState, 47
IncClamp, 43
IncWrap, 43
Index, 32
Insert, 34
Int, 46, 47
Int16, 33
Int2, 46, 47
Int3, 46, 47
Int32, 33
Int4, 46, 47
Int8, 33
InvBlendFactor, 31
InvDestAlpha, 31
InvDestColor, 31
InvSrc1Alpha, 31
InvSrc1Color, 31
InvSrcAlpha, 31
InvSrcColor, 31
InvalidArgument, 33
InvalidState, 33
Invert, 38, 43
InvertedAND, 38
InvertedCopy, 37
InvertedOR, 38
IsArrayTexture, 49
IsCompressedFormat, 49
IsDepthStencilFormat, 49
IsMultiSampleTexture, 50
J, 35
K, 35
Keep, 37, 43
Key, 33
Keypad0, 35
Keypad1, 35
Keypad2, 35
Keypad3, 35
Keypad4, 35
Keypad5, 35
Keypad6, 35
Keypad7, 35
Keypad8, 35
Keypad9, 35
KeypadDecimal, 36

KeypadDivide, 36
KeypadMinus, 35
KeypadMultiply, 35
KeypadPlus, 35
KeypadSeparator, 35
L, 35
LButton, 34
LControl, 36
LMenu, 36
LShift, 36
LWin, 35
LaunchApp1, 37
LaunchApp2, 37
LaunchMail, 37
LaunchMediaSelect, 37
Left, 34
Less, 32
LessEqual, 32
LineList, 39
LineListAdjacency, 39
LineLoop, 39
LineStrip, 39
LineStripAdjacency, 39
Linear, 44
Lines, 40
LogicOp, 37
LowerLeft, 42
M, 35
MButton, 34
Max, 30
MaxColorValue, 50
MaxColorValue< bool >, 50
MaxColorValue< unsigned char >, 50
MediaNextTrack, 36
MediaPlayPause, 37
MediaPrevTrack, 37
MediaStop, 37
Menu, 34
Min, 30
Minus, 37
MinusOneToOne, 32
Mirror, 46
MirrorOnce, 46
N, 35
NAND, 38
NOR, 38
Nearest, 44
Never, 32
NoName, 37
NoWait, 41
NoWaitInverted, 41
Noop, 38
NotEqual, 32
NumLock, 36
NumMipLevels, 50
O, 35
OEMClear, 37
One, 31
OpenGL_1_0, 38
OpenGL_1_1, 38
OpenGL_1_2, 38
OpenGL_1_3, 38
OpenGL_1_4, 38
OpenGL_1_5, 38
OpenGL_2_0, 38
OpenGL_2_1, 38
OpenGL_3_0, 38
OpenGL_3_1, 38
OpenGL_3_2, 38
OpenGL_3_3, 38
OpenGL_4_0, 38
OpenGL_4_1, 38
OpenGL_4_2, 38
OpenGL_4_3, 38
OpenGL_4_4, 38
OpenGL_4_5, 38
OpenGL_Latest, 38
OpenGLVersion, 38
operator!=, 50, 51
operator*, 51
operator+, 51
operator-, 51
operator/, 51
operator==, 51
OR, 38
P, 35
PA1, 37
PageDown, 34
PageUp, 34
Patches1, 39
Patches10, 39
Patches11, 39
Patches12, 39
Patches13, 39
Patches14, 39
Patches15, 39
Patches16, 39
Patches17, 39
Patches18, 39
Patches19, 39
Patches2, 39
Patches20, 39
Patches21, 39
Patches22, 39
Patches23, 40
Patches24, 40
Patches25, 40
Patches26, 40
Patches27, 40
Patches28, 40
Patches29, 40
Patches3, 39
Patches30, 40
Patches31, 40
Patches32, 40
Patches4, 39

Patches5, 39
Patches6, 39
Patches7, 39
Patches8, 39
Patches9, 39
Pause, 34
Period, 37
Play, 37
Plus, 37
Point, 29
PointList, 39
PointlessOperation, 47
Points, 38, 40
PolygonMode, 38
PrimitiveTopology, 38
PrimitiveType, 40
PrimitivesGenerated, 41
PrimitivesSubmitted, 41
Print, 34
Q, 35
QueryType, 40
R, 33, 35, 44
R16, 44
R16Float, 44
R16Sgn, 44
R32Float, 44
R32SInt, 44
R32UInt, 44
R8, 44
R8Sgn, 44
RButton, 34
RControl, 36
RG16, 45
RG16Float, 45
RG16Sgn, 45
RG32Float, 45
RG32SInt, 45
RG32UInt, 45
RG8, 44
RG8Sgn, 45
RGB16, 45
RGB16Float, 45
RGB16Sgn, 45
RGB32Float, 45
RGB32SInt, 45
RGB32UInt, 45
RGB8, 45
RGB8Sgn, 45
RGB_DXT1, 45
RGBA16, 45
RGBA16Float, 45
RGBA16Sgn, 45
RGBA32Float, 45
RGBA32SInt, 45
RGBA32UInt, 45
RGBA8, 45
RGBA8Sgn, 45
RGBA_DXT1, 45
RGBA_DXT3, 45
RGBA_DXT5, 45
RGBA, 33, 44
RGB, 33, 44
RMenu, 36
RShift, 36
RWBuffer, 43
RWByteAddressBuffer, 43
RWStructuredBuffer, 43
RWin, 35
ReadOnly, 31
ReadWrite, 31
RenderConditionMode, 41
Repeat, 46
Replace, 43
Return, 34
RevSubtract, 30
ReverseAND, 38
ReverseOR, 38
RG, 33, 44
Right, 34
S, 35
Sampler1D, 47
Sampler2D, 47
Sampler3D, 47
SamplerCube, 47
SamplesPassed, 41
ScreenOrigin, 41
ScrollLock, 36
Select, 34
Set, 37
ShaderType, 42
ShadingLanguage, 42
Shift, 34
SingleBuffering, 44
Size, 30
Sleep, 35
Snapshot, 34
Space, 34
Src1Alpha, 31
Src1Color, 31
SrcAlpha, 31
SrcAlphaSaturate, 31
SrcColor, 31
StencilOp, 42
Storage, 32
StorageBufferType, 43
StreamOutOverflow, 41
StreamOutPrimitivesWritten, 41
StreamOutput, 32
StructuredBuffer, 43
Subtract, 30
SwapChainMode, 43
T, 35
Tab, 34
TessControl, 42
TessControlShaderInvocations, 41
TessEvaluation, 42

TessEvaluationShaderInvocations, 41
Texture1DArray, 46
Texture1D, 46
Texture2DArray, 46
Texture2DMSArray, 46
Texture2DMS, 46
Texture2D, 46
Texture3D, 46
TextureCube, 46
TextureCubeArray, 46
TextureFilter, 44
TextureFormat, 44
TextureType, 45
TextureWrap, 46
TimeElapsed, 41
TriangleFan, 39
TriangleList, 39
TriangleListAdjacency, 39
TriangleStrip, 39
TriangleStripAdjacency, 39
Triangles, 40
TripleBuffering, 44
U, 35
UInt, 47
UInt16, 33
UInt2, 47
UInt3, 47
UInt32, 33
UInt4, 47
UInt8, 33
UniformType, 46
Unknown, 44
Unsupported, 42
UnsupportedFeature, 33
Up, 34
UpperLeft, 42
V, 35
VectorType, 47
VectorTypeFormat, 51
VectorTypeSize, 52
Vertex, 32, 42
VertexShaderInvocations, 41
VerticesSubmitted, 41
VolumeDown, 36
VolumeMute, 36
VolumeUp, 36
W, 35
Wait, 41
WaitInverted, 41
WarningType, 47
Wireframe, 38
WriteOnly, 31
X, 35
XButton1, 34
XButton2, 34
XNeg, 30
XOR, 38
XPos, 30
Y, 35
YNeg, 30
YPos, 30
Z, 35
ZNeg, 30
ZPos, 30
Zero, 31, 43
ZeroToOne, 32
Zoom, 37
LMenu
LLGL, 36
LShift
LLGL, 36
LWin
LLGL, 35
LaunchApp1
LLGL, 37
LaunchApp2
LLGL, 37
LaunchMail
LLGL, 37
LaunchMediaSelect
LLGL, 37
layer
LLGL::RenderTargetAttachmentDescriptor, 148
layerOffset
LLGL::SubTextureDescriptor::Texture1DDescriptor, 186
LLGL::SubTextureDescriptor::Texture2DDescriptor, 188
LLGL::SubTextureDescriptor::TextureCube↩ Descriptor, 194
layers
LLGL::SubTextureDescriptor::Texture1DDescriptor, 186
LLGL::SubTextureDescriptor::Texture2DDescriptor, 189
LLGL::TextureDescriptor::Texture1DDescriptor, 187
LLGL::TextureDescriptor::Texture2DDescriptor, 188
LLGL::TextureDescriptor::Texture2DMSDescriptor, 190
LLGL::TextureDescriptor::TextureCubeDescriptor, 193
Left
LLGL, 34
Less
LLGL, 32
LessEqual
LLGL, 32
LineList
LLGL, 39
LineListAdjacency
LLGL, 39
LineLoop
LLGL, 39
LineStrip

LLGL, 39
LineStripAdjacency
LLGL, 39
lineWidth
LLGL::GraphicsAPIDependentStateDescriptor::↩
StateOpenGLDescriptor, 174
Linear
LLGL, 44
Lines
LLGL, 40
LinkShaders
LLGL::ShaderProgram, 162
LinuxNativeHandle.h, 228
Load
LLGL::RenderSystem, 139
location
LLGL::UniformDescriptor, 200
LockShaderUniform
LLGL::ShaderProgram, 162
Log.h, 228
LogicOp
LLGL, 37
logicOp
LLGL::GraphicsAPIDependentStateDescriptor::↩
StateOpenGLDescriptor, 174
LowerLeft
LLGL, 42

M
LLGL, 35
MButton
LLGL, 34
MacOSNativeHandle.h, 229
magFilter
LLGL::SamplerDescriptor, 152
MapBuffer
LLGL::RenderSystem, 140
mapBuffer
LLGL::RenderingProfiler, 128
MapReadAccess
LLGL::BufferFlags, 62
MapWriteAccess
LLGL::BufferFlags, 62
Max
LLGL, 30
max1DTextureSize
LLGL::RenderingCaps, 122
max2DTextureSize
LLGL::RenderingCaps, 122
max3DTextureSize
LLGL::RenderingCaps, 122
maxAnisotropy
LLGL::RenderingCaps, 123
LLGL::SamplerDescriptor, 152
MaxColorValue
LLGL, 50
MaxColorValue< bool >
LLGL, 50
MaxColorValue< unsigned char >
LLGL, 50
maxComputeShaderWorkGroupSize
LLGL::RenderingCaps, 123
maxConstantBufferSize
LLGL::RenderingCaps, 123
maxCubeTextureSize
LLGL::RenderingCaps, 123
maxDepth
LLGL::Viewport, 210
maxLOD
LLGL::SamplerDescriptor, 152
maxNumComputeShaderWorkGroups
LLGL::RenderingCaps, 123
maxNumRenderTargetAttachments
LLGL::RenderingCaps, 123
maxNumTextureArrayLayers
LLGL::RenderingCaps, 123
maxPatchVertices
LLGL::RenderingCaps, 123
MeasureTime
LLGL::Timer, 198
MediaNextTrack
LLGL, 36
MediaPlayPause
LLGL, 37
MediaPrevTrack
LLGL, 37
MediaStop
LLGL, 37
Menu
LLGL, 34
Message
LLGL::RenderingDebugger::Message, 102
Min
LLGL, 30
minDepth
LLGL::Viewport, 210
minFilter
LLGL::SamplerDescriptor, 152
minLOD
LLGL::SamplerDescriptor, 152
Minus
LLGL, 37
MinusOneToOne
LLGL, 32
mipLevel
LLGL::RenderTargetAttachmentDescriptor, 148
LLGL::SubTextureDescriptor, 184
mipMapFilter
LLGL::SamplerDescriptor, 152
mipMapLODBias
LLGL::SamplerDescriptor, 152
mipMapping
LLGL::SamplerDescriptor, 152
Mirror
LLGL, 46
MirrorOnce
LLGL, 46

multiSampling
LLGL::RasterizerDescriptor, 110
LLGL::RenderContextDescriptor, 113
LLGL::RenderTargetDescriptor, 149
MultiSamplingDescriptor
LLGL::MultiSamplingDescriptor, 103

N
LLGL, 35
NAND
LLGL, 38
NOR
LLGL, 38
name
LLGL::ConstantBufferViewDescriptor, 88
LLGL::StorageBufferViewDescriptor, 179
LLGL::StreamOutputAttribute, 181
LLGL::UniformDescriptor, 200
LLGL::VertexAttribute, 202
LLGL::VideoAdapterDescriptor, 206
NativeHandle.h, 229
Nearest
LLGL, 44
Never
LLGL, 32
NoName
LLGL, 37
NoWait
LLGL, 41
NoWaitInverted
LLGL, 41
Noop
LLGL, 38
NotEqual
LLGL, 32
NumLock
LLGL, 36
NumMipLevels
LLGL, 50

O
LLGL, 35
O1
LLGL::ShaderCompileFlags, 156
O2
LLGL::ShaderCompileFlags, 156
O3
LLGL::ShaderCompileFlags, 156
OEMClear
LLGL, 37
offset
LLGL::VertexAttribute, 202
OffsetAppend
LLGL::VertexFormat, 205
OnChar
LLGL::Window::EventListener, 92
OnDoubleClick
LLGL::Window::EventListener, 92
OnError
LLGL::RenderingDebugger, 125
OnGlobalMotion
LLGL::Window::EventListener, 92
OnKeyDown
LLGL::Window::EventListener, 92
OnKeyUp
LLGL::Window::EventListener, 92
OnLocalMotion
LLGL::Window::EventListener, 92
OnProcessEvents
LLGL::Window, 213
LLGL::Window::EventListener, 92
OnQuit
LLGL::Window::EventListener, 92
OnResize
LLGL::Window::EventListener, 92
OnWarning
LLGL::RenderingDebugger, 125
OnWheelMotion
LLGL::Window::EventListener, 92
One
LLGL, 31
OpenGL_1_0
LLGL, 38
OpenGL_1_1
LLGL, 38
OpenGL_1_2
LLGL, 38
OpenGL_1_3
LLGL, 38
OpenGL_1_4
LLGL, 38
OpenGL_1_5
LLGL, 38
OpenGL_2_0
LLGL, 38
OpenGL_2_1
LLGL, 38
OpenGL_3_0
LLGL, 38
OpenGL_3_1
LLGL, 38
OpenGL_3_2
LLGL, 38
OpenGL_3_3
LLGL, 38
OpenGL_4_0
LLGL, 38
OpenGL_4_1
LLGL, 38
OpenGL_4_2
LLGL, 38
OpenGL_4_3
LLGL, 38
OpenGL_4_4
LLGL, 38
OpenGL_4_5
LLGL, 38

OpenGL_Latest
LLGL, 38
OpenGLVersion
LLGL, 38
OpenGL
LLGL::RendererID, 115
operator unsigned int
LLGL::RenderingProfiler::Counter, 89
operator!=
LLGL, 50, 51
operator∗
LLGL, 51
operator∗=
LLGL::Color, 64
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
operator+
LLGL, 51
operator+=
LLGL::Color, 64
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
operator-
LLGL::Color, 65
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
LLGL, 51
operator-=
LLGL::Color, 65
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
operator/
LLGL, 51
operator/=
LLGL::Color, 65
LLGL::Color< T, 3u >, 67
LLGL::Color< T, 4u >, 70
operator=
LLGL::Buffer, 58
LLGL::BufferArray, 59
LLGL::CommandBuffer, 78
LLGL::Query, 107
LLGL::RenderContext, 112
LLGL::RenderSystem, 140
LLGL::RenderingDebugger::Message, 102
LLGL::Sampler, 150
LLGL::SamplerArray, 150
LLGL::Shader, 156
LLGL::ShaderProgram, 162
LLGL::StreamOutputAttribute, 181
LLGL::Texture, 185
LLGL::TextureArray, 193
LLGL::VertexAttribute, 202
operator==
LLGL, 51
operator[]
LLGL::Color, 65
LLGL::Color< T, 3u >, 67, 68
LLGL::Color< T, 4u >, 71
OR
LLGL, 38
outputSlot
LLGL::StreamOutputAttribute, 181
outputs
LLGL::VideoAdapterDescriptor, 206

P
LLGL, 35
PA1
LLGL, 37
PageDown
LLGL, 34
PageUp
LLGL, 34
parentWindow
LLGL::NativeContextHandle, 104
Patches1
LLGL, 39
Patches10
LLGL, 39
Patches11
LLGL, 39
Patches12
LLGL, 39
Patches13
LLGL, 39
Patches14
LLGL, 39
Patches15
LLGL, 39
Patches16
LLGL, 39
Patches17
LLGL, 39
Patches18
LLGL, 39
Patches19
LLGL, 39
Patches2
LLGL, 39
Patches20
LLGL, 39
Patches21
LLGL, 39
Patches22
LLGL, 39
Patches23
LLGL, 40
Patches24
LLGL, 40
Patches25
LLGL, 40
Patches26
LLGL, 40
Patches27
LLGL, 40
Patches28

LLGL, 40
Patches29
LLGL, 40
Patches3
LLGL, 39
Patches30
LLGL, 40
Patches31
LLGL, 40
Patches32
LLGL, 40
Patches4
LLGL, 39
Patches5
LLGL, 39
Patches6
LLGL, 39
Patches7
LLGL, 39
Patches8
LLGL, 39
Patches9
LLGL, 39
Pause
LLGL, 34
Period
LLGL, 37
Play
LLGL, 37
Plus
LLGL, 37
Point
LLGL, 29
PointList
LLGL, 39
PointlessOperation
LLGL, 47
Points
LLGL, 38, 40
PolygonMode
LLGL, 38
polygonMode
LLGL::RasterizerDescriptor, 110
position
LLGL::WindowDescriptor, 216
PostChar
LLGL::Window, 213
PostDoubleClick
LLGL::Window, 214
PostError
LLGL::RenderingDebugger, 125
PostGlobalMotion
LLGL::Window, 214
PostKeyDown
LLGL::Window, 214
PostKeyUp
LLGL::Window, 214
PostLocalMotion
LLGL::Window, 214
PostQuit
LLGL::Window, 214
PostResize
LLGL::Window, 214
PostWarning
LLGL::RenderingDebugger, 125
PostWheelMotion
LLGL::Window, 214
Present
LLGL::RenderContext, 112
preventForPowerSafe
LLGL::WindowDescriptor, 216
PrimitiveTopology
LLGL, 38
primitiveTopology
LLGL::GraphicsPipelineDescriptor, 95
PrimitiveType
LLGL, 40
PrimitivesGenerated
LLGL, 41
PrimitivesSubmitted
LLGL, 41
Print
LLGL, 34
ProcessEvents
LLGL::Window, 214
profileOpenGL
LLGL::RenderContextDescriptor, 113
Ptr
LLGL::Color, 65
LLGL::Color< T, 3u >, 68
LLGL::Color< T, 4u >, 71

Q
LLGL, 35
Query
LLGL::Query, 107
Query.h, 229
QueryConstantBuffers
LLGL::ShaderProgram, 163
QueryDesc
LLGL::Window, 214
QueryDescriptor
LLGL::QueryDescriptor, 108
QueryFlags.h, 230
QueryInfoLog
LLGL::Shader, 156
LLGL::ShaderProgram, 163
QueryMipLevelSize
LLGL::Texture, 185
QueryResult
LLGL::CommandBuffer, 78
QueryStorageBuffers
LLGL::ShaderProgram, 163
QueryStreamOutputAttributes
LLGL::ShaderProgram, 163
QueryTextureDescriptor
LLGL::RenderSystem, 140

QueryType
LLGL, 40
QueryUniforms
LLGL::ShaderProgram, 163
QueryVertexAttributes
LLGL::ShaderProgram, 163

R
LLGL, 33, 35, 44
r
LLGL::Color< T, 3u >, 68
LLGL::Color< T, 4u >, 71
R16
LLGL, 44
R16Float
LLGL, 44
R16Sgn
LLGL, 44
R32Float
LLGL, 44
R32SInt
LLGL, 44
R32UInt
LLGL, 44
R8
LLGL, 44
R8Sgn
LLGL, 44
RButton
LLGL, 34
RControl
LLGL, 36
RG16
LLGL, 45
RG16Float
LLGL, 45
RG16Sgn
LLGL, 45
RG32Float
LLGL, 45
RG32SInt
LLGL, 45
RG32UInt
LLGL, 45
RG8
LLGL, 44
RG8Sgn
LLGL, 45
RGB16
LLGL, 45
RGB16Float
LLGL, 45
RGB16Sgn
LLGL, 45
RGB32Float
LLGL, 45
RGB32SInt
LLGL, 45
RGB32UInt
LLGL, 45
RGB8
LLGL, 45
RGB8Sgn
LLGL, 45
RGB_DXT1
LLGL, 45
RGBA16
LLGL, 45
RGBA16Float
LLGL, 45
RGBA16Sgn
LLGL, 45
RGBA32Float
LLGL, 45
RGBA32SInt
LLGL, 45
RGBA32UInt
LLGL, 45
RGBA8
LLGL, 45
RGBA8Sgn
LLGL, 45
RGBA_DXT1
LLGL, 45
RGBA_DXT3
LLGL, 45
RGBA_DXT5
LLGL, 45
RGBA
LLGL, 33, 44
RGB
LLGL, 33, 44
RMenu
LLGL, 36
RShift
LLGL, 36
RWBuffer
LLGL, 43
RWByteAddressBuffer
LLGL, 43
RWStructuredBuffer
LLGL, 43
RWin
LLGL, 35
rasterizer
LLGL::GraphicsPipelineDescriptor, 95
readMask
LLGL::StencilFaceDescriptor, 176
ReadOnly
LLGL, 31
ReadOnlyResource
LLGL::ShaderStageFlags, 168
ReadTexture
LLGL::RenderSystem, 140
ReadWrite
LLGL, 31
RecordDrawCall

LLGL::RenderingProfiler, 127
Recreate
LLGL::Window, 214
reference
LLGL::StencilFaceDescriptor, 177
refreshRate
LLGL::VideoDisplayMode, 207
LLGL::VsyncDescriptor, 212
Release
LLGL::RenderSystem, 141, 142
RemoveEventListener
LLGL::Window, 214
renderCondition
LLGL::QueryDescriptor, 108
RenderConditionMode
LLGL, 41
RenderContext
LLGL::RenderContext, 111
RenderContext.h, 230
RenderContextDescriptor.h, 231
RenderContextFlags.h, 232
RenderSystem
LLGL::RenderSystem, 133
RenderSystem.h, 234
RenderSystemFlags.h, 234
RenderTarget.h, 235
renderedLines
LLGL::RenderingProfiler, 128
renderedPatches
LLGL::RenderingProfiler, 128
renderedPoints
LLGL::RenderingProfiler, 128
renderedTriangles
LLGL::RenderingProfiler, 128
rendererID
LLGL::RendererInfo, 117
rendererName
LLGL::RendererInfo, 117
RenderingDebugger
LLGL::RenderingDebugger, 125
LLGL::RenderingDebugger::Message, 102
RenderingDebugger.h, 232
RenderingProfiler.h, 233
Repeat
LLGL, 46
Replace
LLGL, 43
Reserved5
LLGL::RendererID, 115
Reserved6
LLGL::RendererID, 115
Reserved7
LLGL::RendererID, 115
Reserved8
LLGL::RendererID, 115
Reserved9
LLGL::RendererID, 115
ReservedA
LLGL::RendererID, 115
ReservedB
LLGL::RendererID, 116
ReservedC
LLGL::RendererID, 116
ReservedD
LLGL::RendererID, 116
ReservedE
LLGL::RendererID, 116
ReservedF
LLGL::RendererID, 116
Reset
LLGL::RenderingProfiler::Counter, 89
ResetCounters
LLGL::RenderingProfiler, 127
ResetFrameCounter
LLGL::Timer, 199
ResetResolution
LLGL::RenderTarget, 147
ResetVideoMode
LLGL::Desktop, 52
resizable
LLGL::WindowDescriptor, 216
resolution
LLGL::VideoModeDescriptor, 208
Return
LLGL, 34
RevSubtract
LLGL, 30
ReverseAND
LLGL, 38
ReverseOR
LLGL, 38
RG
LLGL, 33, 44
Right
LLGL, 34

S
LLGL, 35
SampleCount
LLGL::MultiSamplingDescriptor, 103
Sampler
LLGL::Sampler, 150
Sampler.h, 235
Sampler1D
LLGL, 47
Sampler2D
LLGL, 47
Sampler3D
LLGL, 47
SamplerArray
LLGL::SamplerArray, 150
SamplerArray.h, 236
SamplerCube
LLGL, 47
SamplerFlags.h, 236
samples
LLGL::MultiSamplingDescriptor, 103

LLGL::TextureDescriptor::Texture2DMSDescriptor, 190
SamplesPassed
LLGL, 41
Scissor
LLGL::Scissor, 154
scissorTestEnabled
LLGL::RasterizerDescriptor, 110
screen
LLGL::NativeContextHandle, 104
ScreenOrigin
LLGL, 41
screenOrigin
LLGL::RenderingCaps, 123
screenSpaceOriginLowerLeft
LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor, 174
ScrollLock
LLGL, 36
Select
LLGL, 34
semanticIndex
LLGL::StreamOutputAttribute, 181
LLGL::VertexAttribute, 202
Set
LLGL, 37
SetClearColor
LLGL::CommandBuffer, 79
SetClearDepth
LLGL::CommandBuffer, 79
SetClearStencil
LLGL::CommandBuffer, 79
SetComputePipeline
LLGL::CommandBuffer, 79
setComputePipeline
LLGL::RenderingProfiler, 128
SetConfiguration
LLGL::RenderSystem, 142
SetConstantBuffer
LLGL::CommandBuffer, 79
setConstantBuffer
LLGL::RenderingProfiler, 128
SetConstantBufferArray
LLGL::CommandBuffer, 80
SetDesc
LLGL::Window, 215
SetGraphicsAPIDependentState
LLGL::CommandBuffer, 80
SetGraphicsPipeline
LLGL::CommandBuffer, 80
setGraphicsPipeline
LLGL::RenderingProfiler, 129
SetIndexBuffer
LLGL::CommandBuffer, 80
setIndexBuffer
LLGL::RenderingProfiler, 129
SetOrCreateWindow
LLGL::RenderContext, 112
SetPosition
LLGL::Window, 215
SetRenderTarget
LLGL::CommandBuffer, 81
setRenderTarget
LLGL::RenderingProfiler, 129
SetRendererInfo
LLGL::RenderSystem, 142
SetRenderingCaps
LLGL::RenderSystem, 142
SetSampler
LLGL::CommandBuffer, 82
setSampler
LLGL::RenderingProfiler, 129
SetSamplerArray
LLGL::CommandBuffer, 82
SetScissor
LLGL::CommandBuffer, 82
SetScissorArray
LLGL::CommandBuffer, 82
SetSize
LLGL::Window, 215
SetSource
LLGL::RenderingDebugger, 125
SetStdErr
LLGL::Log, 53
SetStdOut
LLGL::Log, 53
SetStorageBuffer
LLGL::CommandBuffer, 83
setStorageBuffer
LLGL::RenderingProfiler, 129
SetStorageBufferArray
LLGL::CommandBuffer, 83
SetStreamOutputBuffer
LLGL::CommandBuffer, 83
setStreamOutputBuffer
LLGL::RenderingProfiler, 129
SetStreamOutputBufferArray
LLGL::CommandBuffer, 84
SetTexture
LLGL::CommandBuffer, 84
setTexture
LLGL::RenderingProfiler, 130
SetTextureArray
LLGL::CommandBuffer, 84
SetTitle
LLGL::Window, 215
SetUniform
LLGL::ShaderUniform, 170, 171
SetUniformArray
LLGL::ShaderUniform, 171, 172
SetVertexBuffer
LLGL::CommandBuffer, 84
setVertexBuffer
LLGL::RenderingProfiler, 130
SetVertexBufferArray
LLGL::CommandBuffer, 85

SetVideoMode
LLGL::Desktop, 52
LLGL::RenderContext, 112
SetViewport
LLGL::CommandBuffer, 85
SetViewportArray
LLGL::CommandBuffer, 85
SetVsync
LLGL::RenderContext, 112
Shader
LLGL::Shader, 155
Shader.h, 237
ShaderFlags.h, 237
ShaderProgram
LLGL::ShaderProgram, 159
shaderProgram
LLGL::ComputePipelineDescriptor, 87
LLGL::GraphicsPipelineDescriptor, 95
ShaderProgram.h, 238
ShaderSource
LLGL::ShaderSource, 165, 166
ShaderType
LLGL, 42
ShaderUniform.h, 238
ShadingLanguage
LLGL, 42
shadingLanguage
LLGL::RenderingCaps, 124
shadingLanguageName
LLGL::RendererInfo, 117
ShareWindowAndVideoMode
LLGL::RenderContext, 112
Shift
LLGL, 34
Show
LLGL::Window, 215
SingleBuffering
LLGL, 44
Size
LLGL, 30
size
LLGL::BufferDescriptor, 60
LLGL::ConstantBufferViewDescriptor, 88
LLGL::UniformDescriptor, 200
LLGL::WindowDescriptor, 216
Sleep
LLGL, 35
slopeScaledDepthBias
LLGL::RasterizerDescriptor, 110
Snapshot
LLGL, 34
sourceCode
LLGL::ShaderSource, 167
sourceHLSL
LLGL::ShaderSource, 167
Space
LLGL, 34
Src1Alpha
LLGL, 31
Src1Color
LLGL, 31
SrcAlpha
LLGL, 31
srcAlpha
LLGL::BlendTargetDescriptor, 57
SrcAlphaSaturate
LLGL, 31
SrcColor
LLGL, 31
srcColor
LLGL::BlendTargetDescriptor, 57
Start
LLGL::Timer, 199
startComponent
LLGL::StreamOutputAttribute, 182
stateOpenGL
LLGL::GraphicsAPIDependentStateDescriptor, 93
StdErr
LLGL::Log, 53
StdOut
LLGL::Log, 53
Stencil
LLGL::ClearFlags, 63
stencil
LLGL::GraphicsPipelineDescriptor, 95
stencilFailOp
LLGL::StencilFaceDescriptor, 177
StencilOp
LLGL, 42
Stop
LLGL::Timer, 199
Storage
LLGL, 32
storageBuffer
LLGL::BufferDescriptor, 61
StorageBufferDesc
Global utility functions, especially to fill descriptor structures., 16
StorageBufferType
LLGL, 43
storageType
LLGL::BufferDescriptor::StorageBufferDescriptor, 178
stream
LLGL::StreamOutputAttribute, 182
StreamOutOverflow
LLGL, 41
StreamOutPrimitivesWritten
LLGL, 41
StreamOutput
LLGL, 32
streamOutput
LLGL::ShaderSource, 167
StreamOutputAttribute
LLGL::StreamOutputAttribute, 181
StreamOutputAttribute.h, 239

StreamOutputFormat.h, 239
stride
LLGL::BufferDescriptor::StorageBufferDescriptor, 178
LLGL::VertexFormat, 205
StructuredBuffer
LLGL, 43
SubTextureDescriptor
LLGL::SubTextureDescriptor, 184
Subtract
LLGL, 30
SwapChainMode
LLGL, 43
swapChainMode
LLGL::VideoModeDescriptor, 208
SyncGPU
LLGL::CommandBuffer, 86

T
LLGL, 35
Tab
LLGL, 34
target
LLGL::ShaderSource::SourceHLSL, 173
targets
LLGL::BlendDescriptor, 56
TessControl
LLGL, 42
TessControlShaderInvocations
LLGL, 41
TessControlStage
LLGL::ShaderStageFlags, 168
TessEvaluation
LLGL, 42
TessEvaluationShaderInvocations
LLGL, 41
TessEvaluationStage
LLGL::ShaderStageFlags, 168
testEnabled
LLGL::DepthDescriptor, 90
LLGL::StencilDescriptor, 175
Texture
LLGL::Texture, 185
Texture.h, 240
Texture1DArray
LLGL, 46
Texture1DArrayDesc
Global utility functions, especially to fill descriptor structures., 16
Texture1DDesc
Global utility functions, especially to fill descriptor structures., 16
Texture1D
LLGL, 46
texture1D
LLGL::SubTextureDescriptor, 184
LLGL::TextureDescriptor, 196
Texture2DArray
LLGL, 46
Texture2DArrayDesc
Global utility functions, especially to fill descriptor structures., 16
Texture2DDesc
Global utility functions, especially to fill descriptor structures., 16
Texture2DMSArray
LLGL, 46
Texture2DMSArrayDesc
Global utility functions, especially to fill descriptor structures., 16
Texture2DMSDesc
Global utility functions, especially to fill descriptor structures., 17
Texture2DMS
LLGL, 46
texture2DMS
LLGL::TextureDescriptor, 196
Texture2D
LLGL, 46
texture2D
LLGL::SubTextureDescriptor, 184
LLGL::TextureDescriptor, 196
Texture3DDesc
Global utility functions, especially to fill descriptor structures., 17
Texture3D
LLGL, 46
texture3D
LLGL::SubTextureDescriptor, 184
LLGL::TextureDescriptor, 196
TextureArray
LLGL::TextureArray, 193
TextureArray.h, 240
TextureCube
LLGL, 46
textureCube
LLGL::SubTextureDescriptor, 185
LLGL::TextureDescriptor, 196
TextureCubeArray
LLGL, 46
TextureCubeArrayDesc
Global utility functions, especially to fill descriptor structures., 17
TextureCubeDesc
Global utility functions, especially to fill descriptor structures., 17
TextureDescriptor
LLGL::TextureDescriptor, 196
TextureFilter
LLGL, 44
TextureFlags.h, 240
TextureFormat
LLGL, 44
TextureType
LLGL, 45
TextureWrap
LLGL, 46

textureWrapU
LLGL::SamplerDescriptor, 152
textureWrapV
LLGL::SamplerDescriptor, 153
textureWrapW
LLGL::SamplerDescriptor, 153
threadCount
LLGL::RenderSystemConfiguration, 144
TimeElapsed
LLGL, 41
Timer.h, 242
title
LLGL::WindowDescriptor, 216
TriangleFan
LLGL, 39
TriangleList
LLGL, 39
TriangleListAdjacency
LLGL, 39
TriangleStrip
LLGL, 39
TriangleStripAdjacency
LLGL, 39
Triangles
LLGL, 40
TripleBuffering
LLGL, 44
type
LLGL::BufferDescriptor, 61
LLGL::QueryDescriptor, 108
LLGL::StorageBufferViewDescriptor, 179
LLGL::TextureDescriptor, 196
LLGL::UniformDescriptor, 200
Types.h, 242

U
LLGL, 35
UInt
LLGL, 47
UInt16
LLGL, 33
UInt2
LLGL, 47
UInt3
LLGL, 47
UInt32
LLGL, 33
UInt4
LLGL, 47
UInt8
LLGL, 33
UniformType
LLGL, 46
Unknown
LLGL, 44
UnlockShaderUniform
LLGL::ShaderProgram, 163
UnmapBuffer
LLGL::RenderSystem, 143
Unsupported
LLGL, 42
UnsupportedFeature
LLGL, 33
Up
LLGL, 34
UpperLeft
LLGL, 42
Utility.h, 243

V
LLGL, 35
ValueType
LLGL::RenderingProfiler::Counter, 89
VectorType
LLGL, 47
vectorType
LLGL::BufferDescriptor::StorageBufferDescriptor, 178
LLGL::VertexAttribute, 203
VectorTypeFormat
LLGL, 51
VectorTypeSize
LLGL, 52
vendor
LLGL::VideoAdapterDescriptor, 206
vendorName
LLGL::RendererInfo, 117
version
LLGL::ProfileOpenGLDescriptor, 106
Version.h, 244
Vertex
LLGL, 32, 42
VertexAttribute
LLGL::VertexAttribute, 201
VertexAttribute.h, 244
vertexBuffer
LLGL::BufferDescriptor, 61
VertexBufferDesc
Global utility functions, especially to fill descriptor structures., 17
VertexFormat.h, 245
VertexShaderInvocations
LLGL, 41
VertexStage
LLGL::ShaderStageFlags, 168
VerticesSubmitted
LLGL, 41
VideoAdapter.h, 245
videoMemory
LLGL::VideoAdapterDescriptor, 207
videoMode
LLGL::RenderContextDescriptor, 114
Viewport
LLGL::Viewport, 210
visible
LLGL::WindowDescriptor, 216
visual
LLGL::NativeContextHandle, 104

LLGL::NativeHandle, 105
VolumeDown
LLGL, 36
VolumeMute
LLGL, 36
VolumeUp
LLGL, 36
vsync
LLGL::RenderContextDescriptor, 114
Vulkan
LLGL::RendererID, 116

W
LLGL, 35
Wait
LLGL, 41
WaitInverted
LLGL, 41
WarnError
LLGL::ShaderCompileFlags, 156
WarningType
LLGL, 47
width
LLGL::Scissor, 154
LLGL::SubTextureDescriptor::Texture1DDescriptor, 186
LLGL::SubTextureDescriptor::Texture2DDescriptor, 189
LLGL::SubTextureDescriptor::Texture3DDescriptor, 192
LLGL::SubTextureDescriptor::TextureCube↩ Descriptor, 194
LLGL::TextureDescriptor::Texture1DDescriptor, 187
LLGL::TextureDescriptor::Texture2DDescriptor, 188
LLGL::TextureDescriptor::Texture2DMSDescriptor, 190
LLGL::TextureDescriptor::Texture3DDescriptor, 191
LLGL::TextureDescriptor::TextureCubeDescriptor, 193
LLGL::VideoDisplayMode, 207
LLGL::Viewport, 210
Win32NativeHandle.h, 245
Window
LLGL::Window::EventListener, 92
window
LLGL::NativeHandle, 105
Window.h, 246
windowContext
LLGL::WindowDescriptor, 216
Wireframe
LLGL, 38
WriteBuffer
LLGL::RenderSystem, 143
writeBuffer
LLGL::RenderingProfiler, 130
writeEnabled
LLGL::DepthDescriptor, 90
writeMask
LLGL::StencilFaceDescriptor, 177
WriteOnly
LLGL, 31
WriteTexture
LLGL::RenderSystem, 143

X
LLGL, 35
x
LLGL::Scissor, 154
LLGL::SubTextureDescriptor::Texture1DDescriptor, 186
LLGL::SubTextureDescriptor::Texture2DDescriptor, 189
LLGL::SubTextureDescriptor::Texture3DDescriptor, 192
LLGL::SubTextureDescriptor::TextureCube↩ Descriptor, 194
LLGL::Viewport, 211
XButton1
LLGL, 34
XButton2
LLGL, 34
XNeg
LLGL, 30
XOR
LLGL, 38
XPos
LLGL, 30

Y
LLGL, 35
y
LLGL::Scissor, 154
LLGL::SubTextureDescriptor::Texture2DDescriptor, 189
LLGL::SubTextureDescriptor::Texture3DDescriptor, 192
LLGL::SubTextureDescriptor::TextureCube↩ Descriptor, 195
LLGL::Viewport, 211
YNeg
LLGL, 30
YPos
LLGL, 30

Z
LLGL, 35
z
LLGL::SubTextureDescriptor::Texture3DDescriptor, 192
ZNeg
LLGL, 30
ZPos
LLGL, 30
Zero
LLGL, 31, 43

ZeroToOne
    LLGL, 32
Zoom
    LLGL, 37