# LLGL

0.03 Beta

Generated by Doxygen 1.8.11

# Contents


**1 LLGL 0.03 Beta Documentation** **1**

**2 Todo List** **3**

**3 Module Index** **5**

3.1 Modules . . . . . 5

**4 Namespace Index** **7**

4.1 Namespace List . . . . . 7

**5 Hierarchical Index** **9**

5.1 Class Hierarchy . . . . . 9

**6 Class Index** **13**

6.1 Class List . . . . . 13

**7 File Index** **19**

7.1 File List . . . . . 19

**8 Module Documentation** **21**

8.1 Global functions for Strict-Weak-Order (SWO) comparisons. . . . . 21

8.1.1 Detailed Description . . . . . 21

8.1.2 Function Documentation . . . . . 21

8.1.2.1 CompareSWO(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs) . . . . . 21

8.2 Global operators for basic data structures. . . . . 22

8.2.1 Detailed Description . . . . . 22

8.2.2 Function Documentation . . . . . 22

8.2.2.1 operator"!=(const Offset2D &lhs, const Offset2D &rhs) . . . 22
8.2.2.2 operator"!=(const Offset3D &lhs, const Offset3D &rhs) . . . 22
8.2.2.3 operator"!=(const Extent2D &lhs, const Extent2D &rhs) . . . 23
8.2.2.4 operator"!=(const Extent3D &lhs, const Extent3D &rhs) . . . 23
8.2.2.5 operator+(const Extent2D &lhs, const Extent2D &rhs) . . . 23
8.2.2.6 operator+(const Extent3D &lhs, const Extent3D &rhs) . . . 23
8.2.2.7 operator+(const Offset2D &lhs, const Offset2D &rhs) . . . 23
8.2.2.8 operator+(const Offset3D &lhs, const Offset3D &rhs) . . . 23
8.2.2.9 operator-(const Extent2D &lhs, const Extent2D &rhs) . . . 24
8.2.2.10 operator-(const Extent3D &lhs, const Extent3D &rhs) . . . 24
8.2.2.11 operator-(const Offset2D &lhs, const Offset2D &rhs) . . . 24
8.2.2.12 operator-(const Offset3D &lhs, const Offset3D &rhs) . . . 24
8.2.2.13 operator==(const Offset2D &lhs, const Offset2D &rhs) . . . 24
8.2.2.14 operator==(const Offset3D &lhs, const Offset3D &rhs) . . . 24
8.2.2.15 operator==(const Extent2D &lhs, const Extent2D &rhs) . . . 24
8.2.2.16 operator==(const Extent3D &lhs, const Extent3D &rhs) . . . 24
8.3 Global type aliases to callback interfaces. . . . 25
8.3.1 Detailed Description . . . 25
8.3.2 Typedef Documentation . . . 25
8.3.2.1 DebugCallback . . . 25
8.3.2.2 ValidateRenderingCapsFunc . . . 25
8.4 Buffer utility functions to determine buffer types. . . . 27
8.4.1 Detailed Description . . . 27
8.4.2 Function Documentation . . . 27
8.4.2.1 IsByteAddressBuffer(const StorageBufferType type) . . . 27
8.4.2.2 IsRWBuffer(const StorageBufferType type) . . . 27
8.4.2.3 IsStructuredBuffer(const StorageBufferType type) . . . 27
8.4.2.4 IsTypedBuffer(const StorageBufferType type) . . . 27
8.5 Hardware format utility functions. . . . 28
8.5.1 Detailed Description . . . 28

8.5.2 Function Documentation . . . 28

8.5.2.1 GetFormatDesc(const Format format) . . . 28

8.5.2.2 IsCompressedFormat(const Format format) . . . 28

8.5.2.3 IsDepthFormat(const Format format) . . . 29

8.5.2.4 IsDepthStencilFormat(const Format format) . . . 29

8.5.2.5 IsFloatFormat(const Format format) . . . 29

8.5.2.6 IsIntegralFormat(const Format format) . . . 29

8.5.2.7 IsNormalizedFormat(const Format format) . . . 30

8.5.2.8 IsStencilFormat(const Format format) . . . 30

8.6 Data type utility functions. . . . 31

8.6.1 Detailed Description . . . 31

8.6.2 Function Documentation . . . 31

8.6.2.1 DataTypeSize(const DataType dataType) . . . 31

8.6.2.2 IsFloatDataType(const DataType dataType) . . . 31

8.6.2.3 IsIntDataType(const DataType dataType) . . . 31

8.6.2.4 IsUIntDataType(const DataType dataType) . . . 31

8.7 Image utility functions to classify and convert image data. . . . 32

8.7.1 Detailed Description . . . 32

8.7.2 Function Documentation . . . 32

8.7.2.1 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, const DstImageDescriptor &dstImageDesc, std::size_t threadCount=0) . . . 32

8.7.2.2 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, ImageFormat dstFormat, DataType dstDataType, std::size_t threadCount=0) . . . 33

8.7.2.3 GenerateEmptyByteBuffer(std::size_t bufferSize, bool initialize=true) . . . 34

8.7.2.4 GenerateImageBuffer(ImageFormat format, DataType dataType, std::size_t imageSize, const ColorRGBAd &fillColor) . . . 34

8.7.2.5 ImageDataSize(const ImageFormat imageFormat, const DataType dataType, std::uint32_t numPixels) . . . 35

8.7.2.6 ImageFormatSize(const ImageFormat imageFormat) . . . 35

8.7.2.7 IsCompressedFormat(const ImageFormat imageFormat) . . . 36

8.7.2.8 IsDepthStencilFormat(const ImageFormat imageFormat) . . . 36

8.8 Buffer argument structures for indirect draw and compute commands. . . . 37

8.8.1 Detailed Description . . . 37

8.9 Global type-to-string conversion functions. . . . 38

8.9.1 Detailed Description . . . 38

8.9.2 Function Documentation . . . 38

8.9.2.1 ToString(const ShaderType t) . . . 38

8.9.2.2 ToString(const ErrorType t) . . . 38

8.9.2.3 ToString(const WarningType t) . . . 38

8.9.2.4 ToString(const ShadingLanguage t) . . . 39

8.9.2.5 ToString(const Format t) . . . 39

8.9.2.6 ToString(const TextureType t) . . . 39

8.10 Texture utility functions to determine texture dimension and buffer sizes. . . . 40

8.10.1 Detailed Description . . . 40

8.10.2 Function Documentation . . . 40

8.10.2.1 IsArrayTexture(const TextureType type) . . . 40

8.10.2.2 IsCubeTexture(const TextureType type) . . . 40

8.10.2.3 IsMipMappedTexture(const TextureDescriptor &textureDesc) . . . 41

8.10.2.4 IsMultiSampleTexture(const TextureType type) . . . 41

8.10.2.5 IsTextureSwizzleIdentity(const TextureSwizzleRGBA &swizzle) . . . 41

8.10.2.6 NumMipLevels(std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1) 41

8.10.2.7 NumMipLevels(const TextureDescriptor &textureDesc) . . . 42

8.10.2.8 TextureBufferSize(const Format format, std::uint32_t numTexels) . . . 42

8.10.2.9 TextureSize(const TextureDescriptor &textureDesc) . . . 43

8.11 Global utility functions, especially to fill descriptor structures. . . . 44

8.11.1 Detailed Description . . . 45

8.11.2 Function Documentation . . . 45

8.11.2.1 ConstantBufferDesc(uint64_t size, long cpuAccessFlags=0) . . . 45

8.11.2.2 IndexBufferDesc(uint64_t size, const Format format, long cpuAccessFlags=0) . . . 45

8.11.2.3 PipelineLayoutDesc(const ShaderReflection &reflection) . . . 45

8.11.2.4 PipelineLayoutDesc(const char *layoutSignature) . . . 45

8.11.2.5 RenderPassDesc(const RenderTargetDescriptor &renderTargetDesc) . . . 47

8.11.2.6 ShaderDescFromFile(const ShaderType type, const char *filename, const char *entryPoint=nullptr, const char *profile=nullptr, long flags=0) . . . . . 47

8.11.2.7 ShaderProgramDesc(const std::initializer_list< Shader * > &shaders, const std::initializer_list< VertexFormat > &vertexFormats={}) . . . . . 48

8.11.2.8 ShaderProgramDesc(const std::vector< Shader * > &shaders, const std↩::vector< VertexFormat > &vertexFormats={}) . . . . . 48

8.11.2.9 StorageBufferDesc(uint64_t size, const StorageBufferType storageType, std↩::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite) . . . . . 48

8.11.2.10 Texture1DArrayDesc(Format format, std::uint32_t width, std::uint32_t array↩Layers, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . 48

8.11.2.11 Texture1DDesc(Format format, std::uint32_t width, long bindFlags=(BindFlags↩::ColorAttachment" | BindFlags::Sampled)) . . . . . 49

8.11.2.12 Texture2DArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment" | Bind↩Flags::Sampled)) . . . . . 49

8.11.2.13 Texture2DDesc(Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . . . 49

8.11.2.14 Texture2DMSArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::↩ColorAttachment" | BindFlags::Sampled)) . . . . . 49

8.11.2.15 Texture2DMSDesc(Format format, std::uint32_t width, std::uint32_t height, std↩::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment" | BindFlags↩::Sampled)) . . . . . 49

8.11.2.16 Texture3DDesc(Format format, std::uint32_t width, std::uint32_t height, std↩::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::↩Sampled)) . . . . . 50

8.11.2.17 TextureCubeArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment" | Bind↩Flags::Sampled)) . . . . . 50

8.11.2.18 TextureCubeDesc(Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . . . 50

8.11.2.19 VertexBufferDesc(uint64_t size, const VertexFormat &vertexFormat, long cpu↩AccessFlags=0) . . . . . 50

**9 Namespace Documentation** **51**

9.1 LLGL Namespace Reference . . . . . 51

9.1.1 Typedef Documentation . . . . . 66

9.1.1.1 ByteBuffer . . . . . 66

9.1.1.2 ColorRGBAb . . . . . 67

9.1.1.3 ColorRGBAd . . . . . 67

9.1.1.4 ColorRGBAf . . . . . 67

9.1.1.5 ColorRGBAT . . . . . 67

9.1.1.6 ColorRGBAub . . . . . 67

9.1.1.7 ColorRGBb . . . . . 67

9.1.1.8 ColorRGBd . . . . . 67

9.1.1.9 ColorRGBf . . . . . 67

9.1.1.10 ColorRGBT . . . . . 67

9.1.1.11 ColorRGBub . . . . . 67

9.1.1.12 UniformLocation . . . . . 67

9.1.2 Enumeration Type Documentation . . . . . 67

9.1.2.1 AttachmentLoadOp . . . . . 67

9.1.2.2 AttachmentStoreOp . . . . . 68

9.1.2.3 AttachmentType . . . . . 68

9.1.2.4 BlendArithmetic . . . . . 69

9.1.2.5 BlendOp . . . . . 69

9.1.2.6 ClippingRange . . . . . 70

9.1.2.7 CompareOp . . . . . 70

9.1.2.8 CPUAccess . . . . . 71

9.1.2.9 CullMode . . . . . 71

9.1.2.10 DataType . . . . . 72

9.1.2.11 ErrorType . . . . . 72

9.1.2.12 Format . . . . . 72

9.1.2.13 ImageFormat . . . . . 77

9.1.2.14 Key . . . . . 78

9.1.2.15 LogicOp . . . 81
9.1.2.16 OpenGLContextProfile . . . 82
9.1.2.17 PolygonMode . . . 82
9.1.2.18 PrimitiveTopology . . . 83
9.1.2.19 PrimitiveType . . . 84
9.1.2.20 QueryType . . . 85
9.1.2.21 RenderConditionMode . . . 85
9.1.2.22 ResourceType . . . 86
9.1.2.23 SamplerAddressMode . . . 87
9.1.2.24 SamplerFilter . . . 88
9.1.2.25 ScreenOrigin . . . 88
9.1.2.26 ShaderSourceType . . . 89
9.1.2.27 ShaderType . . . 89
9.1.2.28 ShadingLanguage . . . 90
9.1.2.29 StencilOp . . . 91
9.1.2.30 StorageBufferType . . . 91
9.1.2.31 SystemValue . . . 92
9.1.2.32 TextureSwizzle . . . 93
9.1.2.33 TextureType . . . 94
9.1.2.34 UniformType . . . 94
9.1.2.35 WarningType . . . 95
9.1.3 Function Documentation . . . 95
9.1.3.1 CastColorValue(const Src &value) . . . 95
9.1.3.2 CastColorValue< bool, bool >(const bool &value) . . . 96
9.1.3.3 CastColorValue< double, double >(const double &value) . . . 96
9.1.3.4 CastColorValue< float, float >(const float &value) . . . 96
9.1.3.5 CastColorValue< std::uint8_t, std::uint8_t >(const std::uint8_t &value) . . . 96
9.1.3.6 CastTo(const Interface *obj) . . . 96
9.1.3.7 CastTo(Interface *obj) . . . 96
9.1.3.8 CastTo(const Interface &obj) . . . 97

9.1.3.9 CastTo(Interface &obj) . . . . . 97
9.1.3.10 GetExtentRatio(const Extent2D &extent) . . . . . 97
9.1.3.11 GetPrimitiveTopologyPatchSize(const PrimitiveTopology primitiveTopology) . . . 97
9.1.3.12 IsPrimitiveTopologyPatches(const PrimitiveTopology primitiveTopology) . . . . . 97
9.1.3.13 IsShaderSourceBinary(const ShaderSourceType type) . . . . . 98
9.1.3.14 IsShaderSourceCode(const ShaderSourceType type) . . . . . 98
9.1.3.15 MaxColorValue() . . . . . 98
9.1.3.16 MaxColorValue< bool >() . . . . . 98
9.1.3.17 MaxColorValue< std::uint8_t >() . . . . . 98
9.1.3.18 operator"!=(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs) 98
9.1.3.19 operator"!=(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 98
9.1.3.20 operator"!=(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) . . . . . 98
9.1.3.21 operator"!=(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 99
9.1.3.22 operator"!=(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . . . 99
9.1.3.23 operator"!=(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.24 operator*(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.25 operator*(const Color< T, N > &lhs, const T &rhs) . . . . . 99
9.1.3.26 operator*(const T &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.27 operator+(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.28 operator-(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.29 operator/(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.30 operator/(const Color< T, N > &lhs, const T &rhs) . . . . . 99
9.1.3.31 operator/(const T &lhs, const Color< T, N > &rhs) . . . . . 99
9.1.3.32 operator==(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs) 99
9.1.3.33 operator==(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 100
9.1.3.34 operator==(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) . . . . . 100
9.1.3.35 operator==(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 100
9.1.3.36 operator==(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . . . 100
9.1.3.37 operator==(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . . . 100

9.1.3.38 ValidateRenderingCaps(const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={}) . . . 100

9.2 LLGL::Constants Namespace Reference . . . 101

9.2.1 Detailed Description . . . 101

9.3 LLGL::Log Namespace Reference . . . 101

9.3.1 Typedef Documentation . . . 102

9.3.1.1 ReportCallback . . . 102

9.3.2 Enumeration Type Documentation . . . 102

9.3.2.1 ReportType . . . 102

9.3.3 Function Documentation . . . 103

9.3.3.1 PostReport(ReportType type, const std::string &message, const std::string &contextInfo="""") . . . 103

9.3.3.2 SetReportCallback(const ReportCallback &callback, void *userData=nullptr) . . . 103

9.3.3.3 SetReportCallbackStd(std::ostream &stream=std::cerr) . . . 103

9.3.3.4 SetReportLimit(std::size_t maxCount) . . . 104

9.4 LLGL::Version Namespace Reference . . . 104

9.4.1 Detailed Description . . . 104

9.4.2 Function Documentation . . . 104

9.4.2.1 GetID() . . . 104

9.4.2.2 GetMajor() . . . 105

9.4.2.3 GetMinor() . . . 105

9.4.2.4 GetRevision() . . . 105

9.4.2.5 GetStatus() . . . 105

9.4.2.6 GetString() . . . 105

**10 Class Documentation** 107

10.1 LLGL::ApplicationDescriptor Struct Reference . . . 107

10.1.1 Detailed Description . . . 107

10.1.2 Member Data Documentation . . . 107

10.1.2.1 applicationName . . . 107

10.1.2.2 applicationVersion . . . 108

10.1.2.3 engineName . . . 108

10.1.2.4 engineVersion . . . 108

10.2 LLGL::AttachmentClear Struct Reference . . . 108

10.2.1 Detailed Description . . . 109

10.2.2 Constructor & Destructor Documentation . . . 109

10.2.2.1 AttachmentClear()=default . . . 109

10.2.2.2 AttachmentClear(const AttachmentClear &)=default . . . 109

10.2.2.3 AttachmentClear(const ColorRGBAf &color, std::uint32_t colorAttachment) . . . 109

10.2.2.4 AttachmentClear(float depth) . . . 109

10.2.2.5 AttachmentClear(std::uint32_t stencil) . . . 109

10.2.2.6 AttachmentClear(float depth, std::uint32_t stencil) . . . 109

10.2.3 Member Function Documentation . . . 109

10.2.3.1 operator=(const AttachmentClear &)=default . . . 109

10.2.4 Member Data Documentation . . . 109

10.2.4.1 clearValue . . . 109

10.2.4.2 colorAttachment . . . 110

10.2.4.3 flags . . . 110

10.3 LLGL::AttachmentDescriptor Struct Reference . . . 110

10.3.1 Detailed Description . . . 111

10.3.2 Constructor & Destructor Documentation . . . 111

10.3.2.1 AttachmentDescriptor()=default . . . 111

10.3.2.2 AttachmentDescriptor(const AttachmentDescriptor &)=default . . . 111

10.3.2.3 AttachmentDescriptor(AttachmentType type) . . . 111

10.3.2.4 AttachmentDescriptor(AttachmentType type, Texture *texture, std::uint32_t mip↩ Level=0, std::uint32_t arrayLayer=0) . . . 111

10.3.3 Member Data Documentation . . . 111
10.3.3.1 arrayLayer . . . 111
10.3.3.2 mipLevel . . . 112
10.3.3.3 texture . . . 112
10.3.3.4 type . . . 112
10.4 LLGL::AttachmentFormatDescriptor Struct Reference . . . 112
10.4.1 Detailed Description . . . 113
10.4.2 Constructor & Destructor Documentation . . . 113
10.4.2.1 AttachmentFormatDescriptor()=default . . . 113
10.4.2.2 AttachmentFormatDescriptor(const AttachmentFormatDescriptor &)=default . . . 113
10.4.2.3 AttachmentFormatDescriptor(const Format format, const AttachmentLoadOp loadOp=AttachmentLoadOp::Load, const AttachmentStoreOp storeOp=AttachmentStoreOp::Store) . . . 113
10.4.3 Member Data Documentation . . . 113
10.4.3.1 format . . . 113
10.4.3.2 loadOp . . . 114
10.4.3.3 storeOp . . . 114
10.5 LLGL::BindFlags Struct Reference . . . 114
10.5.1 Detailed Description . . . 114
10.5.2 Member Enumeration Documentation . . . 115
10.5.2.1 anonymous enum . . . 115
10.6 LLGL::BindingDescriptor Struct Reference . . . 116
10.6.1 Detailed Description . . . 117
10.6.2 Constructor & Destructor Documentation . . . 117
10.6.2.1 BindingDescriptor()=default . . . 117
10.6.2.2 BindingDescriptor(const BindingDescriptor &)=default . . . 117
10.6.2.3 BindingDescriptor(ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1) . . . 117
10.6.2.4 BindingDescriptor(const std::string &name, ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1) . . . 117
10.6.3 Member Data Documentation . . . 117
10.6.3.1 arraySize . . . 117

10.6.3.2 bindFlags . . . 118
10.6.3.3 name . . . 118
10.6.3.4 slot . . . 118
10.6.3.5 stageFlags . . . 118
10.6.3.6 type . . . 119
10.7 LLGL::BlendDescriptor Struct Reference . . . 119
10.7.1 Detailed Description . . . 119
10.7.2 Member Data Documentation . . . 119
10.7.2.1 alphaToCoverageEnabled . . . 119
10.7.2.2 blendFactor . . . 120
10.7.2.3 independentBlendEnabled . . . 120
10.7.2.4 logicOp . . . 120
10.7.2.5 targets . . . 121
10.8 LLGL::BlendTargetDescriptor Struct Reference . . . 121
10.8.1 Detailed Description . . . 121
10.8.2 Member Data Documentation . . . 122
10.8.2.1 alphaArithmetic . . . 122
10.8.2.2 blendEnabled . . . 122
10.8.2.3 colorArithmetic . . . 122
10.8.2.4 colorMask . . . 122
10.8.2.5 dstAlpha . . . 122
10.8.2.6 dstColor . . . 122
10.8.2.7 srcAlpha . . . 122
10.8.2.8 srcColor . . . 122
10.9 LLGL::Buffer Class Reference . . . 123
10.9.1 Detailed Description . . . 123
10.9.2 Constructor & Destructor Documentation . . . 123
10.9.2.1 Buffer(long bindFlags) . . . 123
10.9.3 Member Function Documentation . . . 123
10.9.3.1 GetBindFlags() const . . . 123

10.9.3.2 GetDesc() const =0 . . . 124
10.9.3.3 GetResourceType() const override final . . . 124
10.10 LLGL::BufferArray Class Reference . . . 124
10.10.1 Detailed Description . . . 125
10.10.2 Constructor & Destructor Documentation . . . 125
10.10.2.1 BufferArray(long bindFlags) . . . 125
10.10.3 Member Function Documentation . . . 125
10.10.3.1 GetBindFlags() const . . . 125
10.11 LLGL::BufferDescriptor Struct Reference . . . 125
10.11.1 Detailed Description . . . 126
10.11.2 Member Data Documentation . . . 126
10.11.2.1 bindFlags . . . 126
10.11.2.2 cpuAccessFlags . . . 127
10.11.2.3 indexBuffer . . . 127
10.11.2.4 miscFlags . . . 127
10.11.2.5 size . . . 127
10.11.2.6 storageBuffer . . . 127
10.11.2.7 vertexBuffer . . . 128
10.12 LLGL::Canvas Class Reference . . . 128
10.12.1 Detailed Description . . . 129
10.12.2 Member Function Documentation . . . 129
10.12.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc) override . . . 129
10.12.2.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 129
10.12.2.3 Create(const CanvasDescriptor &desc) . . . 129
10.12.2.4 GetTitle() const =0 . . . 130
10.12.2.5 HasQuit() const . . . 130
10.12.2.6 OnProcessEvents()=0 . . . 130
10.12.2.7 PostQuit() . . . 130
10.12.2.8 ProcessEvents() override final . . . 130
10.12.2.9 RemoveEventListener(const EventListener *eventListener) . . . 131

10.12.2.10 SetTitle(const std::wstring &title)=0 . . . 131
10.13 LLGL::CanvasDescriptor Struct Reference . . . 131
10.13.1 Detailed Description . . . 131
10.13.2 Member Data Documentation . . . 131
10.13.2.1 borderless . . . 131
10.13.2.2 title . . . 131
10.14 LLGL::ClearFlags Struct Reference . . . 132
10.14.1 Detailed Description . . . 132
10.14.2 Member Enumeration Documentation . . . 132
10.14.2.1 anonymous enum . . . 132
10.15 LLGL::ClearValue Struct Reference . . . 132
10.15.1 Detailed Description . . . 133
10.15.2 Member Data Documentation . . . 133
10.15.2.1 color . . . 133
10.15.2.2 depth . . . 133
10.15.2.3 stencil . . . 133
10.16 LLGL::Color< T, N > Class Template Reference . . . 133
10.16.1 Detailed Description . . . 134
10.16.2 Constructor & Destructor Documentation . . . 135
10.16.2.1 Color() . . . 135
10.16.2.2 Color(const Color< T, N > &rhs) . . . 135
10.16.2.3 Color(const T &scalar) . . . 135
10.16.2.4 Color(UninitializeTag) . . . 135
10.16.3 Member Function Documentation . . . 135
10.16.3.1 Cast() const . . . 135
10.16.3.2 operator*=(const Color< T, N > &rhs) . . . 136
10.16.3.3 operator*=(const T rhs) . . . 136
10.16.3.4 operator+=(const Color< T, N > &rhs) . . . 136
10.16.3.5 operator-() const . . . 136
10.16.3.6 operator-=(const Color< T, N > &rhs) . . . 136

10.16.3.7 operator/=(const Color< T, N > &rhs) . . . 136
10.16.3.8 operator/=(const T rhs) . . . 136
10.16.3.9 operator[](std::size_t component) . . . 136
10.16.3.10 operator[](std::size_t component) const . . . 137
10.16.3.11 Ptr() . . . 137
10.16.3.12 Ptr() const . . . 137
10.16.4 Member Data Documentation . . . 137
10.16.4.1 components . . . 137
10.17 LLGL::Color< T, 3u > Class Template Reference . . . 137
10.17.1 Detailed Description . . . 139
10.17.2 Constructor & Destructor Documentation . . . 139
10.17.2.1 Color() . . . 139
10.17.2.2 Color(const Color< T, 3 > &rhs) . . . 139
10.17.2.3 Color(const T &scalar) . . . 139
10.17.2.4 Color(const T &r, const T &g, const T &b) . . . 139
10.17.2.5 Color(UninitializeTag) . . . 139
10.17.3 Member Function Documentation . . . 139
10.17.3.1 Cast() const . . . 139
10.17.3.2 operator*=(const Color< T, 3 > &rhs) . . . 140
10.17.3.3 operator*=(const T rhs) . . . 140
10.17.3.4 operator+=(const Color< T, 3 > &rhs) . . . 140
10.17.3.5 operator-() const . . . 140
10.17.3.6 operator-=(const Color< T, 3 > &rhs) . . . 140
10.17.3.7 operator/=(const Color< T, 3 > &rhs) . . . 140
10.17.3.8 operator/=(const T rhs) . . . 140
10.17.3.9 operator[](std::size_t component) . . . 140
10.17.3.10 operator[](std::size_t component) const . . . 141
10.17.3.11 Ptr() . . . 141
10.17.3.12 Ptr() const . . . 141
10.17.3.13 ToRGBA() const . . . 141

10.17.4 Member Data Documentation . . . 141
10.17.4.1 b . . . 141
10.17.4.2 components . . . 141
10.17.4.3 g . . . 142
10.17.4.4 r . . . 142
10.18 LLGL::Color< T, 4u > Class Template Reference . . . 142
10.18.1 Detailed Description . . . 143
10.18.2 Constructor & Destructor Documentation . . . 143
10.18.2.1 Color() . . . 143
10.18.2.2 Color(const Color< T, 4 > &rhs) . . . 144
10.18.2.3 Color(const T &scalar) . . . 144
10.18.2.4 Color(const Color< T, 3u > &rhs) . . . 144
10.18.2.5 Color(const T &r, const T &g, const T &b) . . . 144
10.18.2.6 Color(const T &r, const T &g, const T &b, const T &a) . . . 144
10.18.2.7 Color(UninitializeTag) . . . 144
10.18.3 Member Function Documentation . . . 144
10.18.3.1 Cast() const . . . 144
10.18.3.2 operator*=(const Color< T, 4 > &rhs) . . . 145
10.18.3.3 operator*=(const T rhs) . . . 145
10.18.3.4 operator+=(const Color< T, 4 > &rhs) . . . 145
10.18.3.5 operator-() const . . . 145
10.18.3.6 operator-=(const Color< T, 4 > &rhs) . . . 145
10.18.3.7 operator/=(const Color< T, 4 > &rhs) . . . 145
10.18.3.8 operator/=(const T rhs) . . . 145
10.18.3.9 operator[](std::size_t component) . . . 145
10.18.3.10 operator[](std::size_t component) const . . . 146
10.18.3.11 Ptr() . . . 146
10.18.3.12 Ptr() const . . . 146
10.18.3.13 ToRGB() const . . . 146
10.18.4 Member Data Documentation . . . 146

10.18.4.1 a . . . 146

10.18.4.2 b . . . 146

10.18.4.3 components . . . 146

10.18.4.4 g . . . 147

10.18.4.5 r . . . 147

10.19 LLGL::CommandBuffer Class Reference . . . 147

10.19.1 Detailed Description . . . 150

10.19.2 Constructor & Destructor Documentation . . . 150

10.19.2.1 CommandBuffer()=default . . . 150

10.19.3 Member Function Documentation . . . 150

10.19.3.1 Begin()=0 . . . 150

10.19.3.2 BeginQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 . . . 150

10.19.3.3 BeginRenderCondition(QueryHeap &queryHeap, std::uint32_t query=0, const RenderConditionMode mode=RenderConditionMode::Wait)=0 . . . 151

10.19.3.4 BeginRenderPass(RenderTarget &renderTarget, const RenderPass *renderPass=nullptr, std::uint32_t numClearValues=0, const ClearValue *clearValues=nullptr)=0 . . . 151

10.19.3.5 BeginStreamOutput(const PrimitiveType primitiveType)=0 . . . 152

10.19.3.6 Clear(long flags)=0 . . . 153

10.19.3.7 ClearAttachments(std::uint32_t numAttachments, const AttachmentClear *attachments)=0 . . . 153

10.19.3.8 CopyBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, Buffer &srcBuffer, std::uint64_t srcOffset, std::uint64_t size)=0 . . . 154

10.19.3.9 CopyTexture(Texture &dstTexture, const TextureLocation &dstLocation, Texture &srcTexture, const TextureLocation &srcLocation, const Extent3D &extent)=0 . . . 154

10.19.3.10 Dispatch(std::uint32_t numWorkGroupsX, std::uint32_t numWorkGroupsY, std::uint32_t numWorkGroupsZ)=0 . . . 155

10.19.3.11 DispatchIndirect(Buffer &buffer, std::uint64_t offset)=0 . . . 155

10.19.3.12 Draw(std::uint32_t numVertices, std::uint32_t firstVertex)=0 . . . 155

10.19.3.13 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex)=0 . . . 156

10.19.3.14 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex, std::int32_t vertexOffset)=0 . . . 156

10.19.3.15 DrawIndexedIndirect(Buffer &buffer, std::uint64_t offset)=0 . . . 156

10.19.3.16 DrawIndexedIndirect(Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0 . . . 157

10.19.3.17 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex)=0 . . . 157

10.19.3.18 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset)=0 . . . 157

10.19.3.19 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset, std::uint32_t firstInstance)=0 . . . 157

10.19.3.20 DrawIndirect(Buffer &buffer, std::uint64_t offset)=0 . . . 158

10.19.3.21 DrawIndirect(Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0 . . . 158

10.19.3.22 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances)=0 . . . 159

10.19.3.23 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances, std::uint32_t firstInstance)=0 . . . 159

10.19.3.24 End()=0 . . . 160

10.19.3.25 EndQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 . . . 160

10.19.3.26 EndRenderCondition()=0 . . . 160

10.19.3.27 EndRenderPass()=0 . . . 160

10.19.3.28 EndStreamOutput()=0 . . . 161

10.19.3.29 Execute(CommandBuffer &deferredCommandBuffer)=0 . . . 161

10.19.3.30 GenerateMips(Texture &texture)=0 . . . 161

10.19.3.31 GenerateMips(Texture &texture, const TextureSubresource &subresource)=0 . . . 162

10.19.3.32 PopDebugGroup()=0 . . . 162

10.19.3.33 PushDebugGroup(const char *name)=0 . . . 162

10.19.3.34 ResetResourceSlots(const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_t numSlots, long bindFlags, long stageFlags=StageFlags::AllStages)=0 163

10.19.3.35 SetClearColor(const ColorRGBAf &color)=0 . . . 164

10.19.3.36 SetClearDepth(float depth)=0 . . . 164

10.19.3.37 SetClearStencil(std::uint32_t stencil)=0 . . . 164

10.19.3.38 SetComputePipeline(ComputePipeline &computePipeline)=0 . . . 164

10.19.3.39 SetComputeResourceHeap(ResourceHeap &resourceHeap, std::uint32_t firstSet=0)=0 . . . 165

10.19.3.40 SetGraphicsAPIDependentState(const void *stateDesc, std::size_t stateDescSize)=0 . . . 165

10.19.3.41 SetGraphicsPipeline(GraphicsPipeline &graphicsPipeline)=0 . . . 166

10.19.3.42 SetGraphicsResourceHeap(ResourceHeap &resourceHeap, std::uint32_t firstSet=0)=0 . . . 166

10.19.3.43 SetIndexBuffer(Buffer &buffer)=0 . . . 167

10.19.3.44 SetIndexBuffer(Buffer &buffer, const Format format, std::uint64_t offset=0)=0 . . . 167

10.19.3.45 SetResource(Resource &resource, std::uint32_t slot, long bindFlags, long stageFlags=StageFlags::AllStages)=0 . . . 168

10.19.3.46 SetScissor(const Scissor &scissor)=0 . . . 168

10.19.3.47 SetScissors(std::uint32_t numScissors, const Scissor *scissors)=0 . . . 169

10.19.3.48 SetStreamOutputBuffer(Buffer &buffer)=0 . . . 169

10.19.3.49 SetStreamOutputBufferArray(BufferArray &bufferArray)=0 . . . 169

10.19.3.50 SetUniform(UniformLocation location, const void *data, std::uint32_t dataSize)=0 170

10.19.3.51 SetUniforms(UniformLocation location, std::uint32_t count, const void *data, std::uint32_t dataSize)=0 . . . 170

10.19.3.52 SetVertexBuffer(Buffer &buffer)=0 . . . 171

10.19.3.53 SetVertexBufferArray(BufferArray &bufferArray)=0 . . . 171

10.19.3.54 SetViewport(const Viewport &viewport)=0 . . . 171

10.19.3.55 SetViewports(std::uint32_t numViewports, const Viewport *viewports)=0 . . . 172

10.19.3.56 UpdateBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint16_t dataSize)=0 . . . 172

10.20 LLGL::CommandBufferDescriptor Struct Reference . . . 173

10.20.1 Detailed Description . . . 173

10.20.2 Member Data Documentation . . . 173

10.20.2.1 flags . . . 173

10.20.2.2 numNativeBuffers . . . 173

10.21 LLGL::CommandBufferFlags Struct Reference . . . 174

10.21.1 Detailed Description . . . 174

10.21.2 Member Enumeration Documentation . . . 174

10.21.2.1 anonymous enum . . . 174

10.22 LLGL::CommandQueue Class Reference . . . 175

10.22.1 Detailed Description . . . 175

10.22.2 Constructor & Destructor Documentation . . . 176

10.22.2.1 CommandQueue()=default . . . 176

10.22.3 Member Function Documentation . . . 176

10.22.3.1 QueryResult(QueryHeap &queryHeap, std::uint32_t firstQuery, std::uint32_t numQueries, void *data, std::size_t dataSize)=0 . . . 176

10.22.3.2 Submit(CommandBuffer &commandBuffer)=0 . . . 176

10.22.3.3 Submit(Fence &fence)=0 . . . 177

10.22.3.4 WaitFence(Fence &fence, std::uint64_t timeout)=0 . . . 177

10.22.3.5 WaitIdle()=0 . . . 177

10.23 LLGL::ComputePipeline Class Reference . . . 178

10.23.1 Detailed Description . . . 178

10.24 LLGL::ComputePipelineDescriptor Struct Reference . . . 178

10.24.1 Detailed Description . . . 179

10.24.2 Member Data Documentation . . . 179

10.24.2.1 pipelineLayout . . . 179

10.24.2.2 shaderProgram . . . 179

10.25 LLGL::CPUAccessFlags Struct Reference . . . 179

10.25.1 Detailed Description . . . 180

10.25.2 Member Enumeration Documentation . . . 180

10.25.2.1 anonymous enum . . . 180

10.26 LLGL::DepthBiasDescriptor Struct Reference . . . 180

10.26.1 Detailed Description . . . 181

10.26.2 Member Data Documentation . . . 181

10.26.2.1 clamp . . . 181

10.26.2.2 constantFactor . . . 181

10.26.2.3 slopeFactor . . . 181

10.27 LLGL::DepthDescriptor Struct Reference . . . 181

10.27.1 Detailed Description . . . 182

10.27.2 Member Data Documentation . . . 182

10.27.2.1 compareOp . . . 182

10.27.2.2 testEnabled . . . 182

10.27.2.3 writeEnabled . . . 182

10.28 LLGL::DispatchIndirectArguments Struct Reference . . . 182
10.28.1 Detailed Description . . . 183
10.28.2 Member Data Documentation . . . 183
10.28.2.1 numThreadGroups . . . 183
10.29 LLGL::Display Class Reference . . . 183
10.29.1 Detailed Description . . . 184
10.29.2 Member Function Documentation . . . 185
10.29.2.1 FinalizeDisplayModes(std::vector< DisplayModeDescriptor > &displayModeDescs) 185
10.29.2.2 GetDeviceName() const =0 . . . 185
10.29.2.3 GetDisplayMode() const =0 . . . 185
10.29.2.4 GetOffset() const =0 . . . 185
10.29.2.5 GetSupportedDisplayModes() const =0 . . . 185
10.29.2.6 InstantiateList() . . . 186
10.29.2.7 InstantiatePrimary() . . . 186
10.29.2.8 IsCursorShown() . . . 186
10.29.2.9 IsPrimary() const =0 . . . 186
10.29.2.10 ResetDisplayMode()=0 . . . 186
10.29.2.11 SetDisplayMode(const DisplayModeDescriptor &displayModeDesc)=0 . . . 186
10.29.2.12 ShowCursor(bool show) . . . 187
10.30 LLGL::DisplayModeDescriptor Struct Reference . . . 187
10.30.1 Detailed Description . . . 188
10.30.2 Member Data Documentation . . . 188
10.30.2.1 refreshRate . . . 188
10.30.2.2 resolution . . . 188
10.31 LLGL::DrawIndexedIndirectArguments Struct Reference . . . 188
10.31.1 Detailed Description . . . 189
10.31.2 Member Data Documentation . . . 189
10.31.2.1 firstIndex . . . 189
10.31.2.2 firstInstance . . . 189
10.31.2.3 numIndices . . . 189

10.31.2.4 numInstances . . . 190
10.31.2.5 vertexOffset . . . 190
10.32 LLGL::DrawIndirectArguments Struct Reference . . . 190
10.32.1 Detailed Description . . . 190
10.32.2 Member Data Documentation . . . 191
10.32.2.1 firstInstance . . . 191
10.32.2.2 firstVertex . . . 191
10.32.2.3 numInstances . . . 191
10.32.2.4 numVertices . . . 191
10.33 LLGL::DrawPatchIndirectArgument Struct Reference . . . 191
10.33.1 Detailed Description . . . 192
10.33.2 Member Data Documentation . . . 192
10.33.2.1 firstInstance . . . 192
10.33.2.2 firstPatch . . . 192
10.33.2.3 numInstances . . . 192
10.33.2.4 numPatches . . . 193
10.34 LLGL::DstImageDescriptor Struct Reference . . . 193
10.34.1 Detailed Description . . . 193
10.34.2 Constructor & Destructor Documentation . . . 194
10.34.2.1 DstImageDescriptor()=default . . . 194
10.34.2.2 DstImageDescriptor(const DstImageDescriptor &)=default . . . 194
10.34.2.3 DstImageDescriptor(ImageFormat format, DataType dataType, void *data, std↩ ::size_t dataSize) . . . 194
10.34.3 Member Data Documentation . . . 194
10.34.3.1 data . . . 194
10.34.3.2 dataSize . . . 194
10.34.3.3 dataType . . . 194
10.34.3.4 format . . . 194
10.35 LLGL::Window::EventListener Class Reference . . . 194
10.35.1 Detailed Description . . . 195
10.35.2 Member Function Documentation . . . 196

10.35.2.1 OnChar(Window &sender, wchar_t chr) . . . 196
10.35.2.2 OnDoubleClick(Window &sender, Key keyCode) . . . 196
10.35.2.3 OnGetFocus(Window &sender) . . . 196
10.35.2.4 OnGlobalMotion(Window &sender, const Offset2D &motion) . . . 196
10.35.2.5 OnKeyDown(Window &sender, Key keyCode) . . . 196
10.35.2.6 OnKeyUp(Window &sender, Key keyCode) . . . 196
10.35.2.7 OnLocalMotion(Window &sender, const Offset2D &position) . . . 196
10.35.2.8 OnLostFocus(Window &sender) . . . 197
10.35.2.9 OnProcessEvents(Window &sender) . . . 197
10.35.2.10 OnQuit(Window &sender, bool &veto) . . . 197
10.35.2.11 OnResize(Window &sender, const Extent2D &clientAreaSize) . . . 197
10.35.2.12 OnTimer(Window &sender, std::uint32_t timerID) . . . 197
10.35.2.13 OnWheelMotion(Window &sender, int motion) . . . 198
10.35.3 Friends And Related Function Documentation . . . 198
10.35.3.1 Window . . . 198
10.36 LLGL::Canvas::EventListener Class Reference . . . 198
10.36.1 Detailed Description . . . 198
10.36.2 Member Function Documentation . . . 199
10.36.2.1 OnProcessEvents(Canvas &sender) . . . 199
10.36.2.2 OnQuit(Canvas &sender, bool &veto) . . . 199
10.36.3 Friends And Related Function Documentation . . . 199
10.36.3.1 Canvas . . . 199
10.37 LLGL::Extent2D Struct Reference . . . 199
10.37.1 Detailed Description . . . 200
10.37.2 Constructor & Destructor Documentation . . . 200
10.37.2.1 Extent2D()=default . . . 200
10.37.2.2 Extent2D(const Extent2D &)=default . . . 200
10.37.2.3 Extent2D(std::uint32_t width, std::uint32_t height) . . . 200
10.37.3 Member Data Documentation . . . 200
10.37.3.1 height . . . 200

10.37.3.2 width . . . 200
10.38 LLGL::Extent3D Struct Reference . . . 200
10.38.1 Detailed Description . . . 201
10.38.2 Constructor & Destructor Documentation . . . 201
10.38.2.1 Extent3D()=default . . . 201
10.38.2.2 Extent3D(const Extent3D &)=default . . . 201
10.38.2.3 Extent3D(std::uint32_t width, std::uint32_t height, std::uint32_t depth) . . . 201
10.38.3 Member Data Documentation . . . 201
10.38.3.1 depth . . . 201
10.38.3.2 height . . . 201
10.38.3.3 width . . . 201
10.39 LLGL::Fence Class Reference . . . 202
10.39.1 Detailed Description . . . 202
10.40 LLGL::FormatDescriptor Struct Reference . . . 202
10.40.1 Detailed Description . . . 203
10.40.2 Member Data Documentation . . . 203
10.40.2.1 bitSize . . . 203
10.40.2.2 blockHeight . . . 204
10.40.2.3 blockWidth . . . 204
10.40.2.4 components . . . 204
10.40.2.5 compressed . . . 204
10.40.2.6 dataType . . . 204
10.40.2.7 depth . . . 204
10.40.2.8 format . . . 204
10.40.2.9 normalized . . . 204
10.40.2.10 packed . . . 204
10.40.2.11 sRGB . . . 205
10.40.2.12 stencil . . . 205
10.41 LLGL::FragmentAttribute Struct Reference . . . 205
10.41.1 Detailed Description . . . 205

10.41.2 Member Data Documentation . . . 205
10.41.2.1 format . . . 205
10.41.2.2 name . . . 206
10.41.2.3 semanticIndex . . . 206
10.41.2.4 systemValue . . . 206
10.42 LLGL::FrameProfile Struct Reference . . . 206
10.42.1 Detailed Description . . . 208
10.42.2 Constructor & Destructor Documentation . . . 208
10.42.2.1 FrameProfile() . . . 208
10.42.3 Member Function Documentation . . . 208
10.42.3.1 Accumulate(const FrameProfile &rhs) . . . 208
10.42.3.2 Clear() . . . 208
10.42.4 Member Data Documentation . . . 208
10.42.4.1 "@4 . . . 208
10.42.4.2 attachmentClears . . . 208
10.42.4.3 bufferCopies . . . 209
10.42.4.4 bufferMappings . . . 209
10.42.4.5 bufferReads . . . 209
10.42.4.6 bufferUpdates . . . 209
10.42.4.7 bufferWrites . . . 209
10.42.4.8 commandBufferEncodings . . . 209
10.42.4.9 commandBufferSubmittions . . . 210
10.42.4.10 computePipelineBindings . . . 210
10.42.4.11 computeResourceHeapBindings . . . 210
10.42.4.12 constantBufferBindings . . . 210
10.42.4.13 dispatchCommands . . . 210
10.42.4.14 drawCommands . . . 211
10.42.4.15 fenceSubmissions . . . 211
10.42.4.16 graphicsPipelineBindings . . . 211
10.42.4.17 graphicsResourceHeapBindings . . . 211

10.42.4.18 indexBufferBindings . . . 211
10.42.4.19 mipMapsGenerations . . . 212
10.42.4.20 querySections . . . 212
10.42.4.21 renderConditionSections . . . 212
10.42.4.22 renderPassSections . . . 212
10.42.4.23 sampledBufferBindings . . . 212
10.42.4.24 sampledTextureBindings . . . 213
10.42.4.25 samplerBindings . . . 213
10.42.4.26 storageBufferBindings . . . 213
10.42.4.27 storageTextureBindings . . . 213
10.42.4.28 streamOutputBufferBindings . . . 213
10.42.4.29 streamOutputSections . . . 214
10.42.4.30 textureCopies . . . 214
10.42.4.31 textureMappings . . . 214
10.42.4.32 textureReads . . . 214
10.42.4.33 textureWrites . . . 214
10.42.4.34 values . . . 214
10.42.4.35 vertexBufferBindings . . . 215
10.43 LLGL::GraphicsPipeline Class Reference . . . 215
10.43.1 Detailed Description . . . 215
10.44 LLGL::GraphicsPipelineDescriptor Struct Reference . . . 216
10.44.1 Detailed Description . . . 216
10.44.2 Member Data Documentation . . . 216
10.44.2.1 blend . . . 216
10.44.2.2 depth . . . 217
10.44.2.3 pipelineLayout . . . 217
10.44.2.4 primitiveTopology . . . 217
10.44.2.5 rasterizer . . . 217
10.44.2.6 renderPass . . . 217
10.44.2.7 scissors . . . 217

10.44.2.8 shaderProgram . . . 218

10.44.2.9 stencil . . . 218

10.44.2.10 viewports . . . 218

10.45 LLGL::Image Class Reference . . . 218

10.45.1 Detailed Description . . . 220

10.45.2 Constructor & Destructor Documentation . . . 221

10.45.2.1 Image()=default . . . 221

10.45.2.2 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType) . . . 221

10.45.2.3 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, const ColorRGBAd &fillColor) . . . 221

10.45.2.4 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) . . . 221

10.45.2.5 Image(const Image &rhs) . . . 221

10.45.2.6 Image(Image &&rhs) . . . 221

10.45.3 Member Function Documentation . . . 221

10.45.3.1 Blit(Offset3D dstRegionOffset, const Image &srcImage, Offset3D srcRegion↩ Offset, Extent3D srcRegionExtent) . . . 221

10.45.3.2 Convert(const ImageFormat format, const DataType dataType, std::size_↩ t threadCount=0) . . . 222

10.45.3.3 Fill(Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor) . . . 222

10.45.3.4 GetBytesPerPixel() const . . . 222

10.45.3.5 GetData() const . . . 223

10.45.3.6 GetData() . . . 223

10.45.3.7 GetDataSize() const . . . 223

10.45.3.8 GetDataType() const . . . 223

10.45.3.9 GetDepthStride() const . . . 223

10.45.3.10 GetDstDesc() . . . 223

10.45.3.11 GetExtent() const . . . 223

10.45.3.12 GetFormat() const . . . 223

10.45.3.13 GetNumPixels() const . . . 224

10.45.3.14 GetRowStride() const . . . 224

10.45.3.15 GetSrcDesc() const . . . 224

10.45.3.16 IsRegionInside(const Offset3D &offset, const Extent3D &extent) const . . . . . 224

10.45.3.17 MirrorXYPlane() . . . . . 224

10.45.3.18 MirrorXZPlane() . . . . . 224

10.45.3.19 MirrorYZPlane() . . . . . 224

10.45.3.20 operator=(const Image &rhs) . . . . . 225

10.45.3.21 operator=(Image &&rhs) . . . . . 225

10.45.3.22 ReadPixels(const Offset3D &offset, const Extent3D &extent, const DstImage↩ Descriptor &imageDesc, std::size_t threadCount=0) const . . . . . 225

10.45.3.23 Release() . . . . . 225

10.45.3.24 Reset() . . . . . 226

10.45.3.25 Reset(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) . . . . . 226

10.45.3.26 Resize(const Extent3D &extent) . . . . . 226

10.45.3.27 Resize(const Extent3D &extent, const ColorRGBAd &fillColor) . . . . . 226

10.45.3.28 Resize(const Extent3D &extent, const ColorRGBAd &fillColor, const Offset3↩ D &offset) . . . . . 226

10.45.3.29 Resize(const Extent3D &extent, const SamplerFilter filter) . . . . . 227

10.45.3.30 Swap(Image &rhs) . . . . . 227

10.45.3.31 WritePixels(const Offset3D &offset, const Extent3D &extent, const SrcImage↩ Descriptor &imageDesc, std::size_t threadCount=0) . . . . . 227

10.46 LLGL::BufferDescriptor::IndexBuffer Struct Reference . . . . . 228

10.46.1 Detailed Description . . . . . 228

10.46.2 Member Data Documentation . . . . . 228

10.46.2.1 format . . . . . 228

10.47 LLGL::Input Class Reference . . . . . 228

10.47.1 Detailed Description . . . . . 230

10.47.2 Constructor & Destructor Documentation . . . . . 230

10.47.2.1 Input() . . . . . 230

10.47.3 Member Function Documentation . . . . . 230

10.47.3.1 GetAnyKeyCount() const . . . . . 230

10.47.3.2 GetEnteredChars() const . . . . . 230

10.47.3.3 GetMouseMotion() const . . . . . 230

10.47.3.4 GetMousePosition() const . . . 230
10.47.3.5 GetWheelMotion() const . . . 230
10.47.3.6 KeyDoubleClick(Key keyCode) const . . . 231
10.47.3.7 KeyDown(Key keyCode) const . . . 231
10.47.3.8 KeyDownRepeated(Key keyCode) const . . . 231
10.47.3.9 KeyPressed(Key keyCode) const . . . 231
10.47.3.10 KeyUp(Key keyCode) const . . . 231
10.47.3.11 OnChar(Window &sender, wchar_t chr) override . . . 231
10.47.3.12 OnDoubleClick(Window &sender, Key keyCode) override . . . 231
10.47.3.13 OnGlobalMotion(Window &sender, const Offset2D &motion) override . . . 231
10.47.3.14 OnKeyDown(Window &sender, Key keyCode) override . . . 232
10.47.3.15 OnKeyUp(Window &sender, Key keyCode) override . . . 232
10.47.3.16 OnLocalMotion(Window &sender, const Offset2D &position) override . . . 232
10.47.3.17 OnLostFocus(Window &sender) override . . . 232
10.47.3.18 OnProcessEvents(Window &sender) override . . . 232
10.47.3.19 OnWheelMotion(Window &sender, int motion) override . . . 232
10.48 LLGL::Interface Class Reference . . . 233
10.48.1 Detailed Description . . . 233
10.48.2 Member Function Documentation . . . 234
10.48.2.1 IsInstanceOf(int id) const . . . 234
10.49 LLGL::InterfaceID Struct Reference . . . 234
10.49.1 Detailed Description . . . 234
10.49.2 Member Enumeration Documentation . . . 235
10.49.2.1 anonymous enum . . . 235
10.50 LLGL::RenderingDebugger::Message Class Reference . . . 236
10.50.1 Detailed Description . . . 237
10.50.2 Constructor & Destructor Documentation . . . 237
10.50.2.1 Message()=default . . . 237
10.50.2.2 Message(const Message &)=default . . . 237
10.50.2.3 Message(const std::string &text, const std::string &source, const std::string &groupName) . . . 237

10.50.3 Member Function Documentation . . . 237
10.50.3.1 Block() . . . 237
10.50.3.2 BlockAfter(std::size_t occurrences) . . . 237
10.50.3.3 GetGroupName() const . . . 238
10.50.3.4 GetOccurrences() const . . . 238
10.50.3.5 GetSource() const . . . 238
10.50.3.6 GetText() const . . . 238
10.50.3.7 IncOccurrence() . . . 238
10.50.3.8 IsBlocked() const . . . 238
10.50.3.9 operator=(const Message &)=default . . . 238
10.50.3.10 ToReportString(const std::string &reportTypeName) const . . . 238
10.50.4 Friends And Related Function Documentation . . . 238
10.50.4.1 RenderingDebugger . . . 238
10.51 LLGL::MiscFlags Struct Reference . . . 239
10.51.1 Detailed Description . . . 239
10.51.2 Member Enumeration Documentation . . . 239
10.51.2.1 anonymous enum . . . 239
10.52 LLGL::MultiSamplingDescriptor Struct Reference . . . 240
10.52.1 Detailed Description . . . 240
10.52.2 Constructor & Destructor Documentation . . . 240
10.52.2.1 MultiSamplingDescriptor()=default . . . 240
10.52.2.2 MultiSamplingDescriptor(std::uint32_t samples, std::uint32_t sampleMask=~0) . . . 240
10.52.3 Member Function Documentation . . . 241
10.52.3.1 SampleCount() const . . . 241
10.52.4 Member Data Documentation . . . 241
10.52.4.1 enabled . . . 241
10.52.4.2 sampleMask . . . 241
10.52.4.3 samples . . . 241
10.53 LLGL::NativeContextHandle Struct Reference . . . 241
10.53.1 Detailed Description . . . 242

10.53.2 Member Data Documentation . . . 242
10.53.2.1 colorMap . . . 242
10.53.2.2 display . . . 242
10.53.2.3 parentView . . . 242
10.53.2.4 parentWindow . . . 242
10.53.2.5 parentWindow . . . 242
10.53.2.6 parentWindow . . . 242
10.53.2.7 screen . . . 242
10.53.2.8 visual . . . 242
10.54 LLGL::NativeHandle Struct Reference . . . 243
10.54.1 Detailed Description . . . 243
10.54.2 Member Data Documentation . . . 243
10.54.2.1 display . . . 243
10.54.2.2 view . . . 243
10.54.2.3 visual . . . 243
10.54.2.4 window . . . 243
10.54.2.5 window . . . 243
10.54.2.6 window . . . 243
10.55 LLGL::NonCopyable Class Reference . . . 244
10.55.1 Detailed Description . . . 244
10.55.2 Constructor & Destructor Documentation . . . 245
10.55.2.1 NonCopyable(const NonCopyable &)=delete . . . 245
10.55.2.2 ~NonCopyable()=default . . . 245
10.55.2.3 NonCopyable()=default . . . 245
10.55.3 Member Function Documentation . . . 245
10.55.3.1 operator=(const NonCopyable &)=delete . . . 245
10.56 LLGL::Offset2D Struct Reference . . . 245
10.56.1 Detailed Description . . . 245
10.56.2 Constructor & Destructor Documentation . . . 246
10.56.2.1 Offset2D()=default . . . 246

10.56.2.2 Offset2D(const Offset2D &)=default . . . 246
10.56.2.3 Offset2D(std::int32_t x, std::int32_t y) . . . 246
10.56.3 Member Data Documentation . . . 246
10.56.3.1 x . . . 246
10.56.3.2 y . . . 246
10.57 LLGL::Offset3D Struct Reference . . . 246
10.57.1 Detailed Description . . . 247
10.57.2 Constructor & Destructor Documentation . . . 247
10.57.2.1 Offset3D()=default . . . 247
10.57.2.2 Offset3D(const Offset3D &)=default . . . 247
10.57.2.3 Offset3D(std::int32_t x, std::int32_t y, std::int32_t z) . . . 247
10.57.3 Member Data Documentation . . . 247
10.57.3.1 x . . . 247
10.57.3.2 y . . . 247
10.57.3.3 z . . . 247
10.58 LLGL::OpenGLDependentStateDescriptor Struct Reference . . . 247
10.58.1 Detailed Description . . . 248
10.58.2 Member Data Documentation . . . 248
10.58.2.1 invertFrontFace . . . 248
10.58.2.2 originLowerLeft . . . 248
10.59 LLGL::PipelineLayout Class Reference . . . 248
10.59.1 Detailed Description . . . 249
10.60 LLGL::PipelineLayoutDescriptor Struct Reference . . . 249
10.60.1 Detailed Description . . . 249
10.60.2 Member Data Documentation . . . 249
10.60.2.1 bindings . . . 249
10.61 LLGL::QueryHeap Class Reference . . . 250
10.61.1 Detailed Description . . . 250
10.61.2 Constructor & Destructor Documentation . . . 250
10.61.2.1 QueryHeap(const QueryType type) . . . 250

10.61.3 Member Function Documentation . . . 250
10.61.3.1 GetType() const . . . 250
10.62 LLGL::QueryHeapDescriptor Struct Reference . . . 251
10.62.1 Detailed Description . . . 251
10.62.2 Member Data Documentation . . . 251
10.62.2.1 numQueries . . . 251
10.62.2.2 renderCondition . . . 251
10.62.2.3 type . . . 252
10.63 LLGL::QueryPipelineStatistics Struct Reference . . . 252
10.63.1 Detailed Description . . . 253
10.63.2 Member Data Documentation . . . 253
10.63.2.1 clippingInvocations . . . 253
10.63.2.2 clippingPrimitives . . . 253
10.63.2.3 computeShaderInvocations . . . 253
10.63.2.4 fragmentShaderInvocations . . . 253
10.63.2.5 geometryShaderInvocations . . . 253
10.63.2.6 geometryShaderPrimitives . . . 254
10.63.2.7 inputAssemblyPrimitives . . . 254
10.63.2.8 inputAssemblyVertices . . . 254
10.63.2.9 tessControlShaderInvocations . . . 254
10.63.2.10 tessEvaluationShaderInvocations . . . 254
10.63.2.11 vertexShaderInvocations . . . 254
10.64 LLGL::RasterizerDescriptor Struct Reference . . . 254
10.64.1 Detailed Description . . . 255
10.64.2 Member Data Documentation . . . 255
10.64.2.1 antiAliasedLineEnabled . . . 255
10.64.2.2 conservativeRasterization . . . 255
10.64.2.3 cullMode . . . 256
10.64.2.4 depthBias . . . 256
10.64.2.5 depthClampEnabled . . . 256

10.64.2.6 discardEnabled . . . 256
10.64.2.7 frontCCW . . . 256
10.64.2.8 lineWidth . . . 256
10.64.2.9 multiSampling . . . 257
10.64.2.10 polygonMode . . . 257
10.64.2.11 scissorTestEnabled . . . 257
10.65 LLGL::RenderContext Class Reference . . . 257
10.65.1 Detailed Description . . . 259
10.65.2 Constructor & Destructor Documentation . . . 259
10.65.2.1 RenderContext()=default . . . 259
10.65.2.2 RenderContext(const VideoModeDescriptor &initialVideoMode, const Vsync↩ Descriptor &initialVsync) . . . 259
10.65.3 Member Function Documentation . . . 259
10.65.3.1 GetColorFormat() const =0 . . . 259
10.65.3.2 GetDepthStencilFormat() const =0 . . . 260
10.65.3.3 GetNumColorAttachments() const final . . . 260
10.65.3.4 GetResolution() const final . . . 260
10.65.3.5 GetSurface() const . . . 260
10.65.3.6 GetVideoMode() const . . . 260
10.65.3.7 GetVsync() const . . . 261
10.65.3.8 HasDepthAttachment() const final . . . 261
10.65.3.9 HasStencilAttachment() const final . . . 261
10.65.3.10 OnSetVideoMode(const VideoModeDescriptor &videoModeDesc)=0 . . . 261
10.65.3.11 OnSetVsync(const VsyncDescriptor &vsyncDesc)=0 . . . 262
10.65.3.12 Present()=0 . . . 262
10.65.3.13 SetDisplayModeByVideoMode(Display &display, const VideoModeDescriptor &videoModeDesc) . . . 262
10.65.3.14 SetOrCreateSurface(const std::shared_ptr< Surface > &surface, VideoMode↩ Descriptor videoModeDesc, const void *windowContext) . . . 262
10.65.3.15 SetVideoMode(const VideoModeDescriptor &videoModeDesc) . . . 263
10.65.3.16 SetVsync(const VsyncDescriptor &vsyncDesc) . . . 263
10.65.3.17 ShareSurfaceAndConfig(RenderContext &other) . . . 263

10.65.3.18 SwitchFullscreenMode(const VideoModeDescriptor &videoModeDesc) . . . . . 264
10.66 LLGL::RenderContextDescriptor Struct Reference . . . . . 264
10.66.1 Detailed Description . . . . . 264
10.66.2 Member Data Documentation . . . . . 264
10.66.2.1 multiSampling . . . . . 264
10.66.2.2 videoMode . . . . . 265
10.66.2.3 vsync . . . . . 265
10.67 LLGL::RendererConfigurationOpenGL Struct Reference . . . . . 265
10.67.1 Detailed Description . . . . . 265
10.67.2 Member Data Documentation . . . . . 265
10.67.2.1 contextProfile . . . . . 265
10.67.2.2 majorVersion . . . . . 266
10.67.2.3 minorVersion . . . . . 266
10.68 LLGL::RendererConfigurationVulkan Struct Reference . . . . . 266
10.68.1 Detailed Description . . . . . 266
10.68.2 Member Data Documentation . . . . . 267
10.68.2.1 application . . . . . 267
10.68.2.2 enabledLayers . . . . . 267
10.68.2.3 minDeviceMemoryAllocationSize . . . . . 267
10.68.2.4 reduceDeviceMemoryFragmentation . . . . . 267
10.69 LLGL::RendererID Struct Reference . . . . . 268
10.69.1 Detailed Description . . . . . 268
10.69.2 Member Data Documentation . . . . . 269
10.69.2.1 Direct3D10 . . . . . 269
10.69.2.2 Direct3D11 . . . . . 269
10.69.2.3 Direct3D12 . . . . . 269
10.69.2.4 Direct3D9 . . . . . 269
10.69.2.5 Metal . . . . . 269
10.69.2.6 OpenGL . . . . . 269
10.69.2.7 OpenGLES1 . . . . . 269

10.69.2.8 OpenGLES2 . . . 269
10.69.2.9 OpenGLES3 . . . 269
10.69.2.10 Reserved . . . 269
10.69.2.11 Undefined . . . 270
10.69.2.12 Vulkan . . . 270
10.70 LLGL::RendererInfo Struct Reference . . . 270
10.70.1 Detailed Description . . . 270
10.70.2 Member Data Documentation . . . 270
10.70.2.1 deviceName . . . 270
10.70.2.2 rendererName . . . 270
10.70.2.3 shadingLanguageName . . . 271
10.70.2.4 vendorName . . . 271
10.71 LLGL::RenderingCapabilities Struct Reference . . . 271
10.71.1 Detailed Description . . . 271
10.71.2 Member Data Documentation . . . 271
10.71.2.1 clippingRange . . . 271
10.71.2.2 features . . . 272
10.71.2.3 limits . . . 272
10.71.2.4 screenOrigin . . . 272
10.71.2.5 shadingLanguages . . . 272
10.71.2.6 textureFormats . . . 272
10.72 LLGL::RenderingDebugger Class Reference . . . 273
10.72.1 Detailed Description . . . 273
10.72.2 Constructor & Destructor Documentation . . . 274
10.72.2.1 ~RenderingDebugger() . . . 274
10.72.3 Member Function Documentation . . . 274
10.72.3.1 OnError(ErrorType type, Message &message) . . . 274
10.72.3.2 OnWarning(WarningType type, Message &message) . . . 274
10.72.3.3 PostError(const ErrorType type, const std::string &message) . . . 274
10.72.3.4 PostWarning(const WarningType type, const std::string &message) . . . 275

10.72.3.5 SetDebugGroup(const char *name) . . . 275
10.72.3.6 SetSource(const char *source) . . . 275
10.73 LLGL::RenderingFeatures Struct Reference . . . 275
10.73.1 Detailed Description . . . 277
10.73.2 Member Data Documentation . . . 277
10.73.2.1 has3DTextures . . . 277
10.73.2.2 hasArrayTextures . . . 277
10.73.2.3 hasComputeShaders . . . 277
10.73.2.4 hasConservativeRasterization . . . 277
10.73.2.5 hasConstantBuffers . . . 278
10.73.2.6 hasCubeArrayTextures . . . 278
10.73.2.7 hasCubeTextures . . . 278
10.73.2.8 hasDirectResourceBinding . . . 278
10.73.2.9 hasGeometryShaders . . . 278
10.73.2.10 hasIndirectDrawing . . . 279
10.73.2.11 hasInstancing . . . 279
10.73.2.12 hasLogicOp . . . 279
10.73.2.13 hasMultiSampleTextures . . . 279
10.73.2.14 hasOffsetInstancing . . . 279
10.73.2.15 hasPipelineStatistics . . . 280
10.73.2.16 hasRenderCondition . . . 280
10.73.2.17 hasRenderTargets . . . 280
10.73.2.18 hasSamplers . . . 280
10.73.2.19 hasStorageBuffers . . . 280
10.73.2.20 hasStreamOutputs . . . 280
10.73.2.21 hasTessellationShaders . . . 281
10.73.2.22 hasTextureViews . . . 281
10.73.2.23 hasTextureViewSwizzle . . . 281
10.73.2.24 hasUniforms . . . 281
10.73.2.25 hasViewportArrays . . . 282

10.74 LLGL::RenderingLimits Struct Reference . . . 282
10.74.1 Detailed Description . . . 283
10.74.2 Member Data Documentation . . . 283
10.74.2.1 lineWidthRange . . . 283
10.74.2.2 max1DTextureSize . . . 283
10.74.2.3 max2DTextureSize . . . 283
10.74.2.4 max3DTextureSize . . . 283
10.74.2.5 maxAnisotropy . . . 284
10.74.2.6 maxBufferSize . . . 284
10.74.2.7 maxColorAttachments . . . 284
10.74.2.8 maxComputeShaderWorkGroups . . . 284
10.74.2.9 maxComputeShaderWorkGroupSize . . . 284
10.74.2.10 maxConstantBufferSize . . . 285
10.74.2.11 maxCubeTextureSize . . . 285
10.74.2.12 maxPatchVertices . . . 285
10.74.2.13 maxTextureArrayLayers . . . 285
10.74.2.14 maxViewports . . . 285
10.74.2.15 maxViewportSize . . . 286
10.75 LLGL::RenderingProfiler Class Reference . . . 286
10.75.1 Detailed Description . . . 286
10.75.2 Member Function Documentation . . . 286
10.75.2.1 Accumulate(const FrameProfile &profile) . . . 286
10.75.2.2 NextProfile(FrameProfile *outputProfile=nullptr) . . . 287
10.75.3 Member Data Documentation . . . 287
10.75.3.1 frameProfile . . . 287
10.76 LLGL::RenderPass Class Reference . . . 287
10.76.1 Detailed Description . . . 288
10.77 LLGL::RenderPassDescriptor Struct Reference . . . 288
10.77.1 Detailed Description . . . 288
10.77.2 Member Data Documentation . . . 289

10.77.2.1 colorAttachments . . . 289
10.77.2.2 depthAttachment . . . 289
10.77.2.3 stencilAttachment . . . 289
10.78 LLGL::RenderSystem Class Reference . . . 290
10.78.1 Detailed Description . . . 293
10.78.2 Constructor & Destructor Documentation . . . 293
10.78.2.1 RenderSystem()=default . . . 293
10.78.3 Member Function Documentation . . . 293
10.78.3.1 AssertCreateBuffer(const BufferDescriptor &desc, std::uint64_t maxSize) . . . 293
10.78.3.2 AssertCreateBufferArray(std::uint32_t numBuffers, Buffer ∗const ∗bufferArray) . . . 293
10.78.3.3 AssertCreateRenderPass(const RenderPassDescriptor &desc) . . . 293
10.78.3.4 AssertCreateRenderTarget(const RenderTargetDescriptor &desc) . . . 293
10.78.3.5 AssertCreateShader(const ShaderDescriptor &desc) . . . 293
10.78.3.6 AssertCreateShaderProgram(const ShaderProgramDescriptor &desc) . . . 294
10.78.3.7 AssertImageDataSize(std::size_t dataSize, std::size_t requiredDataSize, const char ∗info=nullptr) . . . 294
10.78.3.8 CreateBuffer(const BufferDescriptor &desc, const void ∗initialData=nullptr)=0 . . . 294
10.78.3.9 CreateBufferArray(std::uint32_t numBuffers, Buffer ∗const ∗bufferArray)=0 . . . 294
10.78.3.10 CreateCommandBuffer(const CommandBufferDescriptor &desc={})=0 . . . 295
10.78.3.11 CreateComputePipeline(const ComputePipelineDescriptor &desc)=0 . . . 295
10.78.3.12 CreateFence()=0 . . . 295
10.78.3.13 CreateGraphicsPipeline(const GraphicsPipelineDescriptor &desc)=0 . . . 295
10.78.3.14 CreatePipelineLayout(const PipelineLayoutDescriptor &desc)=0 . . . 296
10.78.3.15 CreateQueryHeap(const QueryHeapDescriptor &desc)=0 . . . 296
10.78.3.16 CreateRenderContext(const RenderContextDescriptor &desc, const std↩::shared_ptr< Surface > &surface={})=0 . . . 296
10.78.3.17 CreateRenderPass(const RenderPassDescriptor &desc)=0 . . . 297
10.78.3.18 CreateRenderTarget(const RenderTargetDescriptor &desc)=0 . . . 297
10.78.3.19 CreateResourceHeap(const ResourceHeapDescriptor &desc)=0 . . . 297
10.78.3.20 CreateSampler(const SamplerDescriptor &desc)=0 . . . 298
10.78.3.21 CreateShader(const ShaderDescriptor &desc)=0 . . . 298

10.78.3.22 CreateShaderProgram(const ShaderProgramDescriptor &desc)=0 . . . . . 298

10.78.3.23 CreateTexture(const TextureDescriptor &textureDesc, const SrcImageDescriptor *imageDesc=nullptr)=0 . . . . . 299

10.78.3.24 FindModules() . . . . . 299

10.78.3.25 GetCommandQueue()=0 . . . . . 299

10.78.3.26 GetConfiguration() const . . . . . 299

10.78.3.27 GetName() const . . . . . 300

10.78.3.28 GetRendererID() const . . . . . 300

10.78.3.29 GetRendererInfo() const . . . . . 300

10.78.3.30 GetRenderingCaps() const . . . . . 300

10.78.3.31 Load(const RenderSystemDescriptor &renderSystemDesc, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr) . . . . . 300

10.78.3.32 MapBuffer(Buffer &buffer, const CPUAccess access)=0 . . . . . 301

10.78.3.33 ReadTexture(const Texture &texture, std::uint32_t mipLevel, const DstImage↩ Descriptor &imageDesc)=0 . . . . . 301

10.78.3.34 Release(RenderContext &renderContext)=0 . . . . . 302

10.78.3.35 Release(CommandBuffer &commandBuffer)=0 . . . . . 303

10.78.3.36 Release(Buffer &buffer)=0 . . . . . 303

10.78.3.37 Release(BufferArray &bufferArray)=0 . . . . . 303

10.78.3.38 Release(Texture &texture)=0 . . . . . 303

10.78.3.39 Release(Sampler &sampler)=0 . . . . . 303

10.78.3.40 Release(ResourceHeap &resourceHeap)=0 . . . . . 303

10.78.3.41 Release(RenderPass &renderPass)=0 . . . . . 303

10.78.3.42 Release(RenderTarget &renderTarget)=0 . . . . . 303

10.78.3.43 Release(Shader &shader)=0 . . . . . 303

10.78.3.44 Release(ShaderProgram &shaderProgram)=0 . . . . . 304

10.78.3.45 Release(PipelineLayout &pipelineLayout)=0 . . . . . 304

10.78.3.46 Release(GraphicsPipeline &graphicsPipeline)=0 . . . . . 304

10.78.3.47 Release(ComputePipeline &computePipeline)=0 . . . . . 304

10.78.3.48 Release(QueryHeap &queryHeap)=0 . . . . . 304

10.78.3.49 Release(Fence &fence)=0 . . . . . 304

10.78.3.50 SetConfiguration(const RenderSystemConfiguration &config) . . . . . 304

10.78.3.51 SetRendererInfo(const RendererInfo &info) . . . 304
10.78.3.52 SetRenderingCaps(const RenderingCapabilities &caps) . . . 305
10.78.3.53 Unload(std::unique_ptr< RenderSystem > &&renderSystem) . . . 305
10.78.3.54 UnmapBuffer(Buffer &buffer)=0 . . . 305
10.78.3.55 WriteBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint64_t dataSize)=0 . . . 305
10.78.3.56 WriteTexture(Texture &texture, const TextureRegion &textureRegion, const SrcImageDescriptor &imageDesc)=0 . . . 305
10.79 LLGL::RenderSystemChild Class Reference . . . 306
10.79.1 Detailed Description . . . 307
10.79.2 Member Function Documentation . . . 307
10.79.2.1 SetName(const char *name) . . . 307
10.80 LLGL::RenderSystemConfiguration Struct Reference . . . 307
10.80.1 Detailed Description . . . 308
10.80.2 Member Data Documentation . . . 308
10.80.2.1 threadCount . . . 308
10.81 LLGL::RenderSystemDescriptor Struct Reference . . . 308
10.81.1 Detailed Description . . . 309
10.81.2 Constructor & Destructor Documentation . . . 309
10.81.2.1 RenderSystemDescriptor()=default . . . 309
10.81.2.2 RenderSystemDescriptor(const RenderSystemDescriptor &)=default . . . 309
10.81.2.3 RenderSystemDescriptor(const std::string &moduleName) . . . 309
10.81.2.4 RenderSystemDescriptor(const char *moduleName) . . . 309
10.81.3 Member Function Documentation . . . 309
10.81.3.1 operator=(const RenderSystemDescriptor &)=default . . . 309
10.81.4 Member Data Documentation . . . 309
10.81.4.1 debugCallback . . . 309
10.81.4.2 moduleName . . . 310
10.81.4.3 rendererConfig . . . 310
10.81.4.4 rendererConfigSize . . . 310
10.82 LLGL::RenderTarget Class Reference . . . 311

10.82.1 Detailed Description . . . 312
10.82.2 Member Function Documentation . . . 312
10.82.2.1 GetNumColorAttachments() const =0 . . . 312
10.82.2.2 GetRenderPass() const =0 . . . 312
10.82.2.3 GetResolution() const =0 . . . 313
10.82.2.4 HasDepthAttachment() const =0 . . . 313
10.82.2.5 HasStencilAttachment() const =0 . . . 313
10.82.2.6 IsRenderContext() const . . . 314
10.82.2.7 ValidateMipResolution(const Texture &texture, std::uint32_t mipLevel) . . . 314
10.82.2.8 ValidateResolution(const Extent2D &resolution) . . . 314
10.82.3 Friends And Related Function Documentation . . . 314
10.82.3.1 RenderContext . . . 314
10.83 LLGL::RenderTargetDescriptor Struct Reference . . . 314
10.83.1 Detailed Description . . . 315
10.83.2 Member Data Documentation . . . 315
10.83.2.1 attachments . . . 315
10.83.2.2 customMultiSampling . . . 316
10.83.2.3 multiSampling . . . 316
10.83.2.4 renderPass . . . 316
10.83.2.5 resolution . . . 316
10.84 LLGL::Resource Class Reference . . . 317
10.84.1 Detailed Description . . . 317
10.84.2 Member Function Documentation . . . 317
10.84.2.1 GetResourceType() const =0 . . . 317
10.85 LLGL::ResourceHeap Class Reference . . . 318
10.85.1 Detailed Description . . . 318
10.86 LLGL::ResourceHeapDescriptor Struct Reference . . . 318
10.86.1 Detailed Description . . . 319
10.86.2 Member Data Documentation . . . 319
10.86.2.1 pipelineLayout . . . 319

10.86.2.2 resourceViews . . . 319

10.87 LLGL::ResourceViewDescriptor Struct Reference . . . 320

10.87.1 Detailed Description . . . 320

10.87.2 Constructor & Destructor Documentation . . . 320

10.87.2.1 ResourceViewDescriptor()=default . . . 320

10.87.2.2 ResourceViewDescriptor(Resource *resource) . . . 320

10.87.2.3 ResourceViewDescriptor(Texture *texture, const TextureViewDescriptor &subresourceDesc) . . . 320

10.87.3 Member Data Documentation . . . 321

10.87.3.1 resource . . . 321

10.87.3.2 textureView . . . 321

10.88 LLGL::Sampler Class Reference . . . 321

10.88.1 Detailed Description . . . 322

10.88.2 Constructor & Destructor Documentation . . . 322

10.88.2.1 Sampler()=default . . . 322

10.88.3 Member Function Documentation . . . 322

10.88.3.1 GetResourceType() const override final . . . 322

10.89 LLGL::SamplerDescriptor Struct Reference . . . 322

10.89.1 Detailed Description . . . 323

10.89.2 Member Data Documentation . . . 323

10.89.2.1 addressModeU . . . 323

10.89.2.2 addressModeV . . . 323

10.89.2.3 addressModeW . . . 324

10.89.2.4 borderColor . . . 324

10.89.2.5 compareEnabled . . . 324

10.89.2.6 compareOp . . . 324

10.89.2.7 magFilter . . . 324

10.89.2.8 maxAnisotropy . . . 324

10.89.2.9 maxLOD . . . 324

10.89.2.10 minFilter . . . 324

10.89.2.11 minLOD . . . 324

10.89.2.12 mipMapFilter . . . 325
10.89.2.13 mipMapLODBias . . . 325
10.89.2.14 mipMapping . . . 325
10.90 LLGL::Scissor Struct Reference . . . 325
10.90.1 Detailed Description . . . 326
10.90.2 Constructor & Destructor Documentation . . . 326
10.90.2.1 Scissor()=default . . . 326
10.90.2.2 Scissor(const Scissor &)=default . . . 326
10.90.2.3 Scissor(std::int32_t x, std::int32_t y, std::int32_t width, std::int32_t height) . . . 326
10.90.2.4 Scissor(const Offset2D &offset, const Extent2D &extent) . . . 326
10.90.3 Member Data Documentation . . . 326
10.90.3.1 height . . . 326
10.90.3.2 width . . . 327
10.90.3.3 x . . . 327
10.90.3.4 y . . . 327
10.91 LLGL::Shader Class Reference . . . 327
10.91.1 Detailed Description . . . 328
10.91.2 Constructor & Destructor Documentation . . . 328
10.91.2.1 Shader(const ShaderType type) . . . 328
10.91.3 Member Function Documentation . . . 328
10.91.3.1 GetReport() const =0 . . . 328
10.91.3.2 GetStageFlags() const . . . 328
10.91.3.3 GetType() const . . . 328
10.91.3.4 HasErrors() const =0 . . . 329
10.92 LLGL::ShaderCompileFlags Struct Reference . . . 329
10.92.1 Detailed Description . . . 329
10.92.2 Member Enumeration Documentation . . . 329
10.92.2.1 anonymous enum . . . 329
10.93 LLGL::ShaderDescriptor Struct Reference . . . 330
10.93.1 Detailed Description . . . 331

10.93.2 Constructor & Destructor Documentation . . . 331
10.93.2.1 ShaderDescriptor()=default . . . 331
10.93.2.2 ShaderDescriptor(const ShaderDescriptor &)=default . . . 331
10.93.2.3 ShaderDescriptor(const ShaderType type, const char *source) . . . 331
10.93.2.4 ShaderDescriptor(const ShaderType type, const char *source, const char *entryPoint, const char *profile, long flags=0) . . . 331
10.93.3 Member Function Documentation . . . 331
10.93.3.1 operator=(const ShaderDescriptor &)=default . . . 331
10.93.4 Member Data Documentation . . . 331
10.93.4.1 defines . . . 331
10.93.4.2 entryPoint . . . 332
10.93.4.3 flags . . . 332
10.93.4.4 profile . . . 332
10.93.4.5 source . . . 333
10.93.4.6 sourceSize . . . 333
10.93.4.7 sourceType . . . 333
10.93.4.8 streamOutput . . . 333
10.93.4.9 type . . . 334
10.94 LLGL::ShaderMacro Struct Reference . . . 334
10.94.1 Detailed Description . . . 334
10.94.2 Constructor & Destructor Documentation . . . 334
10.94.2.1 ShaderMacro()=default . . . 334
10.94.2.2 ShaderMacro(const ShaderMacro &)=default . . . 334
10.94.2.3 ShaderMacro(const char *name, const char *definition=nullptr) . . . 334
10.94.3 Member Function Documentation . . . 335
10.94.3.1 operator=(const ShaderMacro &)=default . . . 335
10.94.4 Member Data Documentation . . . 335
10.94.4.1 definition . . . 335
10.94.4.2 name . . . 335
10.95 LLGL::ShaderProgram Class Reference . . . 335
10.95.1 Detailed Description . . . 336

10.95.2 Member Enumeration Documentation . . . 336
10.95.2.1 LinkError . . . 336
10.95.3 Member Function Documentation . . . 337
10.95.3.1 ClearShaderReflection(ShaderReflection &reflection) . . . 337
10.95.3.2 FinalizeShaderReflection(ShaderReflection &reflection) . . . 337
10.95.3.3 FindUniformLocation(const char ∗name) const =0 . . . 337
10.95.3.4 GetReport() const =0 . . . 338
10.95.3.5 GetWorkGroupSize(Extent3D &workGroupSize) const =0 . . . 338
10.95.3.6 HasErrors() const =0 . . . 338
10.95.3.7 LinkErrorToString(const LinkError errorCode) . . . 338
10.95.3.8 Reflect(ShaderReflection &reflection) const =0 . . . 338
10.95.3.9 SetWorkGroupSize(const Extent3D &workGroupSize)=0 . . . 339
10.95.3.10 ValidateShaderComposition(Shader ∗const ∗shaders, std::size_t numShaders) . 340
10.96 LLGL::ShaderProgramDescriptor Struct Reference . . . 340
10.96.1 Detailed Description . . . 341
10.96.2 Member Data Documentation . . . 341
10.96.2.1 computeShader . . . 341
10.96.2.2 fragmentShader . . . 341
10.96.2.3 geometryShader . . . 342
10.96.2.4 tessControlShader . . . 342
10.96.2.5 tessEvaluationShader . . . 342
10.96.2.6 vertexFormats . . . 342
10.96.2.7 vertexShader . . . 343
10.97 LLGL::ShaderReflection Struct Reference . . . 343
10.97.1 Detailed Description . . . 343
10.97.2 Member Data Documentation . . . 344
10.97.2.1 resources . . . 344
10.97.2.2 streamOutputAttributes . . . 344
10.97.2.3 uniforms . . . 344
10.97.2.4 vertexAttributes . . . 344

10.98 LLGL::ShaderResource Struct Reference . . . 344
10.98.1 Detailed Description . . . 345
10.98.2 Member Data Documentation . . . 345
10.98.2.1 binding . . . 345
10.98.2.2 constantBufferSize . . . 345
10.98.2.3 storageBufferType . . . 345
10.99 LLGL::ShaderUniform Struct Reference . . . 346
10.99.1 Detailed Description . . . 346
10.99.2 Member Data Documentation . . . 346
10.99.2.1 location . . . 346
10.99.2.2 name . . . 346
10.99.2.3 size . . . 346
10.99.2.4 type . . . 346
10.100 LLGL::SrcImageDescriptor Struct Reference . . . 347
10.100.1 Detailed Description . . . 347
10.100.2 Constructor & Destructor Documentation . . . 347
10.100.2.1 SrcImageDescriptor()=default . . . 347
10.100.2.2 SrcImageDescriptor(const SrcImageDescriptor &)=default . . . 347
10.100.2.3 SrcImageDescriptor(ImageFormat format, DataType dataType, const void *data, std::size_t dataSize) . . . 347
10.100.3 Member Data Documentation . . . 348
10.100.3.1 data . . . 348
10.100.3.2 dataSize . . . 348
10.100.3.3 dataType . . . 348
10.100.3.4 format . . . 348
10.101 LLGL::StageFlags Struct Reference . . . 348
10.101.1 Detailed Description . . . 348
10.101.2 Member Enumeration Documentation . . . 349
10.101.2.1 anonymous enum . . . 349
10.102 LLGL::StencilDescriptor Struct Reference . . . 349
10.102.1 Detailed Description . . . 349

10.102.2 Member Data Documentation . . . 350
10.102.2.1 back . . . 350
10.102.2.2 front . . . 350
10.102.2.3 testEnabled . . . 350
10.103 LGL::StencilFaceDescriptor Struct Reference . . . 350
10.103.1 Detailed Description . . . 351
10.103.2 Member Data Documentation . . . 351
10.103.2.1 compareOp . . . 351
10.103.2.2 depthFailOp . . . 351
10.103.2.3 depthPassOp . . . 351
10.103.2.4 readMask . . . 351
10.103.2.5 reference . . . 351
10.103.2.6 stencilFailOp . . . 352
10.103.2.7 writeMask . . . 352
10.104 LGL::BufferDescriptor::StorageBuffer Struct Reference . . . 352
10.104.1 Detailed Description . . . 352
10.104.2 Member Data Documentation . . . 353
10.104.2.1 format . . . 353
10.104.2.2 storageType . . . 353
10.104.2.3 stride . . . 353
10.105 LGL::ShaderDescriptor::StreamOutput Struct Reference . . . 353
10.105.1 Detailed Description . . . 354
10.105.2 Member Data Documentation . . . 354
10.105.2.1 format . . . 354
10.106 LGL::StreamOutputAttribute Struct Reference . . . 354
10.106.1 Detailed Description . . . 355
10.106.2 Constructor & Destructor Documentation . . . 355
10.106.2.1 StreamOutputAttribute()=default . . . 355
10.106.2.2 StreamOutputAttribute(const StreamOutputAttribute &)=default . . . 355
10.106.3 Member Function Documentation . . . 355

10.106.3.1 operator=(const StreamOutputAttribute &)=default . . . 355
10.106.4 Member Data Documentation . . . 355
10.106.4.1 components . . . 355
10.106.4.2 name . . . 355
10.106.4.3 outputSlot . . . 355
10.106.4.4 semanticIndex . . . 356
10.106.4.5 startComponent . . . 356
10.106.4.6 stream . . . 356
10.107 LLGL::StreamOutputFormat Struct Reference . . . 356
10.107.1 Detailed Description . . . 356
10.107.2 Member Function Documentation . . . 356
10.107.2.1 AppendAttribute(const StreamOutputAttribute &attrib) . . . 356
10.107.2.2 AppendAttributes(const StreamOutputFormat &format) . . . 357
10.107.3 Member Data Documentation . . . 357
10.107.3.1 attributes . . . 357
10.108 LLGL::Surface Class Reference . . . 357
10.108.1 Detailed Description . . . 358
10.108.2 Member Function Documentation . . . 358
10.108.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc)=0 . . . 358
10.108.2.2 GetContentSize() const =0 . . . 359
10.108.2.3 GetNativeHandle(void *nativeHandle) const =0 . . . 359
10.108.2.4 ProcessEvents()=0 . . . 359
10.108.2.5 ResetPixelFormat()=0 . . . 359
10.109 LLGL::Texture Class Reference . . . 360
10.109.1 Detailed Description . . . 360
10.109.2 Constructor & Destructor Documentation . . . 361
10.109.2.1 Texture(const TextureType type) . . . 361
10.109.3 Member Function Documentation . . . 361
10.109.3.1 GetDesc() const =0 . . . 361
10.109.3.2 GetMipExtent(std::uint32_t mipLevel) const =0 . . . 361

10.109.3.3 GetResourceType() const override final . . . 362
10.109.3.4 GetType() const . . . 362
10.110 LLGL::TextureDescriptor Struct Reference . . . 362
10.110.1 Detailed Description . . . 363
10.110.2 Member Data Documentation . . . 363
10.110.2.1 arrayLayers . . . 363
10.110.2.2 bindFlags . . . 363
10.110.2.3 clearValue . . . 364
10.110.2.4 cpuAccessFlags . . . 364
10.110.2.5 extent . . . 364
10.110.2.6 format . . . 365
10.110.2.7 mipLevels . . . 365
10.110.2.8 miscFlags . . . 365
10.110.2.9 samples . . . 365
10.110.2.10 type . . . 366
10.111 LLGL::TextureLocation Struct Reference . . . 366
10.111.1 Detailed Description . . . 366
10.111.2 Constructor & Destructor Documentation . . . 367
10.111.2.1 TextureLocation()=default . . . 367
10.111.2.2 TextureLocation(const TextureLocation &)=default . . . 367
10.111.2.3 TextureLocation(const Offset3D &offset, std::uint32_t arrayLayer=0, std::uint32↩ _t mipLevel=0) . . . 367
10.111.3 Member Data Documentation . . . 367
10.111.3.1 arrayLayer . . . 367
10.111.3.2 mipLevel . . . 367
10.111.3.3 offset . . . 367
10.112 LLGL::TextureRegion Struct Reference . . . 368
10.112.1 Detailed Description . . . 368
10.112.2 Constructor & Destructor Documentation . . . 368
10.112.2.1 TextureRegion()=default . . . 368
10.112.2.2 TextureRegion(const TextureRegion &)=default . . . 368

10.112.2.3 TextureRegion(const Offset3D &offset, const Extent3D &extent) . . . 368
10.112.3 Member Data Documentation . . . 369
10.112.3.1 extent . . . 369
10.112.3.2 offset . . . 369
10.112.3.3 subresource . . . 369
10.113 LGL::TextureSubresource Struct Reference . . . 369
10.113.1 Detailed Description . . . 370
10.113.2 Constructor & Destructor Documentation . . . 370
10.113.2.1 TextureSubresource()=default . . . 370
10.113.2.2 TextureSubresource(const TextureSubresource &)=default . . . 370
10.113.2.3 TextureSubresource(std::uint32_t baseArrayLayer, std::uint32_t baseMipLevel) . . . 370
10.113.2.4 TextureSubresource(std::uint32_t baseArrayLayer, std::uint32_t numArrayLayers, std::uint32_t baseMipLevel, std::uint32_t numMipLevels) . . . 370
10.113.3 Member Data Documentation . . . 371
10.113.3.1 baseArrayLayer . . . 371
10.113.3.2 baseMipLevel . . . 371
10.113.3.3 numArrayLayers . . . 371
10.113.3.4 numMipLevels . . . 371
10.114 LGL::TextureSwizzleRGBA Struct Reference . . . 372
10.114.1 Detailed Description . . . 372
10.114.2 Member Data Documentation . . . 372
10.114.2.1 a . . . 372
10.114.2.2 b . . . 372
10.114.2.3 g . . . 372
10.114.2.4 r . . . 373
10.115 LGL::TextureViewDescriptor Struct Reference . . . 373
10.115.1 Detailed Description . . . 373
10.115.2 Member Data Documentation . . . 373
10.115.2.1 format . . . 373
10.115.2.2 subresource . . . 374
10.115.2.3 swizzle . . . 374

10.115.2.4 type . . . 374
10.116 LGL::Timer Class Reference . . . 375
10.116.1 Detailed Description . . . 376
10.116.2 Member Function Documentation . . . 376
10.116.2.1 Create() . . . 376
10.116.2.2 GetDeltaTime() const . . . 376
10.116.2.3 GetFrequency() const =0 . . . 376
10.116.2.4 IsRunning() const =0 . . . 376
10.116.2.5 MeasureTime() . . . 377
10.116.2.6 Start()=0 . . . 377
10.116.2.7 Stop()=0 . . . 377
10.117 LGL::UninitializeTag Struct Reference . . . 377
10.117.1 Detailed Description . . . 377
10.118 LGL::VertexAttribute Struct Reference . . . 378
10.118.1 Detailed Description . . . 378
10.118.2 Constructor & Destructor Documentation . . . 378
10.118.2.1 VertexAttribute()=default . . . 378
10.118.2.2 VertexAttribute(const VertexAttribute &)=default . . . 378
10.118.2.3 VertexAttribute(const std::string &name, const Format format, std::uint32_t instanceDivisor=0) . . . 378
10.118.2.4 VertexAttribute(const std::string &semanticName, std::uint32_t semanticIndex, const Format format, std::uint32_t instanceDivisor=0) . . . 379
10.118.3 Member Function Documentation . . . 379
10.118.3.1 GetSize() const . . . 379
10.118.3.2 operator=(const VertexAttribute &)=default . . . 380
10.118.4 Member Data Documentation . . . 380
10.118.4.1 format . . . 380
10.118.4.2 instanceDivisor . . . 380
10.118.4.3 name . . . 380
10.118.4.4 offset . . . 380
10.118.4.5 semanticIndex . . . 380

10.118.4.6 systemValue . . . 381

10.119 LLGL::BufferDescriptor::VertexBuffer Struct Reference . . . 381

10.119.1 Detailed Description . . . 381

10.119.2 Member Data Documentation . . . 381

10.119.2.1 format . . . 381

10.120 LLGL::VertexFormat Struct Reference . . . 382

10.120.1 Detailed Description . . . 382

10.120.2 Member Function Documentation . . . 382

10.120.2.1 AppendAttribute(const VertexAttribute &attrib, std::uint32_t offset=Constants↩ ::ignoreOffset) . . . 382

10.120.3 Member Data Documentation . . . 383

10.120.3.1 attributes . . . 383

10.120.3.2 inputSlot . . . 383

10.120.3.3 stride . . . 383

10.121 LLGL::VideoAdapterDescriptor Struct Reference . . . 384

10.121.1 Detailed Description . . . 384

10.121.2 Member Data Documentation . . . 384

10.121.2.1 name . . . 384

10.121.2.2 outputs . . . 384

10.121.2.3 vendor . . . 384

10.121.2.4 videoMemory . . . 385

10.122 LLGL::VideoModeDescriptor Struct Reference . . . 385

10.122.1 Detailed Description . . . 385

10.122.2 Member Data Documentation . . . 386

10.122.2.1 colorBits . . . 386

10.122.2.2 depthBits . . . 386

10.122.2.3 fullscreen . . . 386

10.122.2.4 resolution . . . 386

10.122.2.5 stencilBits . . . 387

10.122.2.6 swapChainSize . . . 387

10.123 LLGL::VideoOutputDescriptor Struct Reference . . . 387

10.123.1 Detailed Description . . . 387
10.123.2 Member Data Documentation . . . 388
10.123.2.1 displayModes . . . 388
10.124 LGL::Viewport Struct Reference . . . 388
10.124.1 Detailed Description . . . 389
10.124.2 Constructor & Destructor Documentation . . . 389
10.124.2.1 Viewport()=default . . . 389
10.124.2.2 Viewport(const Viewport &)=default . . . 389
10.124.2.3 Viewport(float x, float y, float width, float height) . . . 389
10.124.2.4 Viewport(float x, float y, float width, float height, float minDepth, float maxDepth) . 389
10.124.2.5 Viewport(const Extent2D &extent) . . . 389
10.124.2.6 Viewport(const Extent2D &extent, float minDepth, float maxDepth) . . . 389
10.124.2.7 Viewport(const Offset2D &offset, const Extent2D &extent) . . . 389
10.124.2.8 Viewport(const Offset2D &offset, const Extent2D &extent, float minDepth, float maxDepth) . . . 389
10.124.3 Member Data Documentation . . . 390
10.124.3.1 height . . . 390
10.124.3.2 maxDepth . . . 390
10.124.3.3 minDepth . . . 390
10.124.3.4 width . . . 390
10.124.3.5 x . . . 390
10.124.3.6 y . . . 390
10.125 LGL::VsyncDescriptor Struct Reference . . . 391
10.125.1 Detailed Description . . . 391
10.125.2 Member Data Documentation . . . 391
10.125.2.1 enabled . . . 391
10.125.2.2 interval . . . 391
10.125.2.3 refreshRate . . . 391
10.126 LGL::Window Class Reference . . . 392
10.126.1 Detailed Description . . . 393
10.126.2 Member Function Documentation . . . 394

10.126.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc) override . . . . . 394
10.126.2.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 394
10.126.2.3 Create(const WindowDescriptor &desc) . . . . . 394
10.126.2.4 GetBehavior() const . . . . . 394
10.126.2.5 GetDesc() const =0 . . . . . 394
10.126.2.6 GetPosition() const =0 . . . . . 395
10.126.2.7 GetSize(bool useClientArea=true) const =0 . . . . . 395
10.126.2.8 GetTitle() const =0 . . . . . 395
10.126.2.9 HasFocus() const . . . . . 395
10.126.2.10 HasQuit() const . . . . . 395
10.126.2.11 IsShown() const =0 . . . . . 395
10.126.2.12 OnProcessEvents()=0 . . . . . 395
10.126.2.13 PostChar(wchar_t chr) . . . . . 395
10.126.2.14 PostDoubleClick(Key keyCode) . . . . . 396
10.126.2.15 PostGetFocus() . . . . . 396
10.126.2.16 PostGlobalMotion(const Offset2D &motion) . . . . . 396
10.126.2.17 PostKeyDown(Key keyCode) . . . . . 396
10.126.2.18 PostKeyUp(Key keyCode) . . . . . 396
10.126.2.19 PostLocalMotion(const Offset2D &position) . . . . . 396
10.126.2.20 PostLostFocus() . . . . . 396
10.126.2.21 PostQuit() . . . . . 397
10.126.2.22 PostResize(const Extent2D &clientAreaSize) . . . . . 397
10.126.2.23 PostTimer(std::uint32_t timerID) . . . . . 397
10.126.2.24 PostWheelMotion(int motion) . . . . . 397
10.126.2.25 ProcessEvents() override final . . . . . 397
10.126.2.26 RemoveEventListener(const EventListener *eventListener) . . . . . 398
10.126.2.27 SetBehavior(const WindowBehavior &behavior) . . . . . 398
10.126.2.28 SetDesc(const WindowDescriptor &desc)=0 . . . . . 398
10.126.2.29 SetPosition(const Offset2D &position)=0 . . . . . 398
10.126.2.30 SetSize(const Extent2D &size, bool useClientArea=true)=0 . . . . . 398

10.126.2.31 SetTitle(const std::wstring &title)=0 . . . 398

10.126.2.32 Show(bool show=true)=0 . . . 398

10.127 LGL::WindowBehavior Struct Reference . . . 399

10.127.1 Detailed Description . . . 399

10.127.2 Member Data Documentation . . . 399

10.127.2.1 disableClearOnResize . . . 399

10.127.2.2 moveAndResizeTimerID . . . 399

10.128 LGL::WindowDescriptor Struct Reference . . . 400

10.128.1 Detailed Description . . . 400

10.128.2 Member Data Documentation . . . 400

10.128.2.1 acceptDropFiles . . . 400

10.128.2.2 borderless . . . 400

10.128.2.3 centered . . . 401

10.128.2.4 position . . . 401

10.128.2.5 preventForPowerSafe . . . 401

10.128.2.6 resizable . . . 401

10.128.2.7 size . . . 402

10.128.2.8 title . . . 402

10.128.2.9 visible . . . 402

10.128.2.10 windowContext . . . 402

**11 File Documentation** 403

11.1 Buffer.h File Reference . . . 403

11.2 BufferArray.h File Reference . . . 403

11.3 BufferFlags.h File Reference . . . 404

11.4 Canvas.h File Reference . . . 405

11.5 CanvasFlags.h File Reference . . . 405

11.6 Color.h File Reference . . . 405

11.7 ColorRGB.h File Reference . . . 407

11.8 ColorRGBA.h File Reference . . . 407

11.9 CommandBuffer.h File Reference . . . 408

11.10 CommandBufferFlags.h File Reference . . . 408

11.11 CommandQueue.h File Reference . . . 409

11.12 ComputePipeline.h File Reference . . . 409

11.13 ComputePipelineFlags.h File Reference . . . 410

11.14 Constants.h File Reference . . . 410

11.15 Display.h File Reference . . . 410

11.16 DisplayFlags.h File Reference . . . 410

11.17 Export.h File Reference . . . 411

11.17.1 Macro Definition Documentation . . . 411

11.17.1.1 LLGL_EXPORT . . . 411

11.18 Fence.h File Reference . . . 411

11.19 Format.h File Reference . . . 412

11.20 ForwardDecls.h File Reference . . . 413

11.21 FragmentAttribute.h File Reference . . . 414

11.22 GraphicsPipeline.h File Reference . . . 414

11.23 GraphicsPipelineFlags.h File Reference . . . 414

11.24 Image.h File Reference . . . 417

11.25 ImageFlags.h File Reference . . . 417

11.26 IndirectArguments.h File Reference . . . 418

11.27 Input.h File Reference . . . 419

11.28 Interface.h File Reference . . . 419
11.28.1 Macro Definition Documentation . . . 420
11.28.1.1 LLGL_DECLARE_INTERFACE . . . 420
11.28.1.2 LLGL_IMPLEMENT_INTERFACE . . . 420
11.29 InterfaceID.h File Reference . . . 420
11.30 IOSNativeHandle.h File Reference . . . 420
11.31 Key.h File Reference . . . 421
11.32 LinuxNativeHandle.h File Reference . . . 422
11.33 LLGL.h File Reference . . . 422
11.34 Log.h File Reference . . . 422
11.35 MacOSNativeHandle.h File Reference . . . 423
11.36 NativeHandle.h File Reference . . . 423
11.37 NonCopyable.h File Reference . . . 424
11.38 PipelineLayout.h File Reference . . . 424
11.39 PipelineLayoutFlags.h File Reference . . . 424
11.40 Platform.h File Reference . . . 425
11.40.1 Macro Definition Documentation . . . 425
11.40.1.1 LLGL_API . . . 425
11.40.1.2 LLGL_API_CDECL . . . 425
11.40.1.3 LLGL_API_STDCALL . . . 425
11.40.1.4 LLGL_API_THISCALL . . . 425
11.41 QueryHeap.h File Reference . . . 425
11.42 QueryHeapFlags.h File Reference . . . 425
11.43 RenderContext.h File Reference . . . 426
11.44 RenderContextFlags.h File Reference . . . 427
11.45 RendererConfiguration.h File Reference . . . 427
11.46 RenderingDebugger.h File Reference . . . 428
11.47 RenderingProfiler.h File Reference . . . 428
11.48 RenderPass.h File Reference . . . 429
11.49 RenderPassFlags.h File Reference . . . 429

11.50 RenderSystem.h File Reference . . . 430
11.51 RenderSystemChild.h File Reference . . . 431
11.52 RenderSystemFlags.h File Reference . . . 431
11.53 RenderTarget.h File Reference . . . 433
11.54 RenderTargetFlags.h File Reference . . . 433
11.55 Resource.h File Reference . . . 433
11.56 ResourceFlags.h File Reference . . . 434
11.57 ResourceHeap.h File Reference . . . 434
11.58 ResourceHeapFlags.h File Reference . . . 435
11.59 Sampler.h File Reference . . . 435
11.60 SamplerFlags.h File Reference . . . 435
11.61 Shader.h File Reference . . . 436
11.62 ShaderFlags.h File Reference . . . 436
11.63 ShaderProgram.h File Reference . . . 437
11.64 ShaderProgramFlags.h File Reference . . . 438
11.65 StreamOutputAttribute.h File Reference . . . 439
11.66 StreamOutputFormat.h File Reference . . . 440
11.67 Strings.h File Reference . . . 440
11.68 Surface.h File Reference . . . 441
11.69 SystemValue.h File Reference . . . 441
11.70 Tags.h File Reference . . . 441
11.71 Texture.h File Reference . . . 442
11.72 TextureFlags.h File Reference . . . 442
11.73 Timer.h File Reference . . . 443
11.74 TypeInfo.h File Reference . . . 444
11.75 Types.h File Reference . . . 444
11.76 Utility.h File Reference . . . 445
11.77 Version.h File Reference . . . 447
11.78 VertexAttribute.h File Reference . . . 447
11.79 VertexFormat.h File Reference . . . 448
11.80 VideoAdapter.h File Reference . . . 448
11.81 Win32NativeHandle.h File Reference . . . 449
11.82 Window.h File Reference . . . 449
11.83 WindowFlags.h File Reference . . . 450

**Index** 451

# Chapter 1

# LLGL 0.03 Beta Documentation

## LLGL (Low Level Graphics Library)

### Overview

- **Version**: 0.03 Beta
- **License**: `3-Clause BSD License`

### Progress

- **OpenGL Renderer**: ~90% done
- **Direct3D 11 Renderer**: ~90% done
- **Direct3D 12 Renderer**: ~40% done
- **Vulkan Renderer**: ~50% done
- **Metal enderer**: ~50% done

### 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

```
// Unified Interface:
CommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex);

// OpenGL Implementation:
void GLCommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex) {
    const GLintptr indices = (renderState_.indexBufferOffset + firstIndex * renderState_.indexBufferStride)
      ;
    glDrawElements(
        renderState_.drawMode,
        static_cast<GLsizei>(numIndices),
        renderState_.indexBufferDataType,
        reinterpret_cast<const GLvoid*>(indices)
    );
}

// Direct3D 11 Implementation
void D3D11CommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex) {
    context_->DrawIndexed(numIndices, firstIndex, 0);
}

// Direct3D 12 Implementation
void D3D12CommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex) {
    commandList_->DrawIndexedInstanced(numIndices, 1, firstIndex, 0, 0);
}

// Vulkan Implementation
void VKCommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex) {
    vkCmdDrawIndexed(commandBuffer_, numIndices, 1, firstIndex, 0, 0);
}

// Metal implementation
void MTCommandBuffer::DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex) {
    if (numPatchControlPoints_ > 0) {
        [renderEncoder_
            drawIndexedPatches:             numPatchControlPoints_
            patchStart:                     static_cast<NSUInteger>(firstIndex) / numPatchControlPoints_
            patchCount:                     static_cast<NSUInteger>(numIndices) / numPatchControlPoints_
            patchIndexBuffer:               nil
            patchIndexBufferOffset:         0
            controlPointIndexBuffer:        indexBuffer_
            controlPointIndexBufferOffset:  indexBufferOffset_ + indexTypeSize_ * (static_cast<NSUInteger>(
      firstIndex))
            instanceCount:                  1
            baseInstance:                   0
        ];
    } else {
        [renderEncoder_
            drawIndexedPrimitives:  primitiveType_
            indexCount:             static_cast<NSUInteger>(numIndices)
            indexType:              indexType_
            indexBuffer:            indexBuffer_
            indexBufferOffset:      indexBufferOffset_ + indexTypeSize_ * static_cast<NSUInteger>(
      firstIndex)
        ];
    }
}
```

# Chapter 2

# Todo List

**Member LLGL::BlendDescriptor::blendFactor**

Move this into a dynamic function "CommandBuffer::SetBlendFactor".

**Class LLGL::BufferDescriptor::StorageBuffer**

Move this into a BufferViewDescriptor.

**Member LLGL::CommandBuffer::Execute (CommandBuffer &deferredCommandBuffer)=0**

Incomplete for: D3D12, Vulkan, Metal.

**Member LLGL::DataType**

Maybe replace `DataType` and `ImageFormat` by LLGL::Format.

**Member LLGL::ESProfile**

Not supported yet

**Member LLGL::FrameProfile::bufferReads**

Not available yet.

**Member LLGL::FrameProfile::textureMappings**

Not available yet.

**Member LLGL::Image::Fill (Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor)**

Not implemented yet.

**Member LLGL::Image::MirrorXYPlane ()**

Not implemented yet

**Member LLGL::Image::MirrorXZPlane ()**

Not implemented yet

**Member LLGL::Image::MirrorYZPlane ()**

Not implemented yet

**Member LLGL::Image::Resize (const Extent3D &extent, const SamplerFilter filter)**

Not implemented yet.

**Member LLGL::ImageFormat**

Maybe replace `DataType` and `ImageFormat` by LLGL::Format.

**Member LLGL::MiscFlags::DynamicUsage**

Restriction required to support deferred context in D3D11. This must no longer be just a "hint", it must be a strictly defined attribute for a buffer.

**Member LLGL::RendererConfigurationVulkan::minDeviceMemoryAllocationSize**

Remove this as soon as Vulkan memory manage has been improved.

**Member LLGL::RendererConfigurationVulkan::reduceDeviceMemoryFragmentation**

Remove this as soon as Vulkan memory manage has been improved.

**Class LLGL::RenderingDebugger::Message**

Rename to "Report"

**Class LLGL::RenderingProfiler**

Refactor this for the new ResourceHeap and RenderPass interfaces.

**Member LLGL::RenderPassDescriptor::colorAttachments**

Change to static array with 8 elements, i.e. `AttachmentFormatDescriptor colorAttachments[8]`.

**Member LLGL::RenderSystem::ReadTexture (const Texture &texture, std::uint32_t mipLevel, const Dst↩ImageDescriptor &imageDesc)=0**

Replace `mipLevel` parameter with `textureRegion` just like with the `WriteTexture` function.

**Member LLGL::RenderTarget::IsRenderContext () const**

Replace by IsInstanceOf

**Member LLGL::Resource::GetResourceType () const =0**

(Maybe) replace by IsInstanceOf

**Class LLGL::ResourceHeap**

Maybe rename to "ResourceViewHeap" again?

**Member LLGL::ResourceViewDescriptor::textureView**

Not implemented yet.

**Member LLGL::SamplerDescriptor::mipMapping**

Rename to `mipMapEnabled`.

**Member LLGL::Shader::GetReport () const =0**

Change return value to std::unique_ptr<LLGL::Blob>

**Member LLGL::ShaderProgram::LinkError**

Turn this into a global common enumeration for error codes.

**Member LLGL::StorageBufferType**

Replace by BindFlag with different options (e.g. StructuredBuffer and Storage).

**Member LLGL::TextureDescriptor::cpuAccessFlags**

Not supported yet.

**Class LLGL::VertexFormat**

Remove this struct and move both `stride` and `inputSlot` into `VertexAttribute`.

**Class LLGL::VideoAdapterDescriptor**

Currently unused in the interface.

**Class LLGL::VideoOutputDescriptor**

Currently unused in the interface.

**Class LLGL::VsyncDescriptor**

Maybe remove this entire structure and only use a "vsyncInterval" parameter.

# Chapter 3

# Module Index

## 3.1 Modules

Here is a list of all modules:


Global functions for Strict-Weak-Order (SWO) comparisons. . . . . 21
Global operators for basic data structures. . . . . 22
Global type aliases to callback interfaces. . . . . 25
Buffer utility functions to determine buffer types. . . . . 27
Hardware format utility functions. . . . . 28
Data type utility functions. . . . . 31
Image utility functions to classify and convert image data. . . . . 32
Buffer argument structures for indirect draw and compute commands. . . . . 37
Global type-to-string conversion functions. . . . . 38
Texture utility functions to determine texture dimension and buffer sizes. . . . . 40
Global utility functions, especially to fill descriptor structures. . . . . 44

# Chapter 4

# Namespace Index

## 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:


LLGL . . . . . 51

LLGL::Constants

Namespace with all constants used as default arguments . . . . . 101

LLGL::Log . . . . . 101

LLGL::Version

Namespace with functions to determine LLGL version . . . . . 104

# Chapter 5

# Hierarchical Index

## 5.1 Class Hierarchy

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


LLGL::ApplicationDescriptor . . . 107
LLGL::AttachmentClear . . . 108
LLGL::AttachmentDescriptor . . . 110
LLGL::AttachmentFormatDescriptor . . . 112
LLGL::BindFlags . . . 114
LLGL::BindingDescriptor . . . 116
LLGL::BlendDescriptor . . . 119
LLGL::BlendTargetDescriptor . . . 121
LLGL::BufferDescriptor . . . 125
LLGL::CanvasDescriptor . . . 131
LLGL::ClearFlags . . . 132
LLGL::ClearValue . . . 132
LLGL::Color< T, N > . . . 133
LLGL::Color< bool > . . . 133
LLGL::Color< float > . . . 133
LLGL::Color< T, 3u > . . . 137
LLGL::Color< T, 4u > . . . 142
LLGL::CommandBufferDescriptor . . . 173
LLGL::CommandBufferFlags . . . 174
LLGL::ComputePipelineDescriptor . . . 178
LLGL::CPUAccessFlags . . . 179
LLGL::DepthBiasDescriptor . . . 180
LLGL::DepthDescriptor . . . 181
LLGL::DispatchIndirectArguments . . . 182
LLGL::DisplayModeDescriptor . . . 187
LLGL::DrawIndexedIndirectArguments . . . 188
LLGL::DrawIndirectArguments . . . 190
LLGL::DrawPatchIndirectArgument . . . 191
LLGL::DstImageDescriptor . . . 193
LLGL::Extent2D . . . 199
LLGL::Extent3D . . . 200
LLGL::FormatDescriptor . . . 202
LLGL::FragmentAttribute . . . 205
LLGL::FrameProfile . . . 206
LLGL::GraphicsPipelineDescriptor . . . 216

LLGL::Image . . . 218
LLGL::BufferDescriptor::IndexBuffer . . . 228
LLGL::InterfaceID . . . 234
LLGL::RenderingDebugger::Message . . . 236
LLGL::MiscFlags . . . 239
LLGL::MultiSamplingDescriptor . . . 240
LLGL::NativeContextHandle . . . 241
LLGL::NativeHandle . . . 243
LLGL::NonCopyable . . . 244
- LLGL::Interface . . . 233
  - LLGL::Canvas::EventListener . . . 198
  - LLGL::Display . . . 183
  - LLGL::RenderSystem . . . 290
  - LLGL::RenderSystemChild . . . 306
    - LLGL::BufferArray . . . 124
    - LLGL::CommandBuffer . . . 147
    - LLGL::CommandQueue . . . 175
    - LLGL::ComputePipeline . . . 178
    - LLGL::Fence . . . 202
    - LLGL::GraphicsPipeline . . . 215
    - LLGL::PipelineLayout . . . 248
    - LLGL::QueryHeap . . . 250
    - LLGL::RenderPass . . . 287
    - LLGL::RenderTarget . . . 311
      - LLGL::RenderContext . . . 257
    - LLGL::Resource . . . 317
      - LLGL::Buffer . . . 123
      - LLGL::Sampler . . . 321
      - LLGL::Texture . . . 360
    - LLGL::ResourceHeap . . . 318
    - LLGL::Shader . . . 327
    - LLGL::ShaderProgram . . . 335
  - LLGL::Surface . . . 357
    - LLGL::Canvas . . . 128
    - LLGL::Window . . . 392
  - LLGL::Timer . . . 375
  - LLGL::Window::EventListener . . . 194
    - LLGL::Input . . . 228

LLGL::Offset2D . . . 245
LLGL::Offset3D . . . 246
LLGL::OpenGLDependentStateDescriptor . . . 247
LLGL::PipelineLayoutDescriptor . . . 249
LLGL::QueryHeapDescriptor . . . 251
LLGL::QueryPipelineStatistics . . . 252
LLGL::RasterizerDescriptor . . . 254
LLGL::RenderContextDescriptor . . . 264
LLGL::RendererConfigurationOpenGL . . . 265
LLGL::RendererConfigurationVulkan . . . 266
LLGL::RendererID . . . 268
LLGL::RendererInfo . . . 270
LLGL::RenderingCapabilities . . . 271
LLGL::RenderingDebugger . . . 273
LLGL::RenderingFeatures . . . 275
LLGL::RenderingLimits . . . 282
LLGL::RenderingProfiler . . . 286
LLGL::RenderPassDescriptor . . . 288
LLGL::RenderSystemConfiguration . . . 307
LLGL::RenderSystemDescriptor . . . 308

LLGL::RenderTargetDescriptor . . . 314
LLGL::ResourceHeapDescriptor . . . 318
LLGL::ResourceViewDescriptor . . . 320
LLGL::SamplerDescriptor . . . 322
LLGL::Scissor . . . 325
LLGL::ShaderCompileFlags . . . 329
LLGL::ShaderDescriptor . . . 330
LLGL::ShaderMacro . . . 334
LLGL::ShaderProgramDescriptor . . . 340
LLGL::ShaderReflection . . . 343
LLGL::ShaderResource . . . 344
LLGL::ShaderUniform . . . 346
LLGL::SrcImageDescriptor . . . 347
LLGL::StageFlags . . . 348
LLGL::StencilDescriptor . . . 349
LLGL::StencilFaceDescriptor . . . 350
LLGL::BufferDescriptor::StorageBuffer . . . 352
LLGL::ShaderDescriptor::StreamOutput . . . 353
LLGL::StreamOutputAttribute . . . 354
LLGL::StreamOutputFormat . . . 356
LLGL::TextureDescriptor . . . 362
LLGL::TextureLocation . . . 366
LLGL::TextureRegion . . . 368
LLGL::TextureSubresource . . . 369
LLGL::TextureSwizzleRGBA . . . 372
LLGL::TextureViewDescriptor . . . 373
LLGL::UninitializeTag . . . 377
LLGL::VertexAttribute . . . 378
LLGL::BufferDescriptor::VertexBuffer . . . 381
LLGL::VertexFormat . . . 382
LLGL::VideoAdapterDescriptor . . . 384
LLGL::VideoModeDescriptor . . . 385
LLGL::VideoOutputDescriptor . . . 387
LLGL::Viewport . . . 388
LLGL::VsyncDescriptor . . . 391
LLGL::WindowBehavior . . . 399
LLGL::WindowDescriptor . . . 400

# Chapter 6

# Class Index

## 6.1 Class List

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


LLGL::ApplicationDescriptor
Application descriptor structure . . . 107
LLGL::AttachmentClear
Attachment clear command structure . . . 108
LLGL::AttachmentDescriptor
Render target attachment descriptor structure . . . 110
LLGL::AttachmentFormatDescriptor
Render target attachment descriptor structure . . . 112
LLGL::BindFlags
Flags for Buffer and Texture resources that describe for which purposes they will be used . . . 114
LLGL::BindingDescriptor
Layout structure for a single binding point of the pipeline layout descriptor . . . 116
LLGL::BlendDescriptor
Blending state descriptor structure . . . 119
LLGL::BlendTargetDescriptor
Blend target state descriptor structure . . . 121
LLGL::Buffer
Hardware buffer interface . . . 123
LLGL::BufferArray
Hardware buffer container interface . . . 124
LLGL::BufferDescriptor
Hardware buffer descriptor structure . . . 125
LLGL::Canvas
Canvas interface for mobile platforms . . . 128
LLGL::CanvasDescriptor
Canvas descriptor structure . . . 131
LLGL::ClearFlags
Command buffer clear flags . . . 132
LLGL::ClearValue
Clear value structure for color, depth, and stencil clear operations . . . 132
LLGL::Color< T, N >
Base color class with N components . . . 133
LLGL::Color< T, 3u >
RGB color class with components: r, g, and b . . . 137
LLGL::Color< T, 4u >
RGBA color class with components: r, g, b, and a . . . 142

LLGL::CommandBuffer
Command buffer interface used for storing and encoding GPU commands . . . . . 147
LLGL::CommandBufferDescriptor
Command buffer descriptor structure . . . . . 173
LLGL::CommandBufferFlags
Command buffer creation flags . . . . . 174
LLGL::CommandQueue
Command queue interface . . . . . 175
LLGL::ComputePipeline
Compute pipeline interface . . . . . 178
LLGL::ComputePipelineDescriptor
Compute pipeline descriptor structure . . . . . 178
LLGL::CPUAccessFlags
CPU read/write access flag enumeration . . . . . 179
LLGL::DepthBiasDescriptor
Depth bias descriptor structure to control fragment depth values . . . . . 180
LLGL::DepthDescriptor
Depth state descriptor structure . . . . . 181
LLGL::DispatchIndirectArguments
Format structure for the arguments of an indirect compute command . . . . . 182
LLGL::Display
Display interface to query the attributes of all connected displays/monitors . . . . . 183
LLGL::DisplayModeDescriptor
Display mode descriptor structure . . . . . 187
LLGL::DrawIndexedIndirectArguments
Format structure for the arguments of an indirect indexed draw command . . . . . 188
LLGL::DrawIndirectArguments
Format structure for the arguments of an indirect draw command . . . . . 190
LLGL::DrawPatchIndirectArgument
Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer . . . . . 191
LLGL::DstImageDescriptor
Descriptor structure for an image that is used as destination for writing the image data . . . . . 193
LLGL::Window::EventListener
Interface for all window event listeners . . . . . 194
LLGL::Canvas::EventListener
Interface for all canvas event listeners . . . . . 198
LLGL::Extent2D
2-Dimensional extent structure . . . . . 199
LLGL::Extent3D
3-Dimensional extent structure . . . . . 200
LLGL::Fence
Fence interface for CPU/GPU synchronization . . . . . 202
LLGL::FormatDescriptor
Vertex and texture format descriptor structure . . . . . 202
LLGL::FragmentAttribute
Fragment output attribute structure . . . . . 205
LLGL::FrameProfile
Profile of a rendered frame . . . . . 206
LLGL::GraphicsPipeline
Graphics pipeline interface . . . . . 215
LLGL::GraphicsPipelineDescriptor
Graphics pipeline descriptor structure . . . . . 216
LLGL::Image
Utility class to manage the storage and attributes of an image . . . . . 218
LLGL::BufferDescriptor::IndexBuffer
Index buffer specific descriptor structure . . . . . 228

LLGL::Input
Default window event listener to receive user input . . . 228
LLGL::Interface
Base class for all interfaces in LLGL . . . 233
LLGL::InterfaceID
Enumeration of all interfaces in LLGL . . . 234
LLGL::RenderingDebugger::Message
Rendering debugger message class . . . 236
LLGL::MiscFlags
Miscellaneous resource flags . . . 239
LLGL::MultiSamplingDescriptor
Multi-sampling descriptor structure . . . 240
LLGL::NativeContextHandle
IOS native context handle structure . . . 241
LLGL::NativeHandle
IOS native handle structure . . . 243
LLGL::NonCopyable
Base class for all classes with virtual function tables in LLGL . . . 244
LLGL::Offset2D
2-Dimensional offset structure . . . 245
LLGL::Offset3D
3-Dimensional offset structure . . . 246
LLGL::OpenGLDependentStateDescriptor
Graphics API dependent state descriptor for the OpenGL renderer . . . 247
LLGL::PipelineLayout
Pipeline layout interface . . . 248
LLGL::PipelineLayoutDescriptor
Pipeline layout descritpor structure . . . 249
LLGL::QueryHeap
Query heap interface that holds a certain number of queries that are all of the same type . . . 250
LLGL::QueryHeapDescriptor
Query heap descriptor structure . . . 251
LLGL::QueryPipelineStatistics
Query data structure for pipeline statistics . . . 252
LLGL::RasterizerDescriptor
Rasterizer state descriptor structure . . . 254
LLGL::RenderContext
Render context interface . . . 257
LLGL::RenderContextDescriptor
Render context descriptor structure . . . 264
LLGL::RendererConfigurationOpenGL
OpenGL profile descriptor structure . . . 265
LLGL::RendererConfigurationVulkan
Structure for a Vulkan renderer specific configuration . . . 266
LLGL::RendererID
Renderer identification number enumeration . . . 268
LLGL::RendererInfo
Renderer basic information structure . . . 270
LLGL::RenderingCapabilities
Structure with all attributes describing the rendering capabilities of the render system . . . 271
LLGL::RenderingDebugger
Rendering debugger interface . . . 273
LLGL::RenderingFeatures
Contains the attributes for all supported rendering features . . . 275
LLGL::RenderingLimits
Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc 282
LLGL::RenderingProfiler
Rendering profiler model class . . . 286

LLGL::RenderPass
Render pass interface . . . . . 287
LLGL::RenderPassDescriptor
Render pass descriptor structure . . . . . 288
LLGL::RenderSystem
Render system interface . . . . . 290
LLGL::RenderSystemChild
Base class for all interfaces whoes instances are owned by the RenderSystem . . . . . 306
LLGL::RenderSystemConfiguration
Render system configuration structure . . . . . 307
LLGL::RenderSystemDescriptor
Render system descriptor structure . . . . . 308
LLGL::RenderTarget
Render target interface . . . . . 311
LLGL::RenderTargetDescriptor
Render target descriptor structure . . . . . 314
LLGL::Resource
Base class for all hardware resource interfaces . . . . . 317
LLGL::ResourceHeap
Resource heap interface . . . . . 318
LLGL::ResourceHeapDescriptor
Resource heap descriptor structure . . . . . 318
LLGL::ResourceViewDescriptor
Resource view descriptor structure . . . . . 320
LLGL::Sampler
Sampler interface . . . . . 321
LLGL::SamplerDescriptor
Texture sampler descriptor structure . . . . . 322
LLGL::Scissor
Scissor dimensions . . . . . 325
LLGL::Shader
Shader interface . . . . . 327
LLGL::ShaderCompileFlags
Shader compilation flags enumeration . . . . . 329
LLGL::ShaderDescriptor
Shader source and binary code descriptor structure . . . . . 330
LLGL::ShaderMacro
Shader macro structure with name and optional body definition . . . . . 334
LLGL::ShaderProgram
Shader program interface . . . . . 335
LLGL::ShaderProgramDescriptor
Descriptor structure for shader programs . . . . . 340
LLGL::ShaderReflection
Shader reflection structure . . . . . 343
LLGL::ShaderResource
Shader reflection resource structure . . . . . 344
LLGL::ShaderUniform
Shader reflection uniform structure . . . . . 346
LLGL::SrcImageDescriptor
Descriptor structure for an image that is used as source for reading the image data . . . . . 347
LLGL::StageFlags
Shader stage flags enumeration . . . . . 348
LLGL::StencilDescriptor
Stencil state descriptor structure . . . . . 349
LLGL::StencilFaceDescriptor
Stencil face descriptor structure . . . . . 350
LLGL::BufferDescriptor::StorageBuffer
Storage buffer specific descriptor structure . . . . . 352

LLGL::ShaderDescriptor::StreamOutput
Additional descriptor for stream outputs . . . 353
LLGL::StreamOutputAttribute
Stream-output attribute structure . . . 354
LLGL::StreamOutputFormat
Stream-output format descriptor structure . . . 356
LLGL::Surface
The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms) . . . 357
LLGL::Texture
Texture interface . . . 360
LLGL::TextureDescriptor
Texture descriptor structure . . . 362
LLGL::TextureLocation
Texture location structure: MIP-map level and offset . . . 366
LLGL::TextureRegion
Texture region structure: MIP-map level, offset, and extent . . . 368
LLGL::TextureSubresource
Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource . . . 369
LLGL::TextureSwizzleRGBA
Texture component swizzle structure for red, green, blue, and alpha components . . . 372
LLGL::TextureViewDescriptor
Texture view descriptor structure . . . 373
LLGL::Timer
Interface for a Timer class . . . 375
LLGL::UninitializeTag
Common uninitialize tag . . . 377
LLGL::VertexAttribute
Vertex input attribute structure . . . 378
LLGL::BufferDescriptor::VertexBuffer
Vertex buffer specific descriptor structure . . . 381
LLGL::VertexFormat
Vertex format structure . . . 382
LLGL::VideoAdapterDescriptor
Video adapter descriptor structure . . . 384
LLGL::VideoModeDescriptor
Video mode descriptor structure . . . 385
LLGL::VideoOutputDescriptor
Video output structure . . . 387
LLGL::Viewport
Viewport dimensions . . . 388
LLGL::VsyncDescriptor
Vertical-synchronization (Vsync) descriptor structure . . . 391
LLGL::Window
Window interface for desktop platforms . . . 392
LLGL::WindowBehavior
Window behavior structure . . . 399
LLGL::WindowDescriptor
Window descriptor structure . . . 400

# Chapter 7

# File Index

## 7.1 File List

Here is a list of all files with brief descriptions:


Buffer.h . . . 403
BufferArray.h . . . 403
BufferFlags.h . . . 404
Canvas.h . . . 405
CanvasFlags.h . . . 405
Color.h . . . 405
ColorRGB.h . . . 407
ColorRGBA.h . . . 407
CommandBuffer.h . . . 408
CommandBufferFlags.h . . . 408
CommandQueue.h . . . 409
ComputePipeline.h . . . 409
ComputePipelineFlags.h . . . 410
Constants.h . . . 410
Display.h . . . 410
DisplayFlags.h . . . 410
Export.h . . . 411
Fence.h . . . 411
Format.h . . . 412
ForwardDecls.h . . . 413
FragmentAttribute.h . . . 414
GraphicsPipeline.h . . . 414
GraphicsPipelineFlags.h . . . 414
Image.h . . . 417
ImageFlags.h . . . 417
IndirectArguments.h . . . 418
Input.h . . . 419
Interface.h . . . 419
InterfaceID.h . . . 420
IOSNativeHandle.h . . . 420
Key.h . . . 421
LinuxNativeHandle.h . . . 422
LLGL.h . . . 422
Log.h . . . 422
MacOSNativeHandle.h . . . 423

NativeHandle.h . . . 423
NonCopyable.h . . . 424
PipelineLayout.h . . . 424
PipelineLayoutFlags.h . . . 424
Platform.h . . . 425
QueryHeap.h . . . 425
QueryHeapFlags.h . . . 425
RenderContext.h . . . 426
RenderContextFlags.h . . . 427
RendererConfiguration.h . . . 427
RenderingDebugger.h . . . 428
RenderingProfiler.h . . . 428
RenderPass.h . . . 429
RenderPassFlags.h . . . 429
RenderSystem.h . . . 430
RenderSystemChild.h . . . 431
RenderSystemFlags.h . . . 431
RenderTarget.h . . . 433
RenderTargetFlags.h . . . 433
Resource.h . . . 433
ResourceFlags.h . . . 434
ResourceHeap.h . . . 434
ResourceHeapFlags.h . . . 435
Sampler.h . . . 435
SamplerFlags.h . . . 435
Shader.h . . . 436
ShaderFlags.h . . . 436
ShaderProgram.h . . . 437
ShaderProgramFlags.h . . . 438
StreamOutputAttribute.h . . . 439
StreamOutputFormat.h . . . 440
Strings.h . . . 440
Surface.h . . . 441
SystemValue.h . . . 441
Tags.h . . . 441
Texture.h . . . 442
TextureFlags.h . . . 442
Timer.h . . . 443
TypeInfo.h . . . 444
Types.h . . . 444
Utility.h . . . 445
Version.h . . . 447
VertexAttribute.h . . . 447
VertexFormat.h . . . 448
VideoAdapter.h . . . 448
Win32NativeHandle.h . . . 449
Window.h . . . 449
WindowFlags.h . . . 450

# Chapter 8

# Module Documentation

## 8.1 Global functions for Strict-Weak-Order (SWO) comparisons.

### Functions

- LLGL_EXPORT bool LLGL::CompareSWO (const DisplayModeDescriptor &lhs, const DisplayMode↩ Descriptor &rhs)

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

### 8.1.1 Detailed Description

### 8.1.2 Function Documentation

#### 8.1.2.1 LLGL_EXPORT bool LLGL::CompareSWO ( const DisplayModeDescriptor & *lhs,* const DisplayModeDescriptor & *rhs* )

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

## 8.2 Global operators for basic data structures.

### Functions

- LLGL_EXPORT Extent2D LLGL::operator+ (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent2D LLGL::operator- (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D LLGL::operator+ (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D LLGL::operator- (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Offset2D LLGL::operator+ (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset2D LLGL::operator- (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D LLGL::operator+ (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D LLGL::operator- (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool LLGL::operator!= (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool LLGL::operator!= (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool LLGL::operator!= (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*
- bool LLGL::operator== (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool LLGL::operator!= (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*

### 8.2.1 Detailed Description

### 8.2.2 Function Documentation

#### 8.2.2.1 bool LLGL::operator!= ( const Offset2D & *lhs*, const Offset2D & *rhs* ) [inline]

Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.

#### 8.2.2.2 bool LLGL::operator!= ( const Offset3D & *lhs*, const Offset3D & *rhs* ) [inline]

Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.

#### 8.2.2.3 bool LLGL::operator!= ( const Extent2D & *lhs*, const Extent2D & *rhs* ) [inline]

Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.

#### 8.2.2.4 bool LLGL::operator!= ( const Extent3D & *lhs*, const Extent3D & *rhs* ) [inline]

Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.

#### 8.2.2.5 LLGL_EXPORT Extent2D LLGL::operator+ ( const Extent2D & *lhs*, const Extent2D & *rhs* )

Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.

**Remarks**

If the operation results in a numerical overflow, the respective components will be clamped to its maximum, i.e. `std::numeric_limits<std::uint32_t>::max()`.

#### 8.2.2.6 LLGL_EXPORT Extent3D LLGL::operator+ ( const Extent3D & *lhs*, const Extent3D & *rhs* )

Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.

**Remarks**

If the operation results in a numerical overflow, the respective components will be clamped to its maximum, i.e. `std::numeric_limits<std::uint32_t>::max()`.

#### 8.2.2.7 LLGL_EXPORT Offset2D LLGL::operator+ ( const Offset2D & *lhs*, const Offset2D & *rhs* )

Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.

**Remarks**

If the operation results in a numerical overflow or underflow, the respective components will be clamped into the range `[std::numeric_limits<std::int32_t>::min(), std::numeric_limits<std::int32_t>::max()]`.

#### 8.2.2.8 LLGL_EXPORT Offset3D LLGL::operator+ ( const Offset3D & *lhs*, const Offset3D & *rhs* )

Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.

**Remarks**

If the operation results in a numerical overflow or underflow, the respective components will be clamped into the range `[std::numeric_limits<std::int32_t>::min(), std::numeric_limits<std::int32_t>::max()]`.

#### 8.2.2.9 LLGL_EXPORT Extent2D LLGL::operator- ( const Extent2D & *lhs*, const Extent2D & *rhs* )

Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.

**Remarks**

If the operation results in a numerical underflow, the respective components will be clamped to its minimum, i.e. 0.

#### 8.2.2.10 LLGL_EXPORT Extent3D LLGL::operator- ( const Extent3D & *lhs*, const Extent3D & *rhs* )

Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.

**Remarks**

If the operation results in a numerical underflow, the respective components will be clamped to its minimum, i.e. 0.

#### 8.2.2.11 LLGL_EXPORT Offset2D LLGL::operator- ( const Offset2D & *lhs*, const Offset2D & *rhs* )

Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.

**Remarks**

If the operation results in a numerical overflow or underflow, the respective components will be clamped into the range `[std::numeric_limits<std::int32_t>::min(), std::numeric_limits<std::int32_t>::max()]`.

#### 8.2.2.12 LLGL_EXPORT Offset3D LLGL::operator- ( const Offset3D & *lhs*, const Offset3D & *rhs* )

Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.

**Remarks**

If the operation results in a numerical overflow or underflow, the respective components will be clamped into the range `[std::numeric_limits<std::int32_t>::min(), std::numeric_limits<std::int32_t>::max()]`.

#### 8.2.2.13 bool LLGL::operator== ( const Offset2D & *lhs*, const Offset2D & *rhs* ) [inline]

Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.

#### 8.2.2.14 bool LLGL::operator== ( const Offset3D & *lhs*, const Offset3D & *rhs* ) [inline]

Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.

#### 8.2.2.15 bool LLGL::operator== ( const Extent2D & *lhs*, const Extent2D & *rhs* ) [inline]

Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.

#### 8.2.2.16 bool LLGL::operator== ( const Extent3D & *lhs*, const Extent3D & *rhs* ) [inline]

Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.

## 8.3 Global type aliases to callback interfaces.

### Typedefs

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

  *Debug callback function interface.*
- using LLGL::ValidateRenderingCapsFunc = std::function< bool(const std::string &info, const std::string &attrib)>

  *Callback interface for the ValidateRenderingCaps function.*

### 8.3.1 Detailed Description

### 8.3.2 Typedef Documentation

#### 8.3.2.1 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.

**Note**

Only supported with: OpenGL, Vulkan, Metal.
For Direct3D, debug output will be reported in the output log of the IDE.

**See also**

RenderSystemDescriptor::debugCallback

#### 8.3.2.2 using LLGL::ValidateRenderingCapsFunc = typedef std::function<bool(const std::string& info, const std::string& attrib)>

Callback interface for the ValidateRenderingCaps function.

**Parameters**

| | | |
|---|---|---|
| in | *info* | Specifies a description why an attribute did not fulfill the requirement. |
| in | *attrib* | Name of the attribute which did not fulfill the requirement. |

**Returns**

True to continue the validation process, or false to break the validation process.

**See also**

ValidateRenderingCaps

## 8.4 Buffer utility functions to determine buffer types.

### Functions

- LLGL_EXPORT bool LLGL::IsRWBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a read/write (RW) buffer.*
- LLGL_EXPORT bool LLGL::IsTypedBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a simply typed buffer.*
- LLGL_EXPORT bool LLGL::IsStructuredBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a structured buffer.*
- LLGL_EXPORT bool LLGL::IsByteAddressBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a byte addresse buffer.*

### 8.4.1 Detailed Description

### 8.4.2 Function Documentation

#### 8.4.2.1 LLGL_EXPORT bool LLGL::IsByteAddressBuffer ( const StorageBufferType *type* )

Returns true if the storage buffer type denotes a byte addresse buffer.

**Returns**

True if `type` either StorageBufferType::ByteAddressBuffer or StorageBufferType::RWByteAddressBuffer.

#### 8.4.2.2 LLGL_EXPORT bool LLGL::IsRWBuffer ( const StorageBufferType *type* )

Returns true if the storage buffer type denotes a read/write (RW) buffer.

**Returns**

True if `type` either StorageBufferType::RWBuffer, StorageBufferType::RWStructuredBuffer, StorageBufferType::RWByteAddressBuffer, StorageBufferType::AppendStructuredBuffer, or StorageBufferType::ConsumeStructuredBuffer.

#### 8.4.2.3 LLGL_EXPORT bool LLGL::IsStructuredBuffer ( const StorageBufferType *type* )

Returns true if the storage buffer type denotes a structured buffer.

**Returns**

True if `type` either StorageBufferType::StructuredBuffer, StorageBufferType::RWStructuredBuffer, StorageBufferType::AppendStructuredBuffer, or StorageBufferType::ConsumeStructuredBuffer.

#### 8.4.2.4 LLGL_EXPORT bool LLGL::IsTypedBuffer ( const StorageBufferType *type* )

Returns true if the storage buffer type denotes a simply typed buffer.

**Returns**

True if `type` either StorageBufferType::Buffer or StorageBufferType::RWBuffer.

## 8.5 Hardware format utility functions.

### Functions

- LLGL_EXPORT const FormatDescriptor & LLGL::GetFormatDesc (const Format format)

  *Returns the descriptor for the specified hardware format or a zero-initialized descriptor in case of an invalid argument.*
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const Format format)

  *Returns true if the specified hardware format is a compressed format, i.e. either Format::BC1RGB, Format::BC1RGBA, Format::BC2RGBA, or Format::BC3RGBA.*
- LLGL_EXPORT bool LLGL::IsDepthStencilFormat (const Format format)

  *Returns true if the specified hardware format is a depth or depth-stencil format, i.e. Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsDepthFormat (const Format format)

  *Returns true if the specified hardware format is a depth format, i.e. Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsStencilFormat (const Format format)

  *Returns true if the specified hardware format is a stencil format, i.e. Format::D24UNormS8UInt or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsNormalizedFormat (const Format format)

  *Returns true if the specified hardware format is a normalized format (like Format::RGBA8UNorm, Format::R8SNorm etc.).*
- LLGL_EXPORT bool LLGL::IsIntegralFormat (const Format format)

  *Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::R8SInt etc.).*
- LLGL_EXPORT bool LLGL::IsFloatFormat (const Format format)

  *Returns true if the specified hardware format is a floating-point format (like Format::RGBA32Float, Format::R32Float etc.).*

### 8.5.1 Detailed Description

### 8.5.2 Function Documentation

#### 8.5.2.1 LLGL_EXPORT const FormatDescriptor& LLGL::GetFormatDesc ( const Format *format* )

Returns the descriptor for the specified hardware format or a zero-initialized descriptor in case of an invalid argument.

**See also**

FormatDescriptor
Format

#### 8.5.2.2 LLGL_EXPORT bool LLGL::IsCompressedFormat ( const Format *format* )

Returns true if the specified hardware format is a compressed format, i.e. either Format::BC1RGB, Format::BC1RGBA, Format::BC2RGBA, or Format::BC3RGBA.

**See also**

Format

#### 8.5.2.3 LLGL_EXPORT bool LLGL::IsDepthFormat ( const Format *format* )

Returns true if the specified hardware format is a depth format, i.e. Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.

**See also**

Format

#### 8.5.2.4 LLGL_EXPORT bool LLGL::IsDepthStencilFormat ( const Format *format* )

Returns true if the specified hardware format is a depth or depth-stencil format, i.e. Format::D16UNorm, Format↩ ::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.

**See also**

Format

#### 8.5.2.5 LLGL_EXPORT bool LLGL::IsFloatFormat ( const Format *format* )

Returns true if the specified hardware format is a floating-point format (like Format::RGBA32Float, Format::R32Float etc.).

**Remarks**

This does not include depth-stencil formats or compressed formats.

**See also**

IsDepthStencilFormat
IsCompressedFormat
Format

#### 8.5.2.6 LLGL_EXPORT bool LLGL::IsIntegralFormat ( const Format *format* )

Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::R8SInt etc.).

**Remarks**

This also includes all normalized formats.

**See also**

IsNormalizedFormat
Format

#### 8.5.2.7 LLGL_EXPORT bool LLGL::IsNormalizedFormat ( const Format *format* )

Returns true if the specified hardware format is a normalized format (like Format::RGBA8UNorm, Format::R8SNorm etc.).

**Remarks**

This does not include depth-stencil formats or compressed formats.

**See also**

IsDepthStencilFormat
IsCompressedFormat
Format

#### 8.5.2.8 LLGL_EXPORT bool LLGL::IsStencilFormat ( const Format *format* )

Returns true if the specified hardware format is a stencil format, i.e. Format::D24UNormS8UInt or Format::D32↩FloatS8X24UInt.

**See also**

Format

## 8.6 Data type utility functions.

### Functions

- LLGL_EXPORT std::uint32_t LLGL::DataTypeSize (const DataType dataType)

  *Returns the size (in bytes) of the specified data type.*
- LLGL_EXPORT bool LLGL::IsIntDataType (const DataType dataType)

  *Determines if the argument refers to a signed integer data type.*
- LLGL_EXPORT bool LLGL::IsUIntDataType (const DataType dataType)

  *Determines if the argument refers to an unsigned integer data type.*
- LLGL_EXPORT bool LLGL::IsFloatDataType (const DataType dataType)

  *Determines if the argument refers to a floating-pointer data type.*

### 8.6.1 Detailed Description

### 8.6.2 Function Documentation

#### 8.6.2.1 LLGL_EXPORT std::uint32_t LLGL::DataTypeSize ( const DataType *dataType* )

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

#### 8.6.2.2 LLGL_EXPORT bool LLGL::IsFloatDataType ( const DataType *dataType* )

Determines if the argument refers to a floating-pointer data type.

**Returns**

True if the specified data type equals one of the following enumeration entries: DataType::Float16, DataType::Float32, DataType::Float64.

#### 8.6.2.3 LLGL_EXPORT bool LLGL::IsIntDataType ( const DataType *dataType* )

Determines if the argument refers to a signed integer data type.

**Returns**

True if the specified data type equals one of the following enumeration entries: DataType::Int8, DataType::Int16, DataType::Int32.

#### 8.6.2.4 LLGL_EXPORT bool LLGL::IsUIntDataType ( const DataType *dataType* )

Determines if the argument refers to an unsigned integer data type.

**Returns**

True if the specified data type equals one of the following enumeration entries: DataType::UInt8, DataType::UInt16, DataType::UInt32.

## 8.7 Image utility functions to classify and convert image data.

### Functions

- LLGL_EXPORT std::uint32_t LLGL::ImageFormatSize (const ImageFormat imageFormat)

  *Returns the size (in number of components) of the specified image format.*
- LLGL_EXPORT std::uint32_t LLGL::ImageDataSize (const ImageFormat imageFormat, const DataType dataType, std::uint32_t numPixels)

  *Returns the required data size (in bytes) of an image with the specified format, data type, and number of pixels.*
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const ImageFormat imageFormat)

  *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 imageFormat)

  *Returns true if the specified color format is a depth-stencil format, i.e. either ImageFormat::Depth or ImageFormat↩ ::DepthStencil.*
- LLGL_EXPORT bool LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const Dst↩ ImageDescriptor &dstImageDesc, std::size_t threadCount=0)

  *Converts the image format and data type of the source image (only uncompressed color formats).*
- LLGL_EXPORT ByteBuffer LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, Image↩ Format dstFormat, DataType dstDataType, std::size_t threadCount=0)

  *Converst the image format and data type of the source image (only uncompressed color formats) and returns the new generated image buffer.*
- LLGL_EXPORT ByteBuffer LLGL::GenerateImageBuffer (ImageFormat format, DataType dataType, std↩ ::size_t imageSize, const ColorRGBAd &fillColor)

  *Generates an image buffer with the specified fill data for each pixel.*
- LLGL_EXPORT ByteBuffer LLGL::GenerateEmptyByteBuffer (std::size_t bufferSize, bool initialize=true)

  *Generates a new byte buffer with zeros in each byte.*

### 8.7.1 Detailed Description

### 8.7.2 Function Documentation

#### 8.7.2.1 LLGL_EXPORT bool LLGL::ConvertImageBuffer ( const SrcImageDescriptor & *srcImageDesc,* const DstImageDescriptor & *dstImageDesc,* std::size_t *threadCount* = 0 )

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

**Parameters**

| | | |
|---|---|---|
| in | *srcImageDesc* | Specifies the source image descriptor. |
| out | *dstImageDesc* | Specifies the destination image descriptor. |
| 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 'Constants::maxThreadCount', the maximal count of threads the system supports will be used (e.g. 4 on a quad-core processor). By default 0. |

**Returns**

True if any conversion was necessary. Otherwise, no conversion was necessary and the destination buffer is not modified!

**Note**

Compressed images and depth-stencil images cannot be converted.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If a compressed image format is specified either as source or destination. |
| *std::invalid_argument* | If a depth-stencil format is specified either as source or destination. |
| *std::invalid_argument* | If the source buffer size is not a multiple of the source data type size times the image format size. |
| *std::invalid_argument* | If the source buffer is a null pointer. |
| *std::invalid_argument* | If the destination buffer size does not match the required output buffer size. |
| *std::invalid_argument* | If the destination buffer is a null pointer. |

**See also**

Constants::maxThreadCount
DataTypeSize
ImageFormatSize

#### 8.7.2.2 LLGL_EXPORT ByteBuffer LLGL::ConvertImageBuffer ( const SrcImageDescriptor & *srcImageDesc*, ImageFormat *dstFormat*, DataType *dstDataType*, std::size_t *threadCount* = `0` )

Converst the image format and data type of the source image (only uncompressed color formats) and returns the new generated image buffer.

**Parameters**

| | | |
|---|---|---|
| `in` | *srcImageDesc* | Specifies the source image descriptor. |
| `in` | *dstFormat* | Specifies the destination image format. |
| `in` | *dstDataType* | Specifies the destination image 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 'Constants::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.).

**Note**

Compressed images and depth-stencil images cannot be converted.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If a compressed image format is specified either as source or destination. |
| *std::invalid_argument* | If a depth-stencil format is specified either as source or destination. |
| *std::invalid_argument* | If the source buffer size is not a multiple of the source data type size times the image format size. |
| *std::invalid_argument* | If the source buffer is a null pointer. |

**See also**

Constants::maxThreadCount
ByteBuffer
DataTypeSize
ImageFormatSize

#### 8.7.2.3 LLGL_EXPORT ByteBuffer LLGL::GenerateEmptyByteBuffer ( std::size_t *bufferSize*, bool *initialize* = `true` )

Generates a new byte buffer with zeros in each byte.

**Parameters**

| | | |
|---|---|---|
| in | *bufferSize* | Specifies the size (in bytes) of the buffer. |
| in | *initialize* | Specifies whether to initialize the byte buffer with zeros. By default true. |

**Returns**

The new allocated and initialized byte buffer.

**Remarks**

Use GenerateImageBuffer to generate an image buffer with a fill color.

**See also**

GenerateImageBuffer

#### 8.7.2.4 LLGL_EXPORT ByteBuffer LLGL::GenerateImageBuffer ( ImageFormat *format*, DataType *dataType*, std::size_t *imageSize*, const ColorRGBAd & *fillColor* )

Generates an image buffer with the specified fill data for each pixel.

**Parameters**

| | | |
|---|---|---|
| in | *format* | Specifies the image format of each pixel in the output image. |
| in | *dataType* | Specifies the data type of each component of each pixel in the output image. |
| in | *imageSize* | Specifies the 1-Dimensional size (in pixels) of the output image. For a 2D image, this can be width times height for instance. |
| in | *fillColor* | Specifies the color to fill the image for each pixel. |

**Returns**

The new allocated and initialized byte buffer.

**Remarks**

This can be used to generate a single-colored n-Dimensional image. Usage example for a 2D image:

```
// Generate 2D image of size 512 x 512 with a half-transparent yellow color
auto imageBuffer = LLGL::GenerateImageBuffer(
    LLGL::ImageFormat::RGBA,
    LLGL::DataType::UInt8,
    512 * 512,
    LLGL::ColorRGBAd { 1.0, 1.0, 0.0, 0.5 }
);
```

#### 8.7.2.5 LLGL_EXPORT std::uint32_t LLGL::ImageDataSize ( const ImageFormat *imageFormat*, const DataType *dataType*, std::uint32_t *numPixels* )

Returns the required data size (in bytes) of an image with the specified format, data type, and number of pixels.

**Parameters**

| | | |
|---|---|---|
| in | *imageFormat* | Specifies the image format. |
| in | *dataType* | Specifies the data type of each pixel component. |
| in | *numPixels* | Specifies the number of picture elements (pixels). |

**Remarks**

The counterpart for texture buffers is the function TextureBufferSize.

**See also**

TextureBufferSize

#### 8.7.2.6 LLGL_EXPORT std::uint32_t 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.

**Note**

Compressed formats are not supported.

**See also**

IsCompressedFormat(const ImageFormat)
ImageFormat

#### 8.7.2.7 LLGL_EXPORT bool LLGL::IsCompressedFormat ( const ImageFormat *imageFormat* )

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

**See also**

ImageFormat

#### 8.7.2.8 LLGL_EXPORT bool LLGL::IsDepthStencilFormat ( const ImageFormat *imageFormat* )

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

**See also**

ImageFormat

## 8.8 Buffer argument structures for indirect draw and compute commands.

### Classes

- struct LLGL::DrawIndirectArguments

  *Format structure for the arguments of an indirect draw command.*
- struct LLGL::DrawIndexedIndirectArguments

  *Format structure for the arguments of an indirect indexed draw command.*
- struct LLGL::DrawPatchIndirectArgument

  *Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer.*
- struct LLGL::DispatchIndirectArguments

  *Format structure for the arguments of an indirect compute command.*

### 8.8.1 Detailed Description

## 8.9 Global type-to-string conversion functions.

### Functions

- LLGL_EXPORT const char * LLGL::ToString (const ShaderType t)

  *Returns a string representation for the spcified ShaderType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const ErrorType t)

  *Returns a string representation for the specified ErrorType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const WarningType t)

  *Returns a string representation for the specified WarningType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const ShadingLanguage t)

  *Returns a string representation for the specified ShadingLanguage value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const Format t)

  *Returns a string representation for the specified Format value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const TextureType t)

  *Returns a string representation for the specified TextureType value, or null if the input type is invalid.*

### 8.9.1 Detailed Description

### 8.9.2 Function Documentation

#### 8.9.2.1 LLGL_EXPORT const char* LLGL::ToString ( const ShaderType *t* )

Returns a string representation for the spcified ShaderType value, or null if the input type is invalid.

**Remarks**

Return value examples are `"vertex"`, `"tessellation control"`.

#### 8.9.2.2 LLGL_EXPORT const char* LLGL::ToString ( const ErrorType *t* )

Returns a string representation for the specified ErrorType value, or null if the input type is invalid.

**Remarks**

Return value examples are `"invalid argument"`, `"unsupported feature"`.

#### 8.9.2.3 LLGL_EXPORT const char* LLGL::ToString ( const WarningType *t* )

Returns a string representation for the specified WarningType value, or null if the input type is invalid.

**Remarks**

Return value examples are `"improper argument"`, `"pointless operation"`.

#### 8.9.2.4 LLGL_EXPORT const char* LLGL::ToString ( const ShadingLanguage *t* )

Returns a string representation for the specified ShadingLanguage value, or null if the input type is invalid.

**Remarks**

Return value examples are `"GLSL 450"`, `"HLSL 2.0c"`.

#### 8.9.2.5 LLGL_EXPORT const char* LLGL::ToString ( const Format *t* )

Returns a string representation for the specified Format value, or null if the input type is invalid.

**Remarks**

Return value examples are `"R8UNorm"`, `"RGBA16Float"`, `"D24UNormS8UInt"`, `"RGB DXT1"`.

#### 8.9.2.6 LLGL_EXPORT const char* LLGL::ToString ( const TextureType *t* )

Returns a string representation for the specified TextureType value, or null if the input type is invalid.

**Remarks**

Return value examples are `"Texture1D"`, `"Texture2DArray"`.

## 8.10 Texture utility functions to determine texture dimension and buffer sizes.

### Functions

- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1)

  *Returns the number of MIP-map levels for a texture with the specified size.*
- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (const TextureDescriptor &textureDesc)

  *Returns the number of MIP-map levels for the specified texture descriptor.*
- LLGL_EXPORT std::uint32_t LLGL::TextureBufferSize (const Format format, std::uint32_t numTexels)

  *Returns the required buffer size (in bytes) of a texture with the specified hardware format and number of texels.*
- LLGL_EXPORT std::uint32_t LLGL::TextureSize (const TextureDescriptor &textureDesc)

  *Returns the texture size (in texels) of the specified texture descriptor, or zero if the texture type is invalid.*
- LLGL_EXPORT bool LLGL::IsMipMappedTexture (const TextureDescriptor &textureDesc)

  *Returns true if the specified texture descriptor describes a texture with MIP-mapping enabled.*
- 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.*
- LLGL_EXPORT bool LLGL::IsCubeTexture (const TextureType type)

  *Returns true if the specified texture type is a cube texture.*
- LLGL_EXPORT bool LLGL::IsTextureSwizzleIdentity (const TextureSwizzleRGBA &swizzle)

  *Returns true if the specified texture swizzling is equal to the identity mapping.*

### 8.10.1 Detailed Description

### 8.10.2 Function Documentation

#### 8.10.2.1 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::Texture↩CubeArray, or TextureType::Texture2DMSArray.

#### 8.10.2.2 LLGL_EXPORT bool LLGL::IsCubeTexture ( const TextureType *type* )

Returns true if the specified texture type is a cube texture.

**Returns**

True if `type` is either TextureType::TextureCube or TextureType::TextureCubeArray.

#### 8.10.2.3 LLGL_EXPORT bool LLGL::IsMipMappedTexture ( const TextureDescriptor & *textureDesc* )

Returns true if the specified texture descriptor describes a texture with MIP-mapping enabled.

**Returns**

True if the texture type is not a multi-sampled texture and the number of MIP-map levels in the descriptor is either zero or greater than one.

**See also**

TextureDescriptor::mipLevels

#### 8.10.2.4 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.

#### 8.10.2.5 LLGL_EXPORT bool LLGL::IsTextureSwizzleIdentity ( const TextureSwizzleRGBA & *swizzle* )

Returns true if the specified texture swizzling is equal to the identity mapping.

**Returns**

True if the components are mapped as follows:

- `r` equals TextureSwizzle::Red
- `g` equals TextureSwizzle::Green
- `b` equals TextureSwizzle::Blue
- `a` equals TextureSwizzle::Alpha

**See also**

TextureSwizzle

#### 8.10.2.6 LLGL_EXPORT std::uint32_t LLGL::NumMipLevels ( std::uint32_t *width*, std::uint32_t *height* = `1`, std::uint32_t *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{ width, height, depth })).`

#### 8.10.2.7 LLGL_EXPORT std::uint32_t LLGL::NumMipLevels ( const TextureDescriptor & *textureDesc* )

Returns the number of MIP-map levels for the specified texture descriptor.

**Parameters**

| | | |
|---|---|---|
| in | *textureDesc* | Specifies the descriptor whose parameters are used to determine the number of MIP-map levels. |

**Remarks**

This function will deduce the number MIP-map levels automatically only if the member "mipLevels" is zero. Otherwise, the value of this member is returned.

**See also**

NumMipLevels(std::uint32_t, std::uint32_t, std::uint32_t)

#### 8.10.2.8 LLGL_EXPORT std::uint32_t LLGL::TextureBufferSize ( const Format *format*, std::uint32_t *numTexels* )

Returns the required buffer size (in bytes) of a texture with the specified hardware format and number of texels.

**Parameters**

| | | |
|---|---|---|
| in | *format* | Specifies the texture format. |
| in | *numTexels* | Specifies the number of texture elements (texels). For the DXT compressed texture formats, this must be a multiple of 16, since these formats compress the image in 4x4 texel blocks. |

**Returns**

The required buffer size (in bytes), or zero if the input is invalid.

**Remarks**

The counterpart for image data is the function ImageDataSize.

**See also**

ImageDataSize

#### 8.10.2.9 LLGL_EXPORT std::uint32_t LLGL::TextureSize ( const TextureDescriptor & *textureDesc* )

Returns the texture size (in texels) of the specified texture descriptor, or zero if the texture type is invalid.

**See also**

TextureDescriptor::type

## 8.11 Global utility functions, especially to fill descriptor structures.

### Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (Format format, std::uint32_t width, long bind↩Flags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture1D type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2D type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture3D type.*
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (Format format, std::uint32_t width, std::uint32↩_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCube type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (Format format, std::uint32_t width, std↩::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (Format format, std::uint32_t width, std↩::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::↩Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (Format format, std::uint32_t width, std↩::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::↩Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (Format format, std::uint32_t width, std↩::uint32_t height, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (Format format, std::uint32_t width, std↩::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::Color↩Attachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.*
- LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a vertex buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t size, const Format format, long cpu↩AccessFlags=0)

  *Returns a BufferDescriptor structure for an index buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t size, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a constant buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite)

  *Returns a BufferDescriptor structure for a storage buffer.*
- LLGL_EXPORT ShaderDescriptor LLGL::ShaderDescFromFile (const ShaderType type, const char ∗filename, const char ∗entryPoint=nullptr, const char ∗profile=nullptr, long flags=0)

  *Returns a ShaderDescriptor structure.*
- LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc (const std::initializer_list< Shader ∗ > &shaders, const std::initializer_list< VertexFormat > &vertexFormats={})

  *Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*
- LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc (const std::vector< Shader ∗ > &shaders, const std::vector< VertexFormat > &vertexFormats={})

*Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*

- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const ShaderReflection &reflection)

  *Converts the specified shader reflection descriptor into a pipeline layout descriptor.*
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const char *layoutSignature)

  *Generates a pipeline layout descriptor by parsing the specified string.*
- LLGL_EXPORT RenderPassDescriptor LLGL::RenderPassDesc (const RenderTargetDescriptor &renderTargetDesc)

  *Converts the specified render target descriptor into a render pass descriptor with default settings.*

### 8.11.1 Detailed Description

### 8.11.2 Function Documentation

#### 8.11.2.1 LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc ( uint64_t *size*, long *cpuAccessFlags* = 0 )

Returns a BufferDescriptor structure for a constant buffer.

**See also**

RenderSystem::CreateBuffer

#### 8.11.2.2 LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc ( uint64_t *size*, const Format *format*, long *cpuAccessFlags* = 0 )

Returns a BufferDescriptor structure for an index buffer.

**See also**

RenderSystem::CreateBuffer

#### 8.11.2.3 LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc ( const ShaderReflection & *reflection* )

Converts the specified shader reflection descriptor into a pipeline layout descriptor.

**Remarks**

This can be used to specifiy a pipeline layout that fits the shader layout declaration. Some rendering APIs, such as OpenGL 2.0, do not provide sufficient functionality for shader reflection. Hence, this utility function cannot be used in conjunction with all renderer versions.

#### 8.11.2.4 LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc ( const char * *layoutSignature* )

Generates a pipeline layout descriptor by parsing the specified string.

**Parameters**

| | | |
|---|---|---|
| in | *layoutSignature* | Specifies the string for the layout signature. This string must not be null. The syntax for this string is as follows:<br>• Each pair of binding point type (i.e. BindingDescriptor::type) and binding flags (i.e. BindingDescriptor::bindFlags) is specified by one of the following identifiers:<br>– `cbuffer` for constant buffers (i.e. ResourceType::Buffer and BindFlags::ConstantBuffer).<br>– `buffer` for sampled buffers (i.e. ResourceType::Buffer and BindFlags::Sampled).<br>– `rwbuffer` for read/write storage buffers (i.e. ResourceType::Buffer and BindFlags::Storage).<br>– `texture` for textures (i.e. ResourceType::Texture and BindFlags::Sampled).<br>– `rwtexture` for read/write textures (i.e. ResourceType::Texture and BindFlags::Storage).<br>– `sampler` for sampler states (i.e. ResourceType::Sampler).<br>• Optionally, the resource **name** is specified as an arbitrary identifier followed by the at-sign (e.g. `"texture(myColorMap@1)"`).<br>• The **slot** of each binding point (i.e. BindingDescriptor::slot) is specified as an integral number within brackets (e.g. `"texture(1)"`).<br>• The **array size** of each binding point (i.e. BindingDescriptor::arraySize) can be optionally specified right after the slot within squared brackets (e.g. `"texture(1[2])"`).<br>• Optionally, multiple slots can be specified within the brackets if separated by commas (e.g. `"texture(1[2],3)"`).<br>• Each binding point is separated by a comma, the last comma being optional (e.g. `"texture(1),sampler(2),"` or `"texture(1),sampler(2)"`).<br>• The stage flags (i.e. BindingDescriptor::stageFlags) can be specified after each binding point with a preceding colon using the following identifiers:<br>– `vert` for the vertex shader stage (i.e. StageFlags::VertexStage).<br>– `tesc` for the tessellation-control shader stage (i.e. StageFlags::TessControlStage).<br>– `tese` for the tessellation-evaluation shader stage (i.e. StageFlags::TessEvaluationStage).<br>– `geom` for the geometry shader stage (i.e. StageFlags::GeometryStage).<br>– `frag` for the fragment shader stage (i.e. StageFlags::FragmentStage).<br>– `comp` for the compute shader stage (i.e. StageFlags::ComputeStage).<br>• If no stage flag is specified, all shader stages will be used.<br>• Whitespaces are ignored (e.g. blanks `' '`, tabulators `'\t'`, new-line characters `'\n'` and `'\r'` etc.), see C++ STL function `std::isspace`. |

#### Remarks

Here is a usage example:

```
// Standard way of declaring a pipeline layout:
LLGL::PipelineLayoutDescriptor myLayoutDescStd;

myLayoutDescStd.bindings = {
    LLGL::BindingDescriptor{ "Scene",
      LLGL::ResourceType::Buffer,
      LLGL::BindFlags::ConstantBuffer,
      LLGL::StageFlags::FragmentStage |
      LLGL::StageFlags::VertexStage, 0u,     },
    LLGL::BindingDescriptor{
      LLGL::ResourceType::Texture, LLGL::BindFlags::Sampled,
            LLGL::StageFlags::FragmentStage,                                1u
        },
    LLGL::BindingDescriptor{ "TexArray",
      LLGL::ResourceType::Texture, LLGL::BindFlags::Sampled,
            LLGL::StageFlags::FragmentStage,                                2u, 4u
      , },
    LLGL::BindingDescriptor{
      LLGL::ResourceType::Sampler, 0,
      LLGL::StageFlags::FragmentStage,                              3u      },
};

auto myLayout = myRenderer->CreatePipelineLayout(myLayoutDescStd);
```

The same pipeline layout can be created with the following usage of this utility function:

```
// Abbreviated way of declaring a pipeline layout using the utility function:
auto myLayoutDescUtil = LLGL::PipelineLayoutDesc(
    "cbuffer(Scene@0):frag:vert,"
    "texture(1, TexArray@2[4]):frag,"
    "sampler(3):frag,"
);
auto myLayout = myRenderer->CreatePipelineLayout(myLayoutDescUtil);
```

#### Exceptions

| | |
|---|---|
| *std::invalid_argument* | If the input parameter is null of parsing the layout signature failed. |

### 8.11.2.5 LLGL_EXPORT RenderPassDescriptor LLGL::RenderPassDesc ( const RenderTargetDescriptor & *renderTargetDesc* )

Converts the specified render target descriptor into a render pass descriptor with default settings.

#### Remarks

This can be used to specify a render pass that is compatible with a render target.

### 8.11.2.6 LLGL_EXPORT ShaderDescriptor LLGL::ShaderDescFromFile ( const ShaderType *type*, const char * *filename*, const char * *entryPoint* = `nullptr`, const char * *profile* = `nullptr`, long *flags* = `0` )

Returns a ShaderDescriptor structure.

#### Remarks

The source type is determined by the filename extension using the following rules:

- `.hlsl`, `.fx`, `.glsl`, `.vert`, `.tesc`, `.tese`, `.geom`, `.frag`, `.comp`, and `.metal` result into a code file (i.e. ShaderSourceType::CodeFile)
- All other file extensions result into a binary file (i.e. ShaderSourceType::BinaryFile).

#### See also

RenderSystem::CreateShader

#### 8.11.2.7 LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc ( const std::initializer_list< Shader * > & *shaders*, const std::initializer_list< VertexFormat > & *vertexFormats* = { } )

Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.

**Parameters**

| | | |
|---|---|---|
| in | *shaders* | Specifies the list of shaders to attach to the shader program. Null pointers in the list are ignored. |
| in | *vertexFormats* | Specifies the list of vertex formats. By default empty. |

**See also**

RenderSystem::CreateShaderProgram

#### 8.11.2.8 LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc ( const std::vector< Shader * > & *shaders*, const std::vector< VertexFormat > & *vertexFormats* = { } )

Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.

**Parameters**

| | | |
|---|---|---|
| in | *shaders* | Specifies the list of shaders to attach to the shader program. Null pointers in the list are ignored. |
| in | *vertexFormats* | Specifies the list of vertex formats. By default empty. |

**See also**

RenderSystem::CreateShaderProgram

#### 8.11.2.9 LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc ( uint64_t *size*, const StorageBufferType *storageType*, std::uint32_t *stride*, long *cpuAccessFlags* = CPUAccessFlags::ReadWrite )

Returns a BufferDescriptor structure for a storage buffer.

**See also**

RenderSystem::CreateBuffer

#### 8.11.2.10 LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *arrayLayers*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.11 LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc ( Format *format*, std::uint32_t *width*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.12 LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.13 LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.14 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, std::uint32_t *samples*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.15 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *samples*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.16 LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *depth*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.17 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.18 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc ( Format *format*, std::uint32_t *width*, std::uint32_t *height*, long *bindFlags* = (BindFlags::ColorAttachment|BindFlags::Sampled) )

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

**See also**

RenderSystem::CreateTexture

#### 8.11.2.19 LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc ( uint64_t *size*, const VertexFormat & *vertexFormat*, long *cpuAccessFlags* = 0 )

Returns a BufferDescriptor structure for a vertex buffer.

**See also**

RenderSystem::CreateBuffer

# Chapter 9

# Namespace Documentation

## 9.1 LLGL Namespace Reference

### Namespaces

- Constants

  *Namespace with all constants used as default arguments.*
- Log
- Version

  *Namespace with functions to determine LLGL version.*

### Classes

- struct ApplicationDescriptor

  *Application descriptor structure.*
- struct AttachmentClear

  *Attachment clear command structure.*
- struct AttachmentDescriptor

  *Render target attachment descriptor structure.*
- struct AttachmentFormatDescriptor

  *Render target attachment descriptor structure.*
- struct BindFlags

  *Flags for Buffer and Texture resources that describe for which purposes they will be used.*
- struct BindingDescriptor

  *Layout structure for a single binding point of the pipeline layout descriptor.*
- struct BlendDescriptor

  *Blending state descriptor structure.*
- struct BlendTargetDescriptor

  *Blend target state descriptor structure.*
- class Buffer

  *Hardware buffer interface.*
- class BufferArray

  *Hardware buffer container interface.*
- struct BufferDescriptor

  *Hardware buffer descriptor structure.*

- class Canvas

  *Canvas interface for mobile platforms.*
- struct CanvasDescriptor

  *Canvas descriptor structure.*
- struct ClearFlags

  *Command buffer clear flags.*
- struct ClearValue

  *Clear value structure for color, depth, and stencil clear operations.*
- 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 used for storing and encoding GPU commands.*
- struct CommandBufferDescriptor

  *Command buffer descriptor structure.*
- struct CommandBufferFlags

  *Command buffer creation flags.*
- class CommandQueue

  *Command queue interface.*
- class ComputePipeline

  *Compute pipeline interface.*
- struct ComputePipelineDescriptor

  *Compute pipeline descriptor structure.*
- struct CPUAccessFlags

  *CPU read/write access flag enumeration.*
- struct DepthBiasDescriptor

  *Depth bias descriptor structure to control fragment depth values.*
- struct DepthDescriptor

  *Depth state descriptor structure.*
- struct DispatchIndirectArguments

  *Format structure for the arguments of an indirect compute command.*
- class Display

  *Display interface to query the attributes of all connected displays/monitors.*
- struct DisplayModeDescriptor

  *Display mode descriptor structure.*
- struct DrawIndexedIndirectArguments

  *Format structure for the arguments of an indirect indexed draw command.*
- struct DrawIndirectArguments

  *Format structure for the arguments of an indirect draw command.*
- struct DrawPatchIndirectArgument

  *Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer.*
- struct DstImageDescriptor

  *Descriptor structure for an image that is used as destination for writing the image data.*
- struct Extent2D

  *2-Dimensional extent structure.*
- struct Extent3D

  *3-Dimensional extent structure.*
- class Fence

*Fence interface for CPU/GPU synchronization.*

- struct FormatDescriptor

  *Vertex and texture format descriptor structure.*
- struct FragmentAttribute

  *Fragment output attribute structure.*
- struct FrameProfile

  *Profile of a rendered frame.*
- class GraphicsPipeline

  *Graphics pipeline interface.*
- struct GraphicsPipelineDescriptor

  *Graphics pipeline descriptor structure.*
- class Image

  *Utility class to manage the storage and attributes of an image.*
- class Input

  *Default window event listener to receive user input.*
- class Interface

  *Base class for all interfaces in LLGL.*
- struct InterfaceID

  *Enumeration of all interfaces in LLGL.*
- struct MiscFlags

  *Miscellaneous resource flags.*
- struct MultiSamplingDescriptor

  *Multi-sampling descriptor structure.*
- struct NativeContextHandle

  *iOS native context handle structure.*
- struct NativeHandle

  *iOS native handle structure.*
- class NonCopyable

  *Base class for all classes with virtual function tables in LLGL.*
- struct Offset2D

  *2-Dimensional offset structure.*
- struct Offset3D

  *3-Dimensional offset structure.*
- struct OpenGLDependentStateDescriptor

  *Graphics API dependent state descriptor for the OpenGL renderer.*
- class PipelineLayout

  *Pipeline layout interface.*
- struct PipelineLayoutDescriptor

  *Pipeline layout descritpor structure.*
- class QueryHeap

  *Query heap interface that holds a certain number of queries that are all of the same type.*
- struct QueryHeapDescriptor

  *Query heap descriptor structure.*
- struct QueryPipelineStatistics

  *Query data structure for pipeline statistics.*
- struct RasterizerDescriptor

  *Rasterizer state descriptor structure.*
- class RenderContext

  *Render context interface.*
- struct RenderContextDescriptor

  *Render context descriptor structure.*

- struct RendererConfigurationOpenGL

  *OpenGL profile descriptor structure.*
- struct RendererConfigurationVulkan

  *Structure for a Vulkan renderer specific configuration.*
- struct RendererID

  *Renderer identification number enumeration.*
- struct RendererInfo

  *Renderer basic information structure.*
- struct RenderingCapabilities

  *Structure with all attributes describing the rendering capabilities of the render system.*
- class RenderingDebugger

  *Rendering debugger interface.*
- struct RenderingFeatures

  *Contains the attributes for all supported rendering features.*
- struct RenderingLimits

  *Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.*
- class RenderingProfiler

  *Rendering profiler model class.*
- class RenderPass

  *Render pass interface.*
- struct RenderPassDescriptor

  *Render pass descriptor structure.*
- class RenderSystem

  *Render system interface.*
- class RenderSystemChild

  *Base class for all interfaces whoes instances are owned by the RenderSystem.*
- struct RenderSystemConfiguration

  *Render system configuration structure.*
- struct RenderSystemDescriptor

  *Render system descriptor structure.*
- class RenderTarget

  *Render target interface.*
- struct RenderTargetDescriptor

  *Render target descriptor structure.*
- class Resource

  *Base class for all hardware resource interfaces.*
- class ResourceHeap

  *Resource heap interface.*
- struct ResourceHeapDescriptor

  *Resource heap descriptor structure.*
- struct ResourceViewDescriptor

  *Resource view descriptor structure.*
- class Sampler

  *Sampler interface.*
- struct SamplerDescriptor

  *Texture sampler descriptor structure.*
- struct Scissor

  *Scissor dimensions.*
- class Shader

  *Shader interface.*
- struct ShaderCompileFlags

*Shader compilation flags enumeration.*

- struct ShaderDescriptor

  *Shader source and binary code descriptor structure.*
- struct ShaderMacro

  *Shader macro structure with name and optional body definition.*
- class ShaderProgram

  *Shader program interface.*
- struct ShaderProgramDescriptor

  *Descriptor structure for shader programs.*
- struct ShaderReflection

  *Shader reflection structure.*
- struct ShaderResource

  *Shader reflection resource structure.*
- struct ShaderUniform

  *Shader reflection uniform structure.*
- struct SrcImageDescriptor

  *Descriptor structure for an image that is used as source for reading the image data.*
- struct StageFlags

  *Shader stage flags enumeration.*
- struct StencilDescriptor

  *Stencil state descriptor structure.*
- struct StencilFaceDescriptor

  *Stencil face descriptor structure.*
- struct StreamOutputAttribute

  *Stream-output attribute structure.*
- struct StreamOutputFormat

  *Stream-output format descriptor structure.*
- class Surface

  *The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms).*
- class Texture

  *Texture interface.*
- struct TextureDescriptor

  *Texture descriptor structure.*
- struct TextureLocation

  *Texture location structure: MIP-map level and offset.*
- struct TextureRegion

  *Texture region structure: MIP-map level, offset, and extent.*
- struct TextureSubresource

  *Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource.*
- struct TextureSwizzleRGBA

  *Texture component swizzle structure for red, green, blue, and alpha components.*
- struct TextureViewDescriptor

  *Texture view descriptor structure.*
- class Timer

  *Interface for a Timer class.*
- struct UninitializeTag

  *Common uninitialize tag.*
- struct VertexAttribute

  *Vertex input attribute structure.*
- struct VertexFormat

  *Vertex format structure.*

- struct VideoAdapterDescriptor

  *Video adapter descriptor structure.*
- struct VideoModeDescriptor

  *Video mode descriptor structure.*
- struct VideoOutputDescriptor

  *Video output structure.*
- struct Viewport

  *Viewport dimensions.*
- struct VsyncDescriptor

  *Vertical-synchronization (Vsync) descriptor structure.*
- class Window

  *Window interface for desktop platforms.*
- struct WindowBehavior

  *Window behavior structure.*
- struct WindowDescriptor

  *Window descriptor structure.*

## Typedefs

- template<typename T >
  using ColorRGBT = Color< T, 3 >
- using ColorRGBb = ColorRGBT< bool >
- using ColorRGBf = ColorRGBT< float >
- using ColorRGBd = ColorRGBT< double >
- using ColorRGBub = ColorRGBT< std::uint8_t >
- template<typename T >
  using ColorRGBAT = Color< T, 4 >
- using ColorRGBAb = ColorRGBAT< bool >
- using ColorRGBAf = ColorRGBAT< float >
- using ColorRGBAd = ColorRGBAT< double >
- using ColorRGBAub = ColorRGBAT< std::uint8_t >
- 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 ValidateRenderingCapsFunc = std::function< bool(const std::string &info, const std::string &attrib)>

  *Callback interface for the ValidateRenderingCaps function.*
- using UniformLocation = std::int32_t

  *Shader uniform location type, as zero-based index in 32-bit signed integer format.*

## Enumerations

- enum StorageBufferType {
  StorageBufferType::Undefined, StorageBufferType::Buffer, StorageBufferType::StructuredBuffer, StorageBufferType::ByteAddressBuffer,
  StorageBufferType::RWBuffer, StorageBufferType::RWStructuredBuffer, StorageBufferType::RWByteAddressBuffer, StorageBufferType::AppendStructuredBuffer,
  StorageBufferType::ConsumeStructuredBuffer }

  *Storage buffer type enumeration.*

- enum RenderConditionMode { RenderConditionMode::Wait, RenderConditionMode::NoWait, RenderConditionMode::ByRegionWait, RenderConditionMode::ByRegionNoWait, RenderConditionMode::WaitInverted, RenderConditionMode::NoWaitInverted, RenderConditionMode::ByRegionWaitInverted, RenderConditionMode::ByRegionNoWaitInverted }

  *Render condition mode enumeration.*

- enum Format { Format::Undefined, Format::A8UNorm, Format::R8UNorm, Format::R8SNorm, Format::R8UInt, Format::R8SInt, Format::R16UNorm, Format::R16SNorm, Format::R16UInt, Format::R16SInt, Format::R16Float, Format::R32UInt, Format::R32SInt, Format::R32Float, Format::R64Float, Format::RG8UNorm, Format::RG8SNorm, Format::RG8UInt, Format::RG8SInt, Format::RG16UNorm, Format::RG16SNorm, Format::RG16UInt, Format::RG16SInt, Format::RG16Float, Format::RG32UInt, Format::RG32SInt, Format::RG32Float, Format::RG64Float, Format::RGB8UNorm, Format::RGB8UNorm_sRGB, Format::RGB8SNorm, Format::RGB8UInt, Format::RGB8SInt, Format::RGB16UNorm, Format::RGB16SNorm, Format::RGB16UInt, Format::RGB16SInt, Format::RGB16Float, Format::RGB32UInt, Format::RGB32SInt, Format::RGB32Float, Format::RGB64Float, Format::RGBA8UNorm, Format::RGBA8UNorm_sRGB, Format::RGBA8SNorm, Format::RGBA8UInt, Format::RGBA8SInt, Format::RGBA16UNorm, Format::RGBA16SNorm, Format::RGBA16UInt, Format::RGBA16SInt, Format::RGBA16Float, Format::RGBA32UInt, Format::RGBA32SInt, Format::RGBA32Float, Format::RGBA64Float, Format::BGRA8UNorm, Format::BGRA8UNorm_sRGB, Format::BGRA8SNorm, Format::BGRA8UInt, Format::BGRA8SInt, Format::RGB10A2UNorm, Format::RGB10A2UInt, Format::RG11B10Float, Format::RGB9E5Float, Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, Format::D32FloatS8X24UInt, Format::BC1RGB, Format::BC1RGBA, Format::BC2RGBA, Format::BC3RGBA }

  *Texture and vertex attribute format enumeration used for GPU side operations.*

- enum ImageFormat { ImageFormat::Alpha, ImageFormat::R, ImageFormat::RG, ImageFormat::RGB, ImageFormat::BGR, ImageFormat::RGBA, ImageFormat::BGRA, ImageFormat::ARGB, ImageFormat::ABGR, ImageFormat::Depth, ImageFormat::DepthStencil, ImageFormat::CompressedRGB, ImageFormat::CompressedRGBA }

  *Image format enumeration that applies to each pixel of an image.*

- enum DataType { DataType::Undefined, DataType::Int8, DataType::UInt8, DataType::Int16, DataType::UInt16, DataType::Int32, DataType::UInt32, DataType::Float16, DataType::Float32, DataType::Float64 }

  *Data types enumeration used for CPU side operations.*

- 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, Primitive←

Topology::Patches21,
PrimitiveTopology::Patches22, PrimitiveTopology::Patches23, PrimitiveTopology::Patches24, Primitive↩
Topology::Patches25,
PrimitiveTopology::Patches26, PrimitiveTopology::Patches27, PrimitiveTopology::Patches28, Primitive↩
Topology::Patches29,
PrimitiveTopology::Patches30, PrimitiveTopology::Patches31, PrimitiveTopology::Patches32 }

*Primitive topology enumeration.*

- enum CompareOp {
  CompareOp::NeverPass, CompareOp::Less, CompareOp::Equal, CompareOp::LessEqual,
  CompareOp::Greater, CompareOp::NotEqual, CompareOp::GreaterEqual, CompareOp::AlwaysPass }

  *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::DstColor, BlendOp::InvDstColor,
  BlendOp::DstAlpha, BlendOp::InvDstAlpha, 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 LogicOp {
  LogicOp::Disabled, LogicOp::Clear, LogicOp::Set, LogicOp::Copy,
  LogicOp::CopyInverted, LogicOp::NoOp, LogicOp::Invert, LogicOp::AND,
  LogicOp::ANDReverse, LogicOp::ANDInverted, LogicOp::NAND, LogicOp::OR,
  LogicOp::ORReverse, LogicOp::ORInverted, LogicOp::NOR, LogicOp::XOR,
  LogicOp::Equiv }

  *Logical pixel operation enumeration.*

- 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,
Key::Any }

*Input key codes.*

- enum QueryType {
QueryType::SamplesPassed, QueryType::AnySamplesPassed, QueryType::AnySamplesPassedConservative,
QueryType::TimeElapsed,
QueryType::StreamOutPrimitivesWritten, QueryType::StreamOutOverflow, QueryType::PipelineStatistics }

  *Query type enumeration.*

- enum OpenGLContextProfile { OpenGLContextProfile::CompatibilityProfile, OpenGLContextProfile::Core↩
Profile, OpenGLContextProfile::ESProfile }

  *OpenGL context profile enumeration.*

- enum ErrorType { ErrorType::InvalidArgument, ErrorType::InvalidState, ErrorType::UnsupportedFeature,
ErrorType::UndefinedBehavior }

  *Rendering debugger error types enumeration.*

- enum WarningType { WarningType::ImproperArgument, WarningType::ImproperState, WarningType::↩
PointlessOperation, WarningType::VaryingBehavior }

  *Rendering debugger warning types enumeration.*

- enum AttachmentLoadOp { AttachmentLoadOp::Undefined, AttachmentLoadOp::Load, AttachmentLoad↩
Op::Clear }

  *Enumeration for render pass attachment load operations.*

- enum AttachmentStoreOp { AttachmentStoreOp::Undefined, AttachmentStoreOp::Store }

  *Enumeration for render pass attachment store operations.*

- enum ShadingLanguage { ShadingLanguage::GLSL = (0x10000), ShadingLanguage::GLSL_110 = (0x10000 | 110), ShadingLanguage::GLSL_120 = (0x10000 | 120), ShadingLanguage::GLSL_130 = (0x10000 | 130), ShadingLanguage::GLSL_140 = (0x10000 | 140), ShadingLanguage::GLSL_150 = (0x10000 | 150), ShadingLanguage::GLSL_330 = (0x10000 | 330), ShadingLanguage::GLSL_400 = (0x10000 | 400), ShadingLanguage::GLSL_410 = (0x10000 | 410), ShadingLanguage::GLSL_420 = (0x10000 | 420), ShadingLanguage::GLSL_430 = (0x10000 | 430), ShadingLanguage::GLSL_440 = (0x10000 | 440), ShadingLanguage::GLSL_450 = (0x10000 | 450), ShadingLanguage::GLSL_460 = (0x10000 | 460), ShadingLanguage::ESSL = (0x20000), ShadingLanguage::ESSL_100 = (0x20000 | 100), ShadingLanguage::ESSL_300 = (0x20000 | 300), ShadingLanguage::ESSL_310 = (0x20000 | 310), ShadingLanguage::ESSL_320 = (0x20000 | 320), ShadingLanguage::HLSL = (0x30000), ShadingLanguage::HLSL_2_0 = (0x30000 | 200), ShadingLanguage::HLSL_2_0a = (0x30000 | 201), ShadingLanguage::HLSL_2_0b = (0x30000 | 202), ShadingLanguage::HLSL_3_0 = (0x30000 | 300), ShadingLanguage::HLSL_4_0 = (0x30000 | 400), ShadingLanguage::HLSL_4_1 = (0x30000 | 410), ShadingLanguage::HLSL_5_0 = (0x30000 | 500), ShadingLanguage::HLSL_5_1 = (0x30000 | 510), ShadingLanguage::Metal = (0x40000), ShadingLanguage::Metal_1_0 = (0x40000 | 100), ShadingLanguage::Metal_1_1 = (0x40000 | 110), ShadingLanguage::Metal_1_2 = (0x40000 | 120), ShadingLanguage::Metal_2_0 = (0x40000 | 200), ShadingLanguage::Metal_2_1 = (0x40000 | 210), ShadingLanguage::SPIRV = (0x50000), ShadingLanguage::SPIRV_100 = (0x50000 | 100), ShadingLanguage::VersionBitmask = 0x0000ffff }

  *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 CPUAccess { CPUAccess::ReadOnly, CPUAccess::WriteOnly, CPUAccess::WriteDiscard, CPUAccess::ReadWrite }

  *Classifications of CPU access to mapped resources.*

- enum AttachmentType { AttachmentType::Color, AttachmentType::Depth, AttachmentType::DepthStencil, AttachmentType::Stencil }

  *Render target attachment type enumeration.*

- enum ResourceType { ResourceType::Undefined, ResourceType::Buffer, ResourceType::Texture, ResourceType::Sampler }

  *Hardware resource type enumeration.*

- enum SamplerAddressMode { SamplerAddressMode::Repeat, SamplerAddressMode::Mirror, SamplerAddressMode::Clamp, SamplerAddressMode::Border, SamplerAddressMode::MirrorOnce }

  *Technique for resolving texture coordinates that are outside of the range [0, 1].*

- enum SamplerFilter { SamplerFilter::Nearest, SamplerFilter::Linear }

  *Sampling filter enumeration.*

- enum ShaderType { ShaderType::Undefined, ShaderType::Vertex, ShaderType::TessControl, ShaderType::TessEvaluation, ShaderType::Geometry, ShaderType::Fragment, ShaderType::Compute }

  *Shader type enumeration.*

- enum ShaderSourceType { ShaderSourceType::CodeString, ShaderSourceType::CodeFile, ShaderSourceType::BinaryBuffer, ShaderSourceType::BinaryFile }

  *Shader source type enumeration.*

- enum UniformType {
  UniformType::Undefined, UniformType::Float1, UniformType::Float2, UniformType::Float3, UniformType::Float4, UniformType::Double1, UniformType::Double2, UniformType::Double3, UniformType::Double4, UniformType::Int1, UniformType::Int2, UniformType::Int3, UniformType::Int4, UniformType::UInt1, UniformType::UInt2, UniformType::UInt3, UniformType::UInt4, UniformType::Bool1, UniformType::Bool2, UniformType::Bool3, UniformType::Bool4, UniformType::Float2x2, UniformType::Float2x3, UniformType::Float2x4, UniformType::Float3x2, UniformType::Float3x3, UniformType::Float3x4, UniformType::Float4x2, UniformType::Float4x3, UniformType::Float4x4, UniformType::Double2x2, UniformType::Double2x3, UniformType::Double2x4, UniformType::Double3x2, UniformType::Double3x3, UniformType::Double3x4, UniformType::Double4x2, UniformType::Double4x3, UniformType::Double4x4, UniformType::Sampler, UniformType::Image, UniformType::AtomicCounter }

  *Shader uniform type enumeration.*
- enum SystemValue {
  SystemValue::Undefined, SystemValue::ClipDistance, SystemValue::Color, SystemValue::CullDistance, SystemValue::Depth, SystemValue::DepthGreater, SystemValue::DepthLess, SystemValue::FrontFacing, SystemValue::InstanceID, SystemValue::Position, SystemValue::PrimitiveID, SystemValue::RenderTarget↩Index,
  SystemValue::SampleMask, SystemValue::SampleID, SystemValue::Stencil, SystemValue::VertexID, SystemValue::ViewportIndex }

  *Enumeration of system values for vertex input or fragment output attributes.*
- enum TextureType {
  TextureType::Texture1D, TextureType::Texture2D, TextureType::Texture3D, TextureType::TextureCube, TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, TextureType↩::Texture2DMS,
  TextureType::Texture2DMSArray }

  *Texture type enumeration.*
- enum TextureSwizzle {
  TextureSwizzle::Zero, TextureSwizzle::One, TextureSwizzle::Red, TextureSwizzle::Green, TextureSwizzle::Blue, TextureSwizzle::Alpha }

  *Texture component swizzle enumeration.*

## Functions

- LLGL_EXPORT bool IsRWBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a read/write (RW) buffer.*
- LLGL_EXPORT bool IsTypedBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a simply typed buffer.*
- LLGL_EXPORT bool IsStructuredBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a structured buffer.*
- LLGL_EXPORT bool IsByteAddressBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a byte addresse buffer.*
- template<typename T >
  T MaxColorValue ()

  *Returns the maximal color value for the data type T. By default 1.*
- template<>
  std::uint8_t MaxColorValue< std::uint8_t > ()

  *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 Dst , typename Src >
  Dst CastColorValue (const Src &value)

*Casts the specified color value and transforms it from the source data type range to the destination data type range.*

- template<>
  bool CastColorValue< bool, bool > (const bool &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  float CastColorValue< float, float > (const float &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  double CastColorValue< double, double > (const double &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  std::uint8_t CastColorValue< std::uint8_t, std::uint8_t > (const std::uint8_t &value)

  *Specialized template which merely passes the input value as output.*
- 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>
  Color< T, N > operator/ (const T &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)

  *Returns true if all components of both colors 'lhs' and 'rhs' are equal.*
- template<typename T , std::size_t N>
  bool operator!= (const Color< T, N > &lhs, const Color< T, N > &rhs)

  *Returns true if any component of both colors 'lhs' and 'rhs' are unequal.*
- LLGL_EXPORT bool operator== (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

  *Compares the two specified display mode descriptors on equality.*
- LLGL_EXPORT bool operator!= (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

  *Compares the two specified display mode descriptors on inequality.*
- LLGL_EXPORT bool CompareSWO (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

  *Compares the two display modes in a strict-weak-order (SWO) fashion.*
- LLGL_EXPORT Extent2D GetExtentRatio (const Extent2D &extent)

  *Returns the ratio of the specified extent as another extent, i.e. all attributes are divided by their greatest common divisor.*
- LLGL_EXPORT const FormatDescriptor & GetFormatDesc (const Format format)

  *Returns the descriptor for the specified hardware format or a zero-initialized descriptor in case of an invalid argument.*
- LLGL_EXPORT bool IsCompressedFormat (const Format format)

  *Returns true if the specified hardware format is a compressed format, i.e. either Format::BC1RGB, Format::BC1R↩GBA, Format::BC2RGBA, or Format::BC3RGBA.*
- LLGL_EXPORT bool IsDepthStencilFormat (const Format format)

  *Returns true if the specified hardware format is a depth or depth-stencil format, i.e. Format::D16UNorm, Format::↩D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool IsDepthFormat (const Format format)

*Returns true if the specified hardware format is a depth format, i.e. Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*

- LLGL_EXPORT bool IsStencilFormat (const Format format)

  *Returns true if the specified hardware format is a stencil format, i.e. Format::D24UNormS8UInt or Format::D32↩FloatS8X24UInt.*

- LLGL_EXPORT bool IsNormalizedFormat (const Format format)

  *Returns true if the specified hardware format is a normalized format (like Format::RGBA8UNorm, Format::R8SNorm etc.).*

- LLGL_EXPORT bool IsIntegralFormat (const Format format)

  *Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::R8SInt etc.).*

- LLGL_EXPORT bool IsFloatFormat (const Format format)

  *Returns true if the specified hardware format is a floating-point format (like Format::RGBA32Float, Format::R32Float etc.).*

- LLGL_EXPORT std::uint32_t DataTypeSize (const DataType dataType)

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

- LLGL_EXPORT bool IsIntDataType (const DataType dataType)

  *Determines if the argument refers to a signed integer data type.*

- LLGL_EXPORT bool IsUIntDataType (const DataType dataType)

  *Determines if the argument refers to an unsigned integer data type.*

- LLGL_EXPORT bool IsFloatDataType (const DataType dataType)

  *Determines if the argument refers to a floating-pointer data type.*

- LLGL_EXPORT bool IsPrimitiveTopologyPatches (const PrimitiveTopology primitiveTopology)

  *Returns true if the specified primitive topology is a patch list.*

- LLGL_EXPORT std::uint32_t GetPrimitiveTopologyPatchSize (const PrimitiveTopology primitiveTopology)

  *Returns the number of patch control points of the specified primitive topology (in range [1, 32]), or 0 if the topology is not a patch list.*

- LLGL_EXPORT std::uint32_t ImageFormatSize (const ImageFormat imageFormat)

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

- LLGL_EXPORT std::uint32_t ImageDataSize (const ImageFormat imageFormat, const DataType dataType, std::uint32_t numPixels)

  *Returns the required data size (in bytes) of an image with the specified format, data type, and number of pixels.*

- LLGL_EXPORT bool IsCompressedFormat (const ImageFormat imageFormat)

  *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 imageFormat)

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

- LLGL_EXPORT bool ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const DstImage↩Descriptor &dstImageDesc, std::size_t threadCount=0)

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

- LLGL_EXPORT ByteBuffer ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, ImageFormat dstFormat, DataType dstDataType, std::size_t threadCount=0)

  *Converst the image format and data type of the source image (only uncompressed color formats) and returns the new generated image buffer.*

- LLGL_EXPORT ByteBuffer GenerateImageBuffer (ImageFormat format, DataType dataType, std::size_↩t imageSize, const ColorRGBAd &fillColor)

  *Generates an image buffer with the specified fill data for each pixel.*

- LLGL_EXPORT ByteBuffer GenerateEmptyByteBuffer (std::size_t bufferSize, bool initialize=true)

  *Generates a new byte buffer with zeros in each byte.*

- LLGL_EXPORT bool operator== (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)

  *Compares the two specified V-sync descriptors on equality.*

- LLGL_EXPORT bool operator!= (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)

  *Compares the two specified V-sync descriptors on inequality.*

- LLGL_EXPORT bool operator== (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)

  *Compares the two specified video mode descriptors on equality.*
- LLGL_EXPORT bool operator!= (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)

  *Compares the two specified video mode descriptors on inequality.*
- LLGL_EXPORT bool ValidateRenderingCaps (const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={})

  *Validates the presence of the specified required rendering capabilities.*
- LLGL_EXPORT bool IsShaderSourceCode (const ShaderSourceType type)

  *Returns true if the specified shader source type is either ShaderSourceType::CodeString or ShaderSourceType::CodeFile.*
- LLGL_EXPORT bool IsShaderSourceBinary (const ShaderSourceType type)

  *Returns true if the specified shader source type is either ShaderSourceType::BinaryBuffer or ShaderSourceType::BinaryFile.*
- LLGL_EXPORT bool operator== (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)
- LLGL_EXPORT bool operator!= (const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs)
- LLGL_EXPORT const char * ToString (const ShaderType t)

  *Returns a string representation for the spcified ShaderType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * ToString (const ErrorType t)

  *Returns a string representation for the specified ErrorType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * ToString (const WarningType t)

  *Returns a string representation for the specified WarningType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * ToString (const ShadingLanguage t)

  *Returns a string representation for the specified ShadingLanguage value, or null if the input type is invalid.*
- LLGL_EXPORT const char * ToString (const Format t)

  *Returns a string representation for the specified Format value, or null if the input type is invalid.*
- LLGL_EXPORT const char * ToString (const TextureType t)

  *Returns a string representation for the specified TextureType value, or null if the input type is invalid.*
- LLGL_EXPORT std::uint32_t NumMipLevels (std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1)

  *Returns the number of MIP-map levels for a texture with the specified size.*
- LLGL_EXPORT std::uint32_t NumMipLevels (const TextureDescriptor &textureDesc)

  *Returns the number of MIP-map levels for the specified texture descriptor.*
- LLGL_EXPORT std::uint32_t TextureBufferSize (const Format format, std::uint32_t numTexels)

  *Returns the required buffer size (in bytes) of a texture with the specified hardware format and number of texels.*
- LLGL_EXPORT std::uint32_t TextureSize (const TextureDescriptor &textureDesc)

  *Returns the texture size (in texels) of the specified texture descriptor, or zero if the texture type is invalid.*
- LLGL_EXPORT bool IsMipMappedTexture (const TextureDescriptor &textureDesc)

  *Returns true if the specified texture descriptor describes a texture with MIP-mapping enabled.*
- 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 bool IsCubeTexture (const TextureType type)

  *Returns true if the specified texture type is a cube texture.*
- LLGL_EXPORT bool IsTextureSwizzleIdentity (const TextureSwizzleRGBA &swizzle)

  *Returns true if the specified texture swizzling is equal to the identity mapping.*
- template<typename T >
  const T * CastTo (const Interface *obj)

  *Returns a constant pointer to this instance of type `const T*` if it implements the specified interface.*
- template<typename T >
  T * CastTo (Interface *obj)

  *Returns a pointer to this instance of type `T*` if it implements the specified interface.*

- template<typename T >
  const T & CastTo (const Interface &obj)

  *Returns a constant reference to this instance of type `const T&` if it implements the specified interface.*
- template<typename T >
  T & CastTo (Interface &obj)

  *Returns a reference to this instance of type `T&` if it implements the specified interface.*
- LLGL_EXPORT Extent2D operator+ (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent2D operator- (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D operator+ (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D operator- (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Offset2D operator+ (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset2D operator- (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D operator+ (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D operator- (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- bool operator== (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool operator!= (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool operator== (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool operator!= (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool operator== (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool operator!= (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*
- bool operator== (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool operator!= (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*
- LLGL_EXPORT TextureDescriptor Texture1DDesc (Format format, std::uint32_t width, long bind↩Flags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture1D type.*
- LLGL_EXPORT TextureDescriptor Texture2DDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2D type.*
- LLGL_EXPORT TextureDescriptor Texture3DDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture3D type.*
- LLGL_EXPORT TextureDescriptor TextureCubeDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCube type.*
- LLGL_EXPORT TextureDescriptor Texture1DArrayDesc (Format format, std::uint32_t width, std::uint32_↩t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

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

- LLGL_EXPORT TextureDescriptor Texture2DArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.*
- LLGL_EXPORT TextureDescriptor TextureCubeArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.*
- LLGL_EXPORT TextureDescriptor Texture2DMSDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.*
- LLGL_EXPORT TextureDescriptor Texture2DMSArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.*
- LLGL_EXPORT BufferDescriptor VertexBufferDesc (uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a vertex buffer.*
- LLGL_EXPORT BufferDescriptor IndexBufferDesc (uint64_t size, const Format format, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for an index buffer.*
- LLGL_EXPORT BufferDescriptor ConstantBufferDesc (uint64_t size, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a constant buffer.*
- LLGL_EXPORT BufferDescriptor StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite)

  *Returns a BufferDescriptor structure for a storage buffer.*
- LLGL_EXPORT ShaderDescriptor ShaderDescFromFile (const ShaderType type, const char *filename, const char *entryPoint=nullptr, const char *profile=nullptr, long flags=0)

  *Returns a ShaderDescriptor structure.*
- LLGL_EXPORT ShaderProgramDescriptor ShaderProgramDesc (const std::initializer_list< Shader * > &shaders, const std::initializer_list< VertexFormat > &vertexFormats={})

  *Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*
- LLGL_EXPORT ShaderProgramDescriptor ShaderProgramDesc (const std::vector< Shader * > &shaders, const std::vector< VertexFormat > &vertexFormats={})

  *Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*
- LLGL_EXPORT PipelineLayoutDescriptor PipelineLayoutDesc (const ShaderReflection &reflection)

  *Converts the specified shader reflection descriptor into a pipeline layout descriptor.*
- LLGL_EXPORT PipelineLayoutDescriptor PipelineLayoutDesc (const char *layoutSignature)

  *Generates a pipeline layout descriptor by parsing the specified string.*
- LLGL_EXPORT RenderPassDescriptor RenderPassDesc (const RenderTargetDescriptor &renderTargetDesc)

  *Converts the specified render target descriptor into a render pass descriptor with default settings.*
- LLGL_EXPORT bool operator== (const VertexAttribute &lhs, const VertexAttribute &rhs)

  *Compares the two VertexAttribute types for equality (including their names and all other members).*
- LLGL_EXPORT bool operator!= (const VertexAttribute &lhs, const VertexAttribute &rhs)

  *Compares the two VertexAttribute types for inequality (including their names and all other members).*

### 9.1.1 Typedef Documentation

#### 9.1.1.1 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

#### 9.1.1.2 using LLGL::ColorRGBAb = typedef ColorRGBAT<bool>

#### 9.1.1.3 using LLGL::ColorRGBAd = typedef ColorRGBAT<double>

#### 9.1.1.4 using LLGL::ColorRGBAf = typedef ColorRGBAT<float>

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

#### 9.1.1.6 using LLGL::ColorRGBAub = typedef ColorRGBAT<std::uint8_t>

#### 9.1.1.7 using LLGL::ColorRGBb = typedef ColorRGBT<bool>

#### 9.1.1.8 using LLGL::ColorRGBd = typedef ColorRGBT<double>

#### 9.1.1.9 using LLGL::ColorRGBf = typedef ColorRGBT<float>

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

#### 9.1.1.11 using LLGL::ColorRGBub = typedef ColorRGBT<std::uint8_t>

#### 9.1.1.12 using LLGL::UniformLocation = typedef std::int32_t

Shader uniform location type, as zero-based index in 32-bit signed integer format.

### 9.1.2 Enumeration Type Documentation

#### 9.1.2.1 enum LLGL::AttachmentLoadOp [strong]

Enumeration for render pass attachment load operations.

**See also**

AttachmentFormatDescriptor

**Enumerator**

***Undefined*** We don't care about the previous content of the respective render target attachment.

***Load*** Loads the previous content of the respective render target attachment.

***Clear*** Clear the previous content of the respective render target attachment.

**Remarks**

The clear value used for this load operation is specified at the CommandBuffer::BeginRenderPass function.

**See also**

CommandBuffer::BeginRenderPass

#### 9.1.2.2 enum LLGL::AttachmentStoreOp [strong]

Enumeration for render pass attachment store operations.

**See also**

AttachmentFormatDescriptor

**Enumerator**

***Undefined*** We don't care about the outcome of the respective render target attachment.

**Remarks**

Can be used, for example, if we only need the depth buffer for the depth test, but nothing is written to it.

***Store*** Stores the outcome in the respective render target attachment.

#### 9.1.2.3 enum LLGL::AttachmentType [strong]

Render target attachment type enumeration.

**See also**

AttachmentDescriptor

**Enumerator**

***Color*** Attachment is used for color output.

**Remarks**

A texture attached to a render target with this attachment type must have been created with binding flag BindFlags::ColorAttachment.

***Depth*** Attachment is used for depth component output.

**Remarks**

A texture attached to a render target with this attachment type must have been created with the binding flag BindFlags::DepthStencilAttachment.

***DepthStencil*** Attachment is used for depth component and stencil index output.

**Remarks**

A texture attached to a render target with this attachment type must have been created with the binding flag BindFlags::DepthStencilAttachment.

***Stencil*** Attachment is used for stencil index output.

**Remarks**

A texture attached to a render target with this attachment type must have been created with the binding flag BindFlags::DepthStencilAttachment.

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

Blending arithmetic operations enumeration.

**See also**

BlendTargetDescriptor::colorArithmetic
BlendTargetDescriptor::alphaArithmetic

**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.

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

Blending operations enumeration.

**See also**

BlendTargetDescriptor

**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.

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

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

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

***InvDstAlpha*** 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**

CommandBuffer::SetBlendFactor

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

**See also**

CommandBuffer::SetBlendFactor

***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.

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

Clipping depth range enumeration.

**See also**

RenderingCapabilities::clippingRange

**Enumerator**

***MinusOneToOne*** Specifies the clipping depth range [-1, 1].

**Note**

Native clipping depth range in: OpenGL.

***ZeroToOne*** Specifies the clipping depth range [0, 1].

**Note**

Native clipping depth range in: Direct3D 11, Direct3D 12, Vulkan.

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

Compare operations enumeration.

**Remarks**

This operation is used for depth tests, stencil tests, and texture sample comparisons.

**See also**

DepthDescriptor::compareOp
StencilFaceDescriptor::compareOp
SamplerDescriptor::compareOp

**Enumerator**

***NeverPass*** 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.

***AlwaysPass*** Comparison always passes.

#### 9.1.2.8 enum LLGL::CPUAccess [strong]

Classifications of CPU access to mapped resources.

**See also**

RenderSystem::MapBuffer
CPUAccessFlags

**Enumerator**

***ReadOnly*** CPU read access to a mapped resource.

**Remarks**

If this is used for RenderSystem::MapBuffer, the respective buffer must have been created with the BufferFlags::MapReadAccess flag.

***WriteOnly*** CPU write access to a mapped resource.

**Remarks**

If this is used for RenderSystem::MapBuffer, the respective buffer must have been created with the BufferFlags::MapWriteAccess flag.

***WriteDiscard*** CPU write access to a mapped resource, where the previous content *can* be discarded.

**Remarks**

If this is used for RenderSystem::MapBuffer, the respective buffer must have been created with the BufferFlags::MapWriteAccess flag.

**Note**

Whether the previous content is discarded depends on the rendering API.

***ReadWrite*** CPU read and write access to a mapped resource.

**Remarks**

If this is used for RenderSystem::MapBuffer, the respective buffer must have been created with both the BufferFlags::MapReadAccess and the BufferFlags::MapWriteAccess flag.

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

Polygon culling modes enumeration.

**See also**

RasterizerDescriptor::cullMode

**Enumerator**

***Disabled*** No culling.

***Front*** Front face culling.

***Back*** Back face culling.

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

Data types enumeration used for CPU side operations.

**Remarks**

The counterpart of this enumeration for GPU side operations is LLGL::Format.

**See also**

SrcImageDescriptor::dataType
DstImageDescriptor::dataType

**Todo** Maybe replace `DataType` and `ImageFormat` by LLGL::Format.

**Enumerator**

***Undefined*** Undefined data type.

***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).

***Float16*** 16-bit floating-point (half).

***Float32*** 32-bit floating-point (float).

***Float64*** 64-bit real type (double).

#### 9.1.2.11 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).

***UndefinedBehavior*** Error due to arguments that cause undefined behavior.

#### 9.1.2.12 enum LLGL::Format [strong]

Texture and vertex attribute format enumeration used for GPU side operations.

**Remarks**

The counterpart of this enumeration for CPU side operations are LLGL::ImageFormat and LLGL::DataType.
Packed color formats are stored from least significant bit (LSB) to most significant bit (MSB), i.e. for the Format::RGB10A2UNorm the bit pattern is `A[31:30], B[29:20], G[19:10], R[9:0]` for instance.
The following table illustrates the bit-size for each component of all ordinary and packed color formats (depth-stencil and compressed formats excluded), where **`n`** denotes unsigned normalized formats (`UNorm`), **`s`** denotes signed normalized formats (`SNorm`), **`u`** denotes unsigned integer formats (`UInt`), **`i`** denotes signed integer formats (`SInt`), and **`f`** denotes floating-point formats (`Float`):

| Format | Bits | Red | Green | Blue | Alpha | Shared |
|---|---|---|---|---|---|---|
| Format::A8UNorm | 8 | | | | n8 | |
| Format::R8UNorm | 8 | n8 | | | | |
| Format::R8SNorm | 8 | s8 | | | | |
| Format::R8UInt | 8 | u8 | | | | |
| Format::R8SInt | 8 | i8 | | | | |
| Format::R16UNorm | 16 | n16 | | | | |
| Format::R16SNorm | 16 | s16 | | | | |
| Format::R16UInt | 16 | u16 | | | | |
| Format::R16SInt | 16 | i16 | | | | |
| Format::R16Float | 16 | f16 | | | | |
| Format::R32UInt | 32 | u32 | | | | |
| Format::R32SInt | 32 | i32 | | | | |
| Format::R32Float | 32 | f32 | | | | |
| Format::R64Float | 64 | f64 | | | | |
| Format::RG8UNorm | 16 | n8 | n8 | | | |
| Format::RG8SNorm | 16 | s8 | s8 | | | |
| Format::RG8UInt | 16 | u8 | u8 | | | |
| Format::RG8SInt | 16 | i8 | i8 | | | |
| Format::RG16UNorm | 32 | n16 | n16 | | | |
| Format::RG16SNorm | 32 | s16 | s16 | | | |
| Format::RG16UInt | 32 | u16 | u16 | | | |
| Format::RG16SInt | 32 | i16 | i16 | | | |
| Format::RG16Float | 32 | f16 | f16 | | | |
| Format::RG32UInt | 64 | u32 | u32 | | | |
| Format::RG32SInt | 64 | i32 | i32 | | | |
| Format::RG32Float | 64 | f32 | f32 | | | |
| Format::RG64Float | 128 | f64 | f64 | | | |
| Format::RGB8UNorm | 24 | n8 | n8 | n8 | | |
| Format::RGB8UNorm_sRGB | 24 | n8 | n8 | n8 | | |
| Format::RGB8SNorm | 24 | s8 | s8 | s8 | | |
| Format::RGB8UInt | 24 | u8 | u8 | u8 | | |
| Format::RGB8SInt | 24 | i8 | i8 | i8 | | |
| Format::RGB16UNorm | 48 | n16 | n16 | n16 | | |
| Format::RGB16SNorm | 48 | s16 | s16 | s16 | | |
| Format::RGB16UInt | 48 | u16 | u16 | u16 | | |
| Format::RGB16SInt | 48 | i16 | i16 | i16 | | |
| Format::RGB16Float | 48 | f16 | f16 | f16 | | |
| Format::RGB32UInt | 96 | u32 | u32 | u32 | | |
| Format::RGB32SInt | 96 | i32 | i32 | i32 | | |
| Format::RGB32Float | 96 | f32 | f32 | f32 | | |
| Format::RGB64Float | 192 | f64 | f64 | f64 | | |
| Format::RGBA8UNorm | 32 | n8 | n8 | n8 | n8 | |
| Format::RGBA8UNorm_sRGB | 32 | n8 | n8 | n8 | n8 | |
| Format::RGBA8SNorm | 32 | s8 | s8 | s8 | s8 | |
| Format::RGBA8UInt | 32 | u8 | u8 | u8 | u8 | |
| Format::RGBA8SInt | 32 | i8 | i8 | i8 | i8 | |
| Format::RGBA16UNorm | 64 | n16 | n16 | n16 | n16 | |
| Format::RGBA16SNorm | 64 | s16 | s16 | s16 | s16 | |
| Format::RGBA16UInt | 64 | u16 | u16 | u16 | u16 | |
| Format::RGBA16SInt | 64 | i16 | i16 | i16 | i16 | |
| Format::RGBA16Float | 64 | f16 | f16 | f16 | f16 | |
| Format::RGBA32UInt | 128 | u32 | u32 | u32 | u32 | |

| Format | Bits | Red | Green | Blue | Alpha | Shared |
|---|---|---|---|---|---|---|
| Format::RGBA32SInt | **128** | i32 | i32 | i32 | i32 | |
| Format::RGBA32Float | **128** | f32 | f32 | f32 | f32 | |
| Format::RGBA64Float | **256** | f64 | f64 | f64 | f64 | |
| Format::BGRA8UNorm | **32** | n8 | n8 | n8 | n8 | |
| Format::BGRA8UNorm_sRGB | **32** | n8 | n8 | n8 | n8 | |
| Format::BGRA8SNorm | **32** | s8 | s8 | s8 | s8 | |
| Format::BGRA8UInt | **32** | u8 | u8 | u8 | u8 | |
| Format::BGRA8SInt | **32** | i8 | i8 | i8 | i8 | |
| Format::RGB10A2UNorm | **32** | n10 | n10 | n10 | n2 | |
| Format::RGB10A2UInt | **32** | u10 | u10 | u10 | u2 | |
| Format::RG11B10Float | **32** | f11 | f11 | f10 | | |
| Format::RGB9E5Float | **32** | 9 | 9 | 9 | | 5 |

**See also**

TextureDescriptor::format
VertexAttribute::format
RenderingCapabilities::textureFormats
OpenGL counterpart: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/gl↩TexImage1D.xhtml#id-1.6.14.1
Vulkan counterpart `VkFormat`: https://www.khronos.org/registry/vulkan/specs/1.↩1-extensions/man/html/VkFormat.html
Direct3D counterpart `DXGI_FORMAT`: https://msdn.microsoft.com/en-us/library/windows/desktop/b85).aspx
Metal counterpart for textures `MTLPixelFormat`: https://developer.apple.com/documentation/metal/m
Metal counterpart for vertices `MTLVertexFormat`: https://developer.apple.com/documentation/metal/

**Enumerator**

***Undefined*** Undefined format.

***A8UNorm*** Alpha channel format: alpha 8-bit normalized unsigned integer component.

***R8UNorm*** Ordinary color format: red 8-bit normalized unsigned integer component.

***R8SNorm*** Ordinary color format: red 8-bit normalized signed integer component.

***R8UInt*** Ordinary color format: red 8-bit unsigned integer component.

***R8SInt*** Ordinary color format: red 8-bit signed integer component.

***R16UNorm*** Ordinary color format: red 16-bit normalized unsigned interger component.

***R16SNorm*** Ordinary color format: red 16-bit normalized signed interger component.

***R16UInt*** Ordinary color format: red 16-bit unsigned interger component.

***R16SInt*** Ordinary color format: red 16-bit signed interger component.

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

***R32UInt*** Ordinary color format: red 32-bit unsigned interger component.

***R32SInt*** Ordinary color format: red 32-bit signed interger component.

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

***R64Float*** Ordinary color format: red 64-bit floating point component.

**Note**

Only supported with: Vulkan.

***RG8UNorm*** Ordinary color format: red, green 8-bit normalized unsigned integer components.

***RG8SNorm*** Ordinary color format: red, green 8-bit normalized signed integer components.

***RG8UInt*** Ordinary color format: red, green 8-bit unsigned integer components.

***RG8SInt*** Ordinary color format: red, green 8-bit signed integer components.

***RG16UNorm*** Ordinary color format: red, green 16-bit normalized unsigned interger components.

***RG16SNorm*** Ordinary color format: red, green 16-bit normalized signed interger components.

***RG16UInt*** Ordinary color format: red, green 16-bit unsigned interger components.

***RG16SInt*** Ordinary color format: red, green 16-bit signed interger components.

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

***RG32UInt*** Ordinary color format: red, green 32-bit unsigned interger components.

***RG32SInt*** Ordinary color format: red, green 32-bit signed interger components.

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

***RG64Float*** Ordinary color format: red, green 64-bit floating point components.

**Note**

Only supported with: Vulkan.

***RGB8UNorm*** Ordinary color format: red, green, blue 8-bit normalized unsigned integer components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB8UNorm_sRGB*** Ordinary color format: red, green, blue 8-bit normalized unsigned integer components in non-linear sRGB color space.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB8SNorm*** Ordinary color format: red, green, blue 8-bit normalized signed integer components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB8UInt*** Ordinary color format: red, green, blue 8-bit unsigned integer components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB8SInt*** Ordinary color format: red, green, blue 8-bit signed integer components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB16UNorm*** Ordinary color format: red, green, blue 16-bit normalized unsigned interger components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB16SNorm*** Ordinary color format: red, green, blue 16-bit normalized signed interger components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB16UInt*** Ordinary color format: red, green, blue 16-bit unsigned interger components.

**Note**

Only supported with: OpenGL, Vulkan.

***RGB16SInt*** Ordinary color format: red, green, blue 16-bit signed interger components.

**Note**

Only supported with: OpenGL, Vulkan.

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

**Note**

Only supported with: OpenGL, Vulkan.

***RGB32UInt*** Ordinary color format: red, green, blue 32-bit unsigned interger components.

**Note**

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***RGB32SInt*** Ordinary color format: red, green, blue 32-bit signed interger components.

**Note**

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

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

**Note**

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***RGB64Float*** Ordinary color format: red, green, blue 64-bit floating point components.

**Note**

Only supported with: Vulkan.

***RGBA8UNorm*** Ordinary color format: red, green, blue, alpha 8-bit normalized unsigned integer components.

***RGBA8UNorm_sRGB*** Ordinary color format: red, green, blue, alpha 8-bit normalized unsigned integer components in non-linear sRGB color space.

***RGBA8SNorm*** Ordinary color format: red, green, blue, alpha 8-bit normalized signed integer components.

***RGBA8UInt*** Ordinary color format: red, green, blue, alpha 8-bit unsigned integer components.

***RGBA8SInt*** Ordinary color format: red, green, blue, alpha 8-bit signed integer components.

***RGBA16UNorm*** Ordinary color format: red, green, blue, alpha 16-bit normalized unsigned interger components.

***RGBA16SNorm*** Ordinary color format: red, green, blue, alpha 16-bit normalized signed interger components.

***RGBA16UInt*** Ordinary color format: red, green, blue, alpha 16-bit unsigned interger components.

***RGBA16SInt*** Ordinary color format: red, green, blue, alpha 16-bit signed interger components.

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

***RGBA32UInt*** Ordinary color format: red, green, blue, alpha 32-bit unsigned interger components.

***RGBA32SInt*** Ordinary color format: red, green, blue, alpha 32-bit signed interger components.

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

***RGBA64Float*** Ordinary color format: red, green, blue, alpha 64-bit floating point components.

**Note**

Only supported with: Vulkan.

***BGRA8UNorm*** Ordinary color format: blue, green, red, alpha 8-bit normalized unsigned integer components.

***BGRA8UNorm_sRGB*** Ordinary color format: blue, green, red, alpha 8-bit normalized unsigned integer components in non-linear sRGB color space.

***BGRA8SNorm*** Ordinary color format: blue, green, red, alpha 8-bit normalized signed integer components.

**Note**

Only supported with: Vulkan.

***BGRA8UInt*** Ordinary color format: blue, green, red, alpha 8-bit unsigned integer components.

**Note**

Only supported with: Vulkan.

***BGRA8SInt*** Ordinary color format: blue, green, red, alpha 8-bit signed integer components.

**Note**

Only supported with: Vulkan.

***RGB10A2UNorm*** Packed color format: red, green, blue 10-bit and alpha 2-bit normalized unsigned integer components.

***RGB10A2UInt*** Packed color format: red, green, blue 10-bit and alpha 2-bit unsigned integer components.

***RG11B10Float*** Packed color format: red, green 11-bit and blue 10-bit unsigned floating point, i.e. 6-bit mantissa for red and green, 5-bit mantissa for blue, and 5-bit exponent for all components.

***RGB9E5Float*** Packed color format: red, green, blue 9-bit unsigned floating-point with shared 5-bit exponent, i.e. 9-bit mantissa for each component and one 5-bit exponent for all components.

***D16UNorm*** Depth-stencil format: depth 16-bit normalized unsigned integer component.

***D24UNormS8UInt*** Depth-stencil format: depth 24-bit normalized unsigned integer component, and 8-bit unsigned integer stencil component.

***D32Float*** Depth-stencil format: depth 32-bit floating point component.

***D32FloatS8X24UInt*** Depth-stencil format: depth 32-bit floating point component, and 8-bit unsigned integer stencil components (where the remaining 24 bits are unused).

***BC1RGB*** Compressed color format: RGB S3TC DXT1 with 8 bytes per 4x4 block.

**Note**

Only supported with: OpenGL, Vulkan.

***BC1RGBA*** Compressed color format: RGBA S3TC DXT1 with 8 bytes per 4x4 block.

***BC2RGBA*** Compressed color format: RGBA S3TC DXT3 with 16 bytes per 4x4 block.

***BC3RGBA*** Compressed color format: RGBA S3TC DXT5 with 16 bytes per 4x4 block.

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

Image format enumeration that applies to each pixel of an image.

**See also**

SrcImageDescriptor::format
DstImageDescriptor::format
ImageFormatSize

**Todo** Maybe replace `DataType` and `ImageFormat` by LLGL::Format.

**Enumerator**

***Alpha*** Single color component: Alpha.

***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.

***ARGB*** Four color components: Alpha, Red, Green, Blue. Old format, mainly used in Direct3D 9.

***ABGR*** Four color components: Alpha, Blue, Green, Red. Old format, mainly used in Direct3D 9.

***Depth*** Depth component.

***DepthStencil*** Depth component and stencil index.

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

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

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

Input key codes.

**See also**

Window::EventListener

**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***

***Any*** Refers to any key.

#### 9.1.2.15 enum LLGL::LogicOp [strong]

Logical pixel operation enumeration.

**Remarks**

These logical pixel operations are bitwise operations. In the following documentation, `src` denotes the source color and `dst` denotes the destination color.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11.1+, Direct3D 12.0.

**See also**

BlendDescriptor::logicOp

**Enumerator**

***Disabled*** No logical pixel operation.

***Clear*** Resulting operation: `0`.

***Set*** Resulting operation: `1`.

***Copy*** Resulting operation: `src`.

***CopyInverted*** Resulting operation: `~src`.

***NoOp*** Resulting operation: `dst`.

***Invert*** Resulting operation: `~dst`.

***AND*** Resulting operation: `src & dst`.

***ANDReverse*** Resulting operation: `src & ~dst`.

***ANDInverted*** Resulting operation: `~src & dst`.

***NAND*** Resulting operation: `~(src & dst)`.

***OR*** Resulting operation: `src | dst`.

***ORReverse*** Resulting operation: `src | ~dst`.

***ORInverted*** Resulting operation: `~src | dst`.

***NOR*** Resulting operation: `~(src | dst)`.

***XOR*** Resulting operation: `src ^ dst`.

***Equiv*** Resulting operation: `~(src ^ dst)`.

#### 9.1.2.16 enum LLGL::OpenGLContextProfile [strong]

OpenGL context profile enumeration.

**See also**

RendererConfigurationOpenGL::contextProfile

**Enumerator**

***CompatibilityProfile*** OpenGL compatibility profile.

***CoreProfile*** OpenGL core profile.

***ESProfile*** OpenGL ES profile.

**Todo** Not supported yet

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

Polygon filling modes enumeration.

**See also**

RasterizerDescriptor::polygonMode

**Enumerator**

***Fill*** Draw filled polygon.

***Wireframe*** Draw triangle edges only.

***Points*** Draw vertex points only.

**Note**

Only supported with: OpenGL, Vulkan.

#### 9.1.2.18 enum LLGL::PrimitiveTopology [strong]

Primitive topology enumeration.

**See also**

GraphicsPipelineDescriptor::primitiveTopology

**Enumerator**

***PointList*** Point list, where each vertex represents a single point primitive.

***LineList*** Line list, where each pair of two vertices represetns a single line primitive.

***LineStrip*** Line strip, where each vertex generates a new line primitive while the previous vertex is used as line start.

***LineLoop*** Line loop, which is similar to LineStrip but the first and last vertices generate yet another line primitive.

**Note**

Only supported with: OpenGL.

***LineListAdjacency*** Adjacency line list, which is similar to LineList but each end point has a corresponding adjacent vertex that is accessible in a geometry shader.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***LineStripAdjacency*** Adjacency line strip, which is similar to LineStrip but each end point has a corresponding adjacent vertex that is accessible in a geometry shader.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***TriangleList*** Triangle list, where each set of three vertices represent a single triangle primitive.

***TriangleStrip*** Triangle strip, where each vertex generates a new triangle primitive with an alternative triangle winding.

***TriangleFan*** Triangle fan, where each vertex generates a new triangle primitive while all share the same first vertex.

**Note**

Only supported with: OpenGL, Vulkan.

***TriangleListAdjacency*** Adjacency triangle list, which is similar to TriangleList but each triangle edge has a corresponding adjacent vertex that is accessible in a geometry shader.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***TriangleStripAdjacency*** Adjacency triangle strips which is similar to TriangleStrip but each triangle edge has a corresponding adjacent vertex that is accessible in a geometry shader.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

***Patches1*** Patches with 1 control point that is accessible in a tessellation shader.

***Patches2*** Patches with 2 control points that are accessible in a tessellation shader.

***Patches3*** Patches with 3 control points that are accessible in a tessellation shader.

***Patches4*** Patches with 4 control points that are accessible in a tessellation shader.

***Patches5*** Patches with 5 control points that are accessible in a tessellation shader.

***Patches6*** Patches with 6 control points that are accessible in a tessellation shader.

***Patches7*** Patches with 7 control points that are accessible in a tessellation shader.

***Patches8*** Patches with 8 control points that are accessible in a tessellation shader.

***Patches9*** Patches with 9 control points that are accessible in a tessellation shader.

***Patches10*** Patches with 10 control points that are accessible in a tessellation shader.

***Patches11*** Patches with 11 control points that are accessible in a tessellation shader.

***Patches12*** Patches with 12 control points that are accessible in a tessellation shader.

***Patches13*** Patches with 13 control points that are accessible in a tessellation shader.

***Patches14*** Patches with 14 control points that are accessible in a tessellation shader.

***Patches15*** Patches with 15 control points that are accessible in a tessellation shader.

***Patches16*** Patches with 16 control points that are accessible in a tessellation shader.

***Patches17*** Patches with 17 control points that are accessible in a tessellation shader.

***Patches18*** Patches with 18 control points that are accessible in a tessellation shader.

***Patches19*** Patches with 19 control points that are accessible in a tessellation shader.

***Patches20*** Patches with 20 control points that are accessible in a tessellation shader.

***Patches21*** Patches with 21 control points that are accessible in a tessellation shader.

***Patches22*** Patches with 22 control points that are accessible in a tessellation shader.

***Patches23*** Patches with 23 control points that are accessible in a tessellation shader.

***Patches24*** Patches with 24 control points that are accessible in a tessellation shader.

***Patches25*** Patches with 25 control points that are accessible in a tessellation shader.

***Patches26*** Patches with 26 control points that are accessible in a tessellation shader.

***Patches27*** Patches with 27 control points that are accessible in a tessellation shader.

***Patches28*** Patches with 28 control points that are accessible in a tessellation shader.

***Patches29*** Patches with 29 control points that are accessible in a tessellation shader.

***Patches30*** Patches with 30 control points that are accessible in a tessellation shader.

***Patches31*** Patches with 31 control points that are accessible in a tessellation shader.

***Patches32*** Patches with 32 control points that are accessible in a tessellation shader.

#### 9.1.2.19 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.

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

Query type enumeration.

**See also**

QueryHeapDescriptor::type

**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.

***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.

***PipelineStatistics*** Pipeline statistics such as number of shader invocations, generated primitives, etc.

**See also**

QueryPipelineStatistics
RenderingFeatures::hasPipelineStatistics

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

Render condition mode enumeration.

**Remarks**

The condition is determined by the type of the QueryHeap 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.

#### 9.1.2.22 enum LLGL::ResourceType [strong]

Hardware resource type enumeration.

**Remarks**

This is primarily used to describe the source type for a layout binding (see BindingDescriptor), which is why all buffer types are enumerated but not the texture types.

**See also**

BindingDescriptor::type

**Enumerator**

***Undefined*** Undefined resource type.

***Buffer*** Buffer resource.

**See also**

Buffer
RenderSystem::CreateBuffer

***Texture*** Texture resource.

**See also**

Texture
TextureType
RenderSystem::CreateTexture

***Sampler*** Sampler state resource.

**See also**

Sampler
RenderSystem::CreateSampler

#### 9.1.2.23 enum LLGL::SamplerAddressMode [strong]

Technique for resolving texture coordinates that are outside of the range [0, 1].

**See also**

SamplerDescriptor::addressModeU
SamplerDescriptor::addressModeV
SamplerDescriptor::addressModeW

**Enumerator**

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

**Figure 9.1 SamplerAddressMode::Repeat example**

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

**Figure 9.2 SamplerAddressMode::Mirror example**

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

**Figure 9.3 SamplerAddressMode::Clamp example**

***Border*** Sample border color for texture coordinates that are outside the interval [0, 1].

**Figure 9.4 SamplerAddressMode::Border example**

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

**Figure 9.5 SamplerAddressMode::MirrorOnce example**

#### 9.1.2.24 enum LLGL::SamplerFilter [strong]

Sampling filter enumeration.

**See also**

SamplerDescriptor::minFilter
SamplerDescriptor::magFilter
SamplerDescriptor::mipMapFilter
Image::Resize(const Extent3D&, const SamplerFilter)

**Enumerator**

***Nearest*** Take the nearest texture sample.

**Figure 9.6 SamplerFilter::Nearest example**

***Linear*** Interpolate between multiple texture samples.

**Figure 9.7 SamplerFilter::Linear example**

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

Screen coordinate system origin enumeration.

**See also**

RenderingCapabilities::screenOrigin

**Enumerator**

***LowerLeft*** Specifies a screen origin in the lower-left.

**Note**

Native screen origin in: OpenGL.

***UpperLeft*** Specifies a screen origin in the upper-left.

**Note**

Native screen origin in: Direct3D 11, Direct3D 12, Vulkan.

#### 9.1.2.26 enum LLGL::ShaderSourceType [strong]

Shader source type enumeration.

**See also**

ShaderDescriptor::sourceType
ShaderDescriptor::sourceSize

**Enumerator**

***CodeString*** Refers to `sourceSize+1` bytes, describing shader high-level code (including null terminator).

***CodeFile*** Refers to `sourceSize+1` bytes, describing the filename of the shader high-level code (including null terminator).

***BinaryBuffer*** Refers to `sourceSize` bytes, describing shader binary code.

***BinaryFile*** Refers to `sourceSize+1` bytes, describing the filename of the shader binary code (including null terminator).

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

Shader type enumeration.

**See also**

ShaderDescriptor::type

**Enumerator**

***Undefined*** Undefined shader type.

***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.

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

Shading language version enumation.

**Remarks**

These enumeration entries can be casted to an integer using the bitmask ShadingLanguage::VersionBitmask to get the respective version number:

```
// 'versionNo' will have the value 330
static const auto versionGLSL330 = static_cast<std::uint32_t>(
      LLGL::ShadingLanguage::GLSL_330);
static const auto versionBitmask = static_cast<std::uint32_t>(
      LLGL::ShadingLanguage::VersionBitmask);
static const auto versionNo      = versionGLSL330 & versionBitmask;
```

**Enumerator**

***GLSL*** GLSL (OpenGL Shading Language).

***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).

***GLSL_460*** GLSL 4.60 (since OpenGL 4.6).

***ESSL*** ESSL (OpenGL ES Shading Language).

***ESSL_100*** ESSL 1.00 (since OpenGL ES 2.0).

***ESSL_300*** ESSL 3.00 (since OpenGL ES 3.0).

***ESSL_310*** ESSL 3.10 (since OpenGL ES 3.1).

***ESSL_320*** ESSL 3.20 (since OpenGL ES 3.2).

***HLSL*** HLSL (High Level Shading Language).

***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).

***HLSL_5_1*** HLSL 5.1 (since Direct3D 12 and Direct3D 11.3).

***Metal*** Metal Shading Language.

***Metal_1_0*** Metal 1.0 (since iOS 8.0).

***Metal_1_1*** Metal 1.1 (since iOS 9.0 and OS X 10.11).

***Metal_1_2*** Metal 1.2 (since iOS 10.0 and macOS 10.12).

***Metal_2_0*** Metal 2.0 (since iOS 11.0 and macOS 10.13).

***Metal_2_1*** Metal 2.1 (since iOS 12.0 and macOS 10.14).

***SPIRV*** SPIR-V Shading Language.

***SPIRV_100*** SPIR-V 1.0.

***VersionBitmask*** Bitmask for the version number of each shading language enumeration entry.

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

Stencil operations enumeration.

**See also**

StencilFaceDescriptor

**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.

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

Storage buffer type enumeration.

**Note**

Only supported with: Direct3D 11, Direct3D 12.

**See also**

BufferDescriptor::StorageBuffer::storageType

**Todo** Replace by BindFlag with different options (e.g. StructuredBuffer and Storage).

**Enumerator**

***Undefined*** Undefined storage buffer type.

***Buffer*** Typed buffer.

***StructuredBuffer*** Structured buffer.

**Note**

This cannot be used together with a buffer that has the binding flag BindFlags::IndirectBuffer.

***ByteAddressBuffer*** Byte-address buffer.

***RWBuffer*** Typed read/write buffer.

***RWStructuredBuffer*** Structured read/write buffer.

**Note**

This cannot be used together with a buffer that has the binding flag BindFlags::IndirectBuffer.

***RWByteAddressBuffer*** Byte-address read/write buffer.

***AppendStructuredBuffer*** Append structured buffer.

**Note**

This cannot be used together with a buffer that has the binding flag BindFlags::IndirectBuffer.

***ConsumeStructuredBuffer*** Consume structured buffer.

**Note**

This cannot be used together with a buffer that has the binding flag BindFlags::IndirectBuffer.

#### 9.1.2.31 enum LLGL::SystemValue [strong]

Enumeration of system values for vertex input or fragment output attributes.

**Remarks**

This is only used for shader code reflection.
Here is an overview of all supported system values and their mapping to the respective shading language:

**Table 9.2 System value mapping**

| Type | HLSL | GLSL | SPIR-V | Metal |
|---|---|---|---|---|
| SystemValue::ClipDistance | SV_Clip Distance | gl_Clip Distance | gl_Clip Distance | [[clip_ distance]] |
| SystemValue::Color | SV_Target | N/A | N/A | [[color(N)]] |
| SystemValue::CullDistance | SV_Cull Distance | gl_Cull Distance | gl_Cull Distance | N/A |
| SystemValue::Depth | SV_Depth | gl_FragDepth | gl_FragDepth | [[depth(any)]] |
| SystemValue::DepthGreater | SV_Depth GreaterEqual | N/A | N/A | [[depth(greater)]] |
| SystemValue::DepthLess | SV_Depth LessEqual | N/A | N/A | [[depth(less)]] |
| SystemValue::FrontFacing | SV_IsFront Face | gl_Front Facing | gl_Front Facing | [[front_ facing]] |
| SystemValue::InstanceID | SV_ InstanceID | gl_ InstanceID | gl_ Instance Index | [[instance _id] |
| SystemValue::Position | SV_Position | gl_Position, gl_FragCoord | gl_Position, gl_FragCoord | [[position]] |
| SystemValue::PrimitiveID | SV_ PrimitiveID | gl_ PrimitiveID | gl_ PrimitiveID | N/A |
| SystemValue::RenderTarget Index | SV_Render Target ArrayIndex | gl_Layer | gl_Layer | [[render _target _array_ index]] |
| SystemValue::SampleMask | SV_Coverage | gl_Sample Mask | gl_Sample Mask | [[sample_ mask]] |
| SystemValue::SampleID | SV_Sample Index | gl_SampleID | gl_SampleID | [[sample_ id]] |
| SystemValue::Stencil | SV_Stencil Ref | N/A | N/A | [[stencil]] |
| SystemValue::VertexID | SV_VertexID | gl_VertexID | gl_Vertex Index | [[vertex_ id]] |
| SystemValue::ViewportIndex | SV_ Viewport ArrayIndex | gl_ Viewport Index | gl_ Viewport Index | [[viewport _array_ index]] |

**See also**

VertexAttribute::systemValue
FragmentAttribute::systemValue

**Enumerator**

***Undefined*** Undefined system value.

***ClipDistance*** Forward-compatible mechanism for vertex clipping.

***Color*** Fragment output color value.

***CullDistance*** Mechanism for controlling user culling.

***Depth*** Fragment depth value.

***DepthGreater*** Fragment depth value that is greater than or equal to the previous one.

***DepthLess*** Fragment depth value that is less than or equal to the previous one.

***FrontFacing*** Indicates whether a primitive is front or back facing.

***InstanceID*** Index of the input instance.

**Note**

This value behalves differently between Direct3D and OpenGL.

**See also**

CommandBuffer::DrawInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::uint32_t)

***Position*** Vertex or fragment position.

***PrimitiveID*** Index of the geometry primitive.

***RenderTargetIndex*** Index of the render target layer.

***SampleMask*** Sample coverage mask.

***SampleID*** Index of the input sample.

***Stencil*** Fragment stencil value.

**Note**

Only supported with: Direct3D 11.3, Direct3D 12, Metal.

***VertexID*** Index of the input vertex.

**Note**

This value behalves differently between Direct3D and OpenGL.

**See also**

CommandBuffer::Draw

***ViewportIndex*** Index of the viewport array.

#### 9.1.2.32 enum LLGL::TextureSwizzle [strong]

Texture component swizzle enumeration.

**Remarks**

Can be used to change the order of texel components independently of a shader.

**See also**

TextureSwizzleRGBA

**Enumerator**

***Zero*** The component is replaced by the constant zero.

***One*** The component is replaced by the constant one.

***Red*** The component is replaced by red component.

***Green*** The component is replaced by green component.

***Blue*** The component is replaced by blue component.

***Alpha*** The component is replaced by alpha component.

#### 9.1.2.33 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.

#### 9.1.2.34 enum LLGL::UniformType [strong]

Shader uniform type enumeration.

**Remarks**

Because "Bool" is a reserved identifier for an Xlib macro on GNU/Linux, all scalar types also have a component index (e.g. "Bool1" instead of "Bool").

**Enumerator**

***Undefined*** Undefined uniform type.

***Float1*** float uniform.

***Float2*** float2/ vec2 uniform.

***Float3*** float3/ vec3 uniform.

***Float4*** float4/ vec4 uniform.

***Double1*** double uniform.

***Double2*** double2/ dvec2 uniform.

***Double3*** double3/ dvec3 uniform.

***Double4*** double4/ dvec4 uniform.

***Int1*** int uniform.

***Int2*** int2/ ivec2 uniform.

***Int3*** int3/ ivec3 uniform.

***Int4*** int4/ ivec4 uniform.

***UInt1*** uint uniform.

***UInt2*** uint2/ uvec2 uniform.

***UInt3*** uint3/ uvec3 uniform.

***UInt4*** uint4/ uvec4 uniform.

***Bool1*** bool uniform.

***Bool2*** bool2/ bvec2 uniform.

***Bool3*** bool3/ bvec3 uniform.

***Bool4*** bool4/ bvec4 uniform.

***Float2x2*** float2x2/ mat2 uniform.

***Float2x3*** float2x3/ mat2x3 uniform.

***Float2x4*** float2x4/ mat2x4 uniform.

***Float3x2*** float3x2/ mat3x2 uniform.

***Float3x3*** float3x3/ mat3 uniform.

***Float3x4*** float3x4/ mat3x4 uniform.

***Float4x2*** float4x2/ mat4x2 uniform.

***Float4x3*** float4x3/ mat4x3 uniform.

***Float4x4*** float4x4/ mat4 uniform.

***Double2x2*** double2x2/ dmat2 uniform.

***Double2x3*** double2x3/ dmat2x3 uniform.

***Double2x4*** double2x4/ dmat2x4 uniform.

***Double3x2*** double3x2/ dmat3x2 uniform.

***Double3x3*** double3x3/ dmat3 uniform.

***Double3x4*** double3x4/ dmat3x4 uniform.

***Double4x2*** double4x2/ dmat4x2 uniform.

***Double4x3*** double4x3/ dmat4x3 uniform.

***Double4x4*** double4x4/ dmat4 uniform.

***Sampler*** Sampler uniform (e.g. "sampler2D").

***Image*** Image uniform (e.g. "image2D").

***AtomicCounter*** Atomic counter uniform (e.g. "atomic_uint").

#### 9.1.2.35 enum LLGL::WarningType [strong]

Rendering debugger warning types enumeration.

**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).

***VaryingBehavior*** Warning due to a varying behavior between the native APIs (e.g. `SV_VertexID` in HLSL behaves different to `gl_VertexID` in GLSL or `gl_VertexIndex` in SPIRV).

### 9.1.3 Function Documentation

#### 9.1.3.1 template<typename Dst , typename Src > Dst LLGL::CastColorValue ( const Src & *value* ) [inline]

Casts the specified color value and transforms it from the source data type range to the destination data type range.

**See also**

MaxColorValue

#### 9.1.3.2 template<> bool LLGL::CastColorValue< bool, bool > ( const bool & *value* ) [inline]

Specialized template which merely passes the input value as output.

#### 9.1.3.3 template<> double LLGL::CastColorValue< double, double > ( const double & *value* ) [inline]

Specialized template which merely passes the input value as output.

#### 9.1.3.4 template<> float LLGL::CastColorValue< float, float > ( const float & *value* ) [inline]

Specialized template which merely passes the input value as output.

#### 9.1.3.5 template<> std::uint8_t LLGL::CastColorValue< std::uint8_t, std::uint8_t > ( const std::uint8_t & *value* ) [inline]

Specialized template which merely passes the input value as output.

#### 9.1.3.6 template<typename T > const T* LLGL::CastTo ( const Interface * *obj* ) [inline]

Returns a constant pointer to this instance of type `const T*` if it implements the specified interface.

**Remarks**

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
const LLGL::Surface* mySurface = ...
if (const LLGL::Window* myWindow = LLGL::CastTo<LLGL::Window>(mySurface))
{
    ...
}
```

#### 9.1.3.7 template<typename T > T* LLGL::CastTo ( Interface * *obj* ) [inline]

Returns a pointer to this instance of type `T*` if it implements the specified interface.

**Remarks**

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
LLGL::Surface* mySurface = ...
if (LLGL::Window* myWindow = LLGL::CastTo<LLGL::Window>(mySurface))
{
    ...
}
```

#### 9.1.3.8 template<typename T > const T& LLGL::CastTo ( const Interface & *obj* ) [inline]

Returns a constant reference to this instance of type `const T&` if it implements the specified interface.

**Remarks**

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
const LLGL::Surface& mySurface = ...
const LLGL::Window& myWindow = LLGL::CastTo<LLGL::Window>(mySurface);
```

#### 9.1.3.9 template<typename T > T& LLGL::CastTo ( Interface & *obj* ) [inline]

Returns a reference to this instance of type `T&` if it implements the specified interface.

**Remarks**

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
LLGL::Surface& mySurface = ...
LLGL::Window& myWindow = LLGL::CastTo<LLGL::Window&>(mySurface);
```

#### 9.1.3.10 LLGL_EXPORT Extent2D LLGL::GetExtentRatio ( const Extent2D & *extent* )

Returns the ratio of the specified extent as another extent, i.e. all attributes are divided by their greatest common divisor.

**Remarks**

This can be used to print out a display mode resolution in a better format (e.g. "16:9" rather than "1920:1080").

**See also**

DisplayModeDescriptor::resolution

#### 9.1.3.11 LLGL_EXPORT std::uint32_t LLGL::GetPrimitiveTopologyPatchSize ( const PrimitiveTopology *primitiveTopology* )

Returns the number of patch control points of the specified primitive topology (in range [1, 32]), or 0 if the topology is not a patch list.

#### 9.1.3.12 LLGL_EXPORT bool LLGL::IsPrimitiveTopologyPatches ( const PrimitiveTopology *primitiveTopology* )

Returns true if the specified primitive topology is a patch list.

#### 9.1.3.13 LLGL_EXPORT bool LLGL::IsShaderSourceBinary ( const ShaderSourceType *type* )

Returns true if the specified shader source type is either ShaderSourceType::BinaryBuffer or ShaderSourceType↩ ::BinaryFile.

**See also**

ShaderSourceType

#### 9.1.3.14 LLGL_EXPORT bool LLGL::IsShaderSourceCode ( const ShaderSourceType *type* )

Returns true if the specified shader source type is either ShaderSourceType::CodeString or ShaderSourceType::↩ CodeFile.

**See also**

ShaderSourceType

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

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

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

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

#### 9.1.3.17 template<> std::uint8_t LLGL::MaxColorValue< std::uint8_t > ( ) [inline]

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

#### 9.1.3.18 LLGL_EXPORT bool LLGL::operator!= ( const DisplayModeDescriptor & *lhs*, const DisplayModeDescriptor & *rhs* )

Compares the two specified display mode descriptors on inequality.

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

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

Compares the two specified V-sync descriptors on inequality.

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

Compares the two specified video mode descriptors on inequality.

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

Compares the two VertexAttribute types for inequality (including their names and all other members).

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

Returns true if any component of both colors 'lhs' and 'rhs' are unequal.

**Remarks**

The comparison uses the 'operator ==' of the underlying component type. Note that this comparison is quite limited for floating-point types, due to precision issues.

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

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

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

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

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

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

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

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

#### 9.1.3.32 LLGL_EXPORT bool LLGL::operator== ( const DisplayModeDescriptor & *lhs*, const DisplayModeDescriptor & *rhs* )

Compares the two specified display mode descriptors on equality.

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

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

Compares the two specified V-sync descriptors on equality.

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

Compares the two specified video mode descriptors on equality.

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

Compares the two VertexAttribute types for equality (including their names and all other members).

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

Returns true if all components of both colors 'lhs' and 'rhs' are equal.

**Remarks**

The comparison uses the 'operator ==' of the underlying component type. Note that this comparison is quite limited for floating-point types, due to precision issues.

#### 9.1.3.38 LLGL_EXPORT bool LLGL::ValidateRenderingCaps ( const RenderingCapabilities & *presentCaps*, const RenderingCapabilities & *requiredCaps*, const ValidateRenderingCapsFunc & *callback* = { } )

Validates the presence of the specified required rendering capabilities.

**Parameters**

| | | |
|---|---|---|
| in | *presentCaps* | Specifies the rendering capabilities that are present for a certain renderer. |
| in | *requiredCaps* | Specifies the rendering capabilities that are required for the host application to work properly. |
| in | *callback* | Optional callback to retrieve information about the attributes that did not fulfill the requirement. If this is null the validation process breaks with the first attribute that did not fulfill the requirement. By default null. |

**Returns**

True on success, otherwise at least one attribute did not fulfill the requirement.

#### Remarks

Here is an example usage to print out all attributes that did not fulfill the requirement:

```
// Initialize the requirements
LLGL::RenderingCapabilities myRequirements;
myRequirements.features.hasStorageBuffers               = true;
myRequirements.features.hasComputeShaders               = true;
myRequirements.limits.maxComputeShaderWorkGroups[0]     = 1024;
myRequirements.limits.maxComputeShaderWorkGroups[1]     = 1024;
myRequirements.limits.maxComputeShaderWorkGroups[2]     = 1;
myRequirements.limits.maxComputeShaderWorkGroupSize[0]  = 8;
myRequirements.limits.maxComputeShaderWorkGroupSize[1]  = 8;
myRequirements.limits.maxComputeShaderWorkGroupSize[2]  = 8;

// Validate rendering capabilities supported by the render system
LLGL::ValidateRenderingCaps(
    myRenderer->GetRenderingCaps(),
    myRequirements,
    [](const std::string& info, const std::string& attrib) {
        std::cerr << info << ": " << attrib << std::endl;
        return true;
    }
);
```

#### Note

The following attributes of the RenderingCapabilities structure are ignored: 'screenOrigin' and 'clippingRange'.

#### See also

RenderingCapabilities
ValidateRenderingCapsFunc

## 9.2 LLGL::Constants Namespace Reference

Namespace with all constants used as default arguments.

### 9.2.1 Detailed Description

Namespace with all constants used as default arguments.

## 9.3 LLGL::Log Namespace Reference

### Typedefs

- using ReportCallback = std::function< void(ReportType type, const std::string &message, const std::string &contextInfo, void *userData)>

  *Report callback function signature.*

### Enumerations

- enum ReportType { ReportType::Error, ReportType::Warning, ReportType::Information, ReportType::Performance }

  *Report type enumeration.*

## Functions

- LLGL_EXPORT void PostReport (ReportType type, const std::string &message, const std::string &contextInfo="")
- LLGL_EXPORT void SetReportCallback (const ReportCallback &callback, void *userData=nullptr)

  *Sets the new report callback. No report callback is specified by default, in which case the reports are ignored.*
- LLGL_EXPORT void SetReportCallbackStd (std::ostream &stream=std::cerr)

  *Sets the new report callback to the standard output streams.*
- LLGL_EXPORT void SetReportLimit (std::size_t maxCount)

  *Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.*

### 9.3.1 Typedef Documentation

#### 9.3.1.1 using LLGL::Log::ReportCallback = typedef std::function<void(ReportType type, const std::string& message, const std::string& contextInfo, void* userData)>

Report callback function signature.

**Parameters**

| | | |
|---|---|---|
| in | *type* | Specifies the type of the report message. |
| in | *message* | Specifies the report message. |
| in | *contextInfo* | Specifies a descriptive string about the context of the report (e.g. "in 'LLGL::RenderSystem::CreateShader'"). This may also be empty. |
| in | *userData* | Specifies the user data that was set in the previous call to SetReportCallback. |

**See also**

ReportType
SetReportCallback

### 9.3.2 Enumeration Type Documentation

#### 9.3.2.1 enum LLGL::Log::ReportType [strong]

Report type enumeration.

**See also**

ReportCallback

**Enumerator**

***Error*** Error message type.

**Remarks**

For example, when a feature is used that is not supported.

***Warning*** Warning message type.

**Remarks**

For example, when an operation has no effect like submitting a draw command with zero vertices.

***Information*** Information message type.

**Remarks**

For example, when a multi-sampling format is not supported so it's set to a lower quality than it was specified.

***Performance*** Performance penelty message type.

**Remarks**

For example, when unnecessary clear commands are submitted.

### 9.3.3 Function Documentation

#### 9.3.3.1 LLGL_EXPORT void LLGL::Log::PostReport ( ReportType *type*, const std::string & *message*, const std::string & *contextInfo* = " " )

Posts a report to the currently set report callback.

**See also**

ReportCallback

#### 9.3.3.2 LLGL_EXPORT void LLGL::Log::SetReportCallback ( const ReportCallback & *callback*, void * *userData* = `nullptr` )

Sets the new report callback. No report callback is specified by default, in which case the reports are ignored.

**Parameters**

| | | |
|---|---|---|
| in | *callback* | Specifies the new report callback. This can also be null. |
| in | *userData* | Optional raw pointer to some user data that will be passed to the callback each time a report is generated. |

**Remarks**

The reports can be generated in a multi-threaded environment. Even this function can be called on multiple threads. The functionality of the entire Log namespace is synchronized by LLGL. Use SetReportCallbackStd to forward the reports to the standard C++ I/O streams.

**See also**

PostReport
SetReportCallbackStd

#### 9.3.3.3 LLGL_EXPORT void LLGL::Log::SetReportCallbackStd ( std::ostream & *stream* = `std::cerr` )

Sets the new report callback to the standard output streams.

**Parameters**

| | | |
|---|---|---|
| in | *stream* | Specifies the output stream. By default `std::cerr`. |

**See also**

SetReportCallback

#### 9.3.3.4 LLGL_EXPORT void LLGL::Log::SetReportLimit ( std::size_t *maxCount* )

Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.

**Parameters**

| | | |
|---|---|---|
| in | *maxCount* | Specifies the maximum number of reports. If this is 0, there is affectively no limit. By default 0. |

## 9.4 LLGL::Version Namespace Reference

Namespace with functions to determine LLGL version.

### Functions

- LLGL_EXPORT std::uint32_t GetMajor ()

  *Returns the major LLGL version (e.g. 1 stands for "1.00").*
- LLGL_EXPORT std::uint32_t GetMinor ()

  *Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.*
- LLGL_EXPORT std::uint32_t 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 std::uint32_t 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)").*

### 9.4.1 Detailed Description

Namespace with functions to determine LLGL version.

### 9.4.2 Function Documentation

#### 9.4.2.1 LLGL_EXPORT std::uint32_t LLGL::Version::GetID ( )

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

#### 9.4.2.2 LLGL_EXPORT std::uint32_t LLGL::Version::GetMajor ( )

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

#### 9.4.2.3 LLGL_EXPORT std::uint32_t LLGL::Version::GetMinor ( )

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

#### 9.4.2.4 LLGL_EXPORT std::uint32_t LLGL::Version::GetRevision ( )

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

#### 9.4.2.5 LLGL_EXPORT std::string LLGL::Version::GetStatus ( )

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

#### 9.4.2.6 LLGL_EXPORT std::string LLGL::Version::GetString ( )

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

# Chapter 10

# Class Documentation

## 10.1 LLGL::ApplicationDescriptor Struct Reference

Application descriptor structure.

```
#include <RendererConfiguration.h>
```

### Public Attributes

- std::string applicationName

  *Descriptive string of the application.*
- std::uint32_t applicationVersion

  *Version number of the application.*
- std::string engineName

  *Descriptive string of the engine or middleware.*
- std::uint32_t engineVersion

  *Version number of the engine or middleware.*

### 10.1.1 Detailed Description

Application descriptor structure.

**Note**

Only supported with: Vulkan.

**See also**

RendererConfigurationVulkan::application

### 10.1.2 Member Data Documentation

#### 10.1.2.1 std::string LLGL::ApplicationDescriptor::applicationName

Descriptive string of the application.

#### 10.1.2.2 std::uint32_t LLGL::ApplicationDescriptor::applicationVersion

Version number of the application.

#### 10.1.2.3 std::string LLGL::ApplicationDescriptor::engineName

Descriptive string of the engine or middleware.

#### 10.1.2.4 std::uint32_t LLGL::ApplicationDescriptor::engineVersion

Version number of the engine or middleware.

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

- RendererConfiguration.h

## 10.2 LLGL::AttachmentClear Struct Reference

Attachment clear command structure.

```
#include <CommandBufferFlags.h>
```

### Public Member Functions

- AttachmentClear ()=default
- AttachmentClear (const AttachmentClear &)=default
- AttachmentClear & operator= (const AttachmentClear &)=default
- AttachmentClear (const ColorRGBAf &color, std::uint32_t colorAttachment)

  *Constructor for a color attachment clear command.*
- AttachmentClear (float depth)

  *Constructor for a depth attachment clear command.*
- AttachmentClear (std::uint32_t stencil)

  *Constructor for a stencil attachment clear command.*
- AttachmentClear (float depth, std::uint32_t stencil)

  *Constructor for a depth-stencil attachment clear command.*

### Public Attributes

- long flags = 0

  *Specifies the clear buffer flags.*
- std::uint32_t colorAttachment = 0

  *Specifies the index of the color attachment within the active render target. By default 0.*
- ClearValue clearValue

  *Clear value for color, depth, and stencil buffers.*

### 10.2.1 Detailed Description

Attachment clear command structure.

**See also**

CommandBuffer::ClearAttachments

### 10.2.2 Constructor & Destructor Documentation

#### 10.2.2.1 LLGL::AttachmentClear::AttachmentClear ( ) [default]

#### 10.2.2.2 LLGL::AttachmentClear::AttachmentClear ( const AttachmentClear & ) [default]

#### 10.2.2.3 LLGL::AttachmentClear::AttachmentClear ( const ColorRGBAf & *color*, std::uint32_t *colorAttachment* ) [inline]

Constructor for a color attachment clear command.

#### 10.2.2.4 LLGL::AttachmentClear::AttachmentClear ( float *depth* ) [inline]

Constructor for a depth attachment clear command.

#### 10.2.2.5 LLGL::AttachmentClear::AttachmentClear ( std::uint32_t *stencil* ) [inline]

Constructor for a stencil attachment clear command.

#### 10.2.2.6 LLGL::AttachmentClear::AttachmentClear ( float *depth*, std::uint32_t *stencil* ) [inline]

Constructor for a depth-stencil attachment clear command.

### 10.2.3 Member Function Documentation

#### 10.2.3.1 AttachmentClear& LLGL::AttachmentClear::operator= ( const AttachmentClear & ) [default]

### 10.2.4 Member Data Documentation

#### 10.2.4.1 ClearValue LLGL::AttachmentClear::clearValue

Clear value for color, depth, and stencil buffers.

#### 10.2.4.2 std::uint32_t LLGL::AttachmentClear::colorAttachment = 0

Specifies the index of the color attachment within the active render target. By default 0.

**Remarks**

This is ignored if the ClearFlags::Color bit is not set in the 'flags' member.

**See also**

flags

#### 10.2.4.3 long LLGL::AttachmentClear::flags = 0

Specifies the clear buffer flags.

**Remarks**

This can be a bitwise OR combination of the "ClearFlags" enumeration entries. However, if the ClearFlags↩ ::Color bit is set, all other bits are ignored. It is recommended to clear depth- and stencil buffers always simultaneously if both are meant to be cleared (i.e. use ClearFlags::DepthStencil in this case).

**See also**

ClearFlags

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

- CommandBufferFlags.h

## 10.3 LLGL::AttachmentDescriptor Struct Reference

Render target attachment descriptor structure.

```
#include <RenderTargetFlags.h>
```

### Public Member Functions

- AttachmentDescriptor ()=default
- AttachmentDescriptor (const AttachmentDescriptor &)=default
- AttachmentDescriptor (AttachmentType type)

  *Constructor for the specified depth-, or stencil attachment.*
- AttachmentDescriptor (AttachmentType type, Texture *texture, std::uint32_t mipLevel=0, std::uint32_t array↩ Layer=0)

  *Constructor for the specified depth-, stencil-, or color attachment.*

### Public Attributes

- AttachmentType type = AttachmentType::Color

  *Specifies for which output information the texture attachment is to be used, e.g. for color or depth information. By default AttachmentType::Color.*
- Texture * texture = nullptr

  *Pointer to the texture which is to be used as target output. By default null.*
- std::uint32_t mipLevel = 0

  *Specifies the MIP-map level which is to be attached to a render target.*
- std::uint32_t arrayLayer = 0

  *Specifies the array texture layer which is to be used as render target attachment.*

## 10.3.1 Detailed Description

Render target attachment descriptor structure.

**See also**

RenderTargetDescriptor

## 10.3.2 Constructor & Destructor Documentation

### 10.3.2.1 LLGL::AttachmentDescriptor::AttachmentDescriptor ( ) [default]

### 10.3.2.2 LLGL::AttachmentDescriptor::AttachmentDescriptor ( const AttachmentDescriptor & ) [default]

### 10.3.2.3 LLGL::AttachmentDescriptor::AttachmentDescriptor ( AttachmentType *type* ) [inline]

Constructor for the specified depth-, or stencil attachment.

### 10.3.2.4 LLGL::AttachmentDescriptor::AttachmentDescriptor ( AttachmentType *type*, Texture * *texture*, std::uint32_t *mipLevel* = 0, std::uint32_t *arrayLayer* = 0 ) [inline]

Constructor for the specified depth-, stencil-, or color attachment.

## 10.3.3 Member Data Documentation

### 10.3.3.1 std::uint32_t LLGL::AttachmentDescriptor::arrayLayer = 0

Specifies the array texture layer which is to be used as render target attachment.

**Remarks**

This is only used for array textures and cube textures (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCube, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray).
For cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray), each cube has its own 6 array layers. The layer index for the respective cube faces is described at the TextureDescriptor::arrayLayers member.
If `texture` is null, this attribute is ignored.

**See also**

TextureDescriptor::arrayLayers

#### 10.3.3.2 std::uint32_t LLGL::AttachmentDescriptor::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).
If `texture` is null, this attribute is ignored.

#### 10.3.3.3 Texture* LLGL::AttachmentDescriptor::texture = nullptr

Pointer to the texture which is to be used as target output. By default null.

**Remarks**

If this is null, the attribute `type` **must not** be AttachmentType::Color. The texture must also have been created either with the binding flag BindFlags::ColorAttachment or BindFlags::DepthStencilAttachment.

**See also**

AttachmentDescriptor::type
TextureDescriptor::bindFlags

#### 10.3.3.4 AttachmentType LLGL::AttachmentDescriptor::type = AttachmentType::Color

Specifies for which output information the texture attachment is to be used, e.g. for color or depth information. By default AttachmentType::Color.

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

- RenderTargetFlags.h

## 10.4 LLGL::AttachmentFormatDescriptor Struct Reference

Render target attachment descriptor structure.

```
#include <RenderPassFlags.h>
```

### Public Member Functions

- AttachmentFormatDescriptor ()=default
- AttachmentFormatDescriptor (const AttachmentFormatDescriptor &)=default
- AttachmentFormatDescriptor (const Format format, const AttachmentLoadOp loadOp=AttachmentLoadOp↩ ::Load, const AttachmentStoreOp storeOp=AttachmentStoreOp::Store)

  *Constructor to initialize the format and optionally the load and store operations.*

### Public Attributes

- Format format = Format::Undefined

  *Specifies the render target attachment format. By default Format::Undefined.*
- AttachmentLoadOp loadOp = AttachmentLoadOp::Undefined

  *Specifies the load operation of the previous attachment content. By default AttachmentLoadOp::Undefined.*
- AttachmentStoreOp storeOp = AttachmentStoreOp::Undefined

  *Specifies the store operation of the outcome for the respective attachment content. By default AttachmentStoreOp↩ ::Undefined.*

## 10.4.1 Detailed Description

Render target attachment descriptor structure.

**Remarks**

Two attachment format descriptors are considered compatible when their formats are matching.

**See also**

RenderPassDescriptor

## 10.4.2 Constructor & Destructor Documentation

### 10.4.2.1 LLGL::AttachmentFormatDescriptor::AttachmentFormatDescriptor ( ) [default]

### 10.4.2.2 LLGL::AttachmentFormatDescriptor::AttachmentFormatDescriptor ( const AttachmentFormatDescriptor & ) [default]

### 10.4.2.3 LLGL::AttachmentFormatDescriptor::AttachmentFormatDescriptor ( const Format *format*, const AttachmentLoadOp *loadOp* = AttachmentLoadOp::Load, const AttachmentStoreOp *storeOp* = AttachmentStoreOp::Store ) [inline]

Constructor to initialize the format and optionally the load and store operations.

## 10.4.3 Member Data Documentation

### 10.4.3.1 Format LLGL::AttachmentFormatDescriptor::format = Format::Undefined

Specifies the render target attachment format. By default Format::Undefined.

**Remarks**

If the render pass is used for a render context, the appropriate color format can be determined by the RenderContext::GetColorFormat function, and the appropriate depth-stencil format can be determined by the RenderContext::GetDepthStencilFormat function. If the render pass is used for render targets, the format depends on the render target attachments. If this is undefined, the corresponding attachment is not used.

**See also**

RenderContext::GetColorFormat
RenderContext::GetDepthStencilFormat

#### 10.4.3.2 AttachmentLoadOp LLGL::AttachmentFormatDescriptor::loadOp = AttachmentLoadOp::Undefined

Specifies the load operation of the previous attachment content. By default AttachmentLoadOp::Undefined.

**Remarks**

If the attachment is meant to be cleared when a render pass begins, set this to AttachmentLoadOp::Clear.

**See also**

AttachmentLoadOp

#### 10.4.3.3 AttachmentStoreOp LLGL::AttachmentFormatDescriptor::storeOp = AttachmentStoreOp::Undefined

Specifies the store operation of the outcome for the respective attachment content. By default AttachmentStore↩Op::Undefined.

**See also**

AttachmentStoreOp

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

- RenderPassFlags.h

## 10.5 LLGL::BindFlags Struct Reference

Flags for Buffer and Texture resources that describe for which purposes they will be used.

```
#include <ResourceFlags.h>
```

### Public Types

- enum {
  VertexBuffer = (1 << 0), IndexBuffer = (1 << 1), ConstantBuffer = (1 << 2), StreamOutputBuffer = (1 << 3),
  IndirectBuffer = (1 << 4), Sampled = (1 << 5), Storage = (1 << 6), ColorAttachment = (1 << 7),
  DepthStencilAttachment = (1 << 8), CombinedTextureSampler = (1 << 9) }

### 10.5.1 Detailed Description

Flags for Buffer and Texture resources that describe for which purposes they will be used.

**See also**

BufferDescriptor::bindFlags
TextureDescriptor::bindFlags
BindingDescriptor::bindFlags

## 10.5.2 Member Enumeration Documentation

### 10.5.2.1 anonymous enum

**Enumerator**

***VertexBuffer*** The resource can be used to bind a stream of vertices.

**Remarks**

This can only be used for Buffer resources.

**See also**

CommandBuffer::SetVertexBuffer

***IndexBuffer*** The resource can be used to bind a stream of indices.

**Remarks**

This can only be used for Buffer resources.

**See also**

CommandBuffer::SetIndexBuffer

***ConstantBuffer*** The resource can be used to bind a set of constants.

**Remarks**

This can only be used for Buffer resources.

***StreamOutputBuffer*** The resource can be used to bind an output stream buffer (also referred to as "transform feedback").

**Remarks**

This can only be used for Buffer resources.

**See also**

CommandBuffer::SetStreamOutputBuffer

***IndirectBuffer*** Hint to the renderer that the resource will hold the arguments for indirect commands.

**Remarks**

This can only be used for Buffer resources.

**See also**

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect
CommandBuffer::DispatchIndirect

***Sampled*** The resource can be used to bind a buffer or texture for read access.

**Remarks**

This can be used for Buffer resources (e.g. `samplerBuffer` in GLSL, or `Structured↩Buffer` in HLSL) and Texture resources (e.g. `sampler2D` in GLSL, or `Texture2D` in HLSL).

***Storage*** The resource can be used to bind a buffer or texture for unordered read/write access.

**Remarks**

This can be used for Buffer resources (e.g. `buffer` in GLSL, or `RWStructuredBuffer` in HLSL) and Texture resources (e.g. `image2D` in GLSL, or `RWTexture2D` in HLSL).

***ColorAttachment*** Texture can be used as render target color attachment.

**Remarks**

This can only be used for Texture resources.

**Note**

This cannot be used together with the BindFlags::DepthStencilAttachment flag.

**See also**

AttachmentDescriptor::texture
AttachmentType::Color

***DepthStencilAttachment*** Texture can be used as render target depth-stencil attachment.

**Remarks**

This can only be used for Texture resources.

**Note**

This cannot be used together with the BindFlags::ColorAttachment flag.

**See also**

AttachmentDescriptor::texture
AttachmentType::DepthStencil

***CombinedTextureSampler*** Specifies a resource as a combination of a Texture and Sampler (e.g. `sampler2D` in GLSL).

**Remarks**

This is only used for shader reflection and ignored by resource creation.

**Note**

Only supported with: OpenGL, Vulkan.

**See also**

ShaderResource::binding

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

- ResourceFlags.h

## 10.6 LLGL::BindingDescriptor Struct Reference

Layout structure for a single binding point of the pipeline layout descriptor.

```
#include <PipelineLayoutFlags.h>
```

### Public Member Functions

- BindingDescriptor ()=default
- BindingDescriptor (const BindingDescriptor &)=default
- BindingDescriptor (ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1)

  *Constructors with all primary attributes and a default value for a uniform array.*
- BindingDescriptor (const std::string &name, ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1)

  *Constructors with all attributes.*

### Public Attributes

- std::string name

  *Optional name for shading languages that do not support binding slots within the shader.*
- ResourceType type = ResourceType::Undefined

  *Resource view type for this layout binding. By default ResourceType::Undefined.*
- long bindFlags = 0

  *Specifies to which kind of resource slot the resource will be bound. By default 0.*
- long stageFlags = 0

  *Specifies which shader stages are affected by this layout binding. By default 0.*
- std::uint32_t slot = 0

  *Specifies the zero-based binding slot. By default 0.*
- std::uint32_t arraySize = 1

  *Specifies the number of binding slots for an array resource. By default 1.*

## 10.6.1 Detailed Description

Layout structure for a single binding point of the pipeline layout descriptor.

**See also**

PipelineLayoutDescriptor::bindings

## 10.6.2 Constructor & Destructor Documentation

### 10.6.2.1 LLGL::BindingDescriptor::BindingDescriptor ( ) [default]

### 10.6.2.2 LLGL::BindingDescriptor::BindingDescriptor ( const BindingDescriptor & ) [default]

### 10.6.2.3 LLGL::BindingDescriptor::BindingDescriptor ( ResourceType *type*, long *bindFlags*, long *stageFlags*, std::uint32_t *slot*, std::uint32_t *arraySize* = 1 ) [inline]

Constructors with all primary attributes and a default value for a uniform array.

### 10.6.2.4 LLGL::BindingDescriptor::BindingDescriptor ( const std::string & *name*, ResourceType *type*, long *bindFlags*, long *stageFlags*, std::uint32_t *slot*, std::uint32_t *arraySize* = 1 ) [inline]

Constructors with all attributes.

## 10.6.3 Member Data Documentation

### 10.6.3.1 std::uint32_t LLGL::BindingDescriptor::arraySize = 1

Specifies the number of binding slots for an array resource. By default 1.

**Note**

For Vulkan, this number specifies the size of an array of resources (e.g. an array of uniform buffers).

#### 10.6.3.2 long LLGL::BindingDescriptor::bindFlags = 0

Specifies to which kind of resource slot the resource will be bound. By default 0.

**Remarks**

When a Buffer is bound to a constant buffer slot for instance, the BindFlags::ConstantBuffer is requried. For texture resources, the bind flags are optional and only required to bind a read/write texture, e.g. `image2D` in GLSL or `RWTexture2D` in HLSL, using the BindFlags::Storage flag.

**See also**

BindFlags

#### 10.6.3.3 std::string LLGL::BindingDescriptor::name

Optional name for shading languages that do not support binding slots within the shader.

**Remarks**

This is only used for the OpenGL backend, when the GLSL version does not support `explicit binding points`. If GLSL 420 or later is supported, this can be ignored and the binding points can be specified like this:

```
#version 420
layout(binding = 1) uniform sampler2D mySampler;
```

Otherwise, the name of the resource must be included in this binding descriptor, e.g. `"mySampler"`.

#### 10.6.3.4 std::uint32_t LLGL::BindingDescriptor::slot = 0

Specifies the zero-based binding slot. By default 0.

**Note**

For Vulkan, each binding slot of all layout bindings must have a different value within a pipeline layout.

#### 10.6.3.5 long LLGL::BindingDescriptor::stageFlags = 0

Specifies which shader stages are affected by this layout binding. By default 0.

**Remarks**

This can be a bitwise OR combination of the StageFlags bitmasks.

**See also**

StageFlags

#### 10.6.3.6 ResourceType LLGL::BindingDescriptor::type = ResourceType::Undefined

Resource view type for this layout binding. By default ResourceType::Undefined.

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

- PipelineLayoutFlags.h

## 10.7 LLGL::BlendDescriptor Struct Reference

Blending state descriptor structure.

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

### Public Attributes

- bool alphaToCoverageEnabled = false

  *Specifies whether to use alpha-to-coverage as a multi-sampling technique when setting a pixel to a render target. By default disabled.*
- bool independentBlendEnabled = false

  *Specifies whether to enable independent blending in simultaneous color attachments. By default false.*
- LogicOp logicOp = LogicOp::Disabled

  *Specifies the logic fragment operation. By default LogicOp::Disabled.*
- ColorRGBAf blendFactor = { 0.0f, 0.0f, 0.0f, 0.0f }

  *Specifies the blending color factor. By default (0, 0, 0, 0).*
- BlendTargetDescriptor targets [8]

  *Render-target blend states for the respective color attachments. A maximum of 8 targets is supported.*

### 10.7.1 Detailed Description

Blending state descriptor structure.

**See also**

GraphicsPipelineDescriptor::blend

### 10.7.2 Member Data Documentation

#### 10.7.2.1 bool LLGL::BlendDescriptor::alphaToCoverageEnabled = false

Specifies whether to use alpha-to-coverage as a multi-sampling technique when setting a pixel to a render target. By default disabled.

**Remarks**

This is useful when multi-sampling is enabled and alpha tests are implemented in a fragment shader (e.g. to render fences, plants, or other transparent geometries).

#### 10.7.2.2 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

**Todo** Move this into a dynamic function "CommandBuffer::SetBlendFactor".

#### 10.7.2.3 bool LLGL::BlendDescriptor::independentBlendEnabled = false

Specifies whether to enable independent blending in simultaneous color attachments. By default false.

**Remarks**

If this is true, each color attachment has its own blending configuration described in the `targets` array. Otherwise, each color attachment uses the blending configuration described only by the first entry of the `targets` array, i.e. `targets[0]` and all remaining entries `targets[1..7]` are ignored.

**See also**

targets

#### 10.7.2.4 LogicOp LLGL::BlendDescriptor::logicOp = LogicOp::Disabled

Specifies the logic fragment operation. By default LogicOp::Disabled.

**Remarks**

Logic pixel operations can not be used in combination with color and alpha blending. Therefore, if this is not LogicOp::Disabled, `independentBlendEnabled` must be false and `blendEnabled` of the first target must be false as well. If logic fragment operations are not supported by the renderer, this must be LogicOp::Disabled.

**Note**

For Direct3D 11, feature level 11.1 is required.

**See also**

blendEnabled
RenderingFeatures::hasLogicOp

#### 10.7.2.5 BlendTargetDescriptor LLGL::BlendDescriptor::targets[8]

Render-target blend states for the respective color attachments. A maximum of 8 targets is supported.

**Remarks**

If `independentBlendEnabled` is set to `false`, only the first entry is used, i.e. `targets[0]` and all remaining entries `targets[1..7]` are ignored.

**See also**

independentBlendEnabled

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

- GraphicsPipelineFlags.h

## 10.8 LLGL::BlendTargetDescriptor Struct Reference

Blend target state descriptor structure.

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

### Public Attributes

- bool blendEnabled = false

  *Specifies whether blending is enabled or disabled for the respective color attachment.*
- BlendOp srcColor = BlendOp::SrcAlpha

  *Source color blending operation. By default BlendOp::SrcAlpha.*
- BlendOp dstColor = 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 dstAlpha = BlendOp::InvSrcAlpha

  *Destination alpha blending operation. By default BlendOp::InvSrcAlpha.*
- BlendArithmetic alphaArithmetic = BlendArithmetic::Add

  *Alpha blending arithmetic. By default BlendArithmetic::Add.*
- ColorRGBAb colorMask = { true, true, true, true }

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

### 10.8.1 Detailed Description

Blend target state descriptor structure.

**See also**

BlendDescriptor::targets

### 10.8.2 Member Data Documentation

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

Alpha blending arithmetic. By default BlendArithmetic::Add.

#### 10.8.2.2 bool LLGL::BlendTargetDescriptor::blendEnabled = false

Specifies whether blending is enabled or disabled for the respective color attachment.

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

Color blending arithmetic. By default BlendArithmetic::Add.

#### 10.8.2.4 ColorRGBAb LLGL::BlendTargetDescriptor::colorMask = { true, true, true, true }

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

**Remarks**

If no pixel shader is used in the graphics pipeline, the color mask **must** be set to `false` for all components. Otherwise, the behavior is undefined.

#### 10.8.2.5 BlendOp LLGL::BlendTargetDescriptor::dstAlpha = BlendOp::InvSrcAlpha

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

#### 10.8.2.6 BlendOp LLGL::BlendTargetDescriptor::dstColor = BlendOp::InvSrcAlpha

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

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

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

#### 10.8.2.8 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

## 10.9 LLGL::Buffer Class Reference

Hardware buffer interface.

`#include <Buffer.h>`

Inheritance diagram for LLGL::Buffer:

### Public Member Functions

- ResourceType GetResourceType () const override final

  *Returns ResourceType::Buffer.*
- long GetBindFlags () const

  *Returns the binding flags this buffer was created with.*
- virtual BufferDescriptor GetDesc () const =0

  *Queries a descriptor of this buffer.*

### Protected Member Functions

- Buffer (long bindFlags)

### 10.9.1 Detailed Description

Hardware buffer interface.

**See also**

RenderSystem::CreateBuffer

### 10.9.2 Constructor & Destructor Documentation

#### 10.9.2.1 LLGL::Buffer::Buffer ( long *bindFlags* ) [protected]

### 10.9.3 Member Function Documentation

#### 10.9.3.1 long LLGL::Buffer::GetBindFlags ( ) const [inline]

Returns the binding flags this buffer was created with.

**See also**

BufferDescriptor::bindFlags

#### 10.9.3.2 virtual BufferDescriptor LLGL::Buffer::GetDesc ( ) const [pure virtual]

Queries a descriptor of this buffer.

**Remarks**

This function only queries the following attributes:

- `size`
- `bindFlags`
- `cpuAccessFlags`
- `miscFlags`

All other attributes (such as `vertexBuffer` etc.) cannot be queried by this function. Those attributes are set to the default value specified in BufferDescriptor.
The returned flags (such as `cpuAccessFlags` etc.) are not necessarily the same that were specified when the resource was created. They reflect the capabilities of the actual hardware buffer. For example, a buffer created with CPUAccessFlags::Read might return CPUAccessFlags::ReadWrite, if the renderer backend does not distinguish between different CPU access flags.

**See also**

BufferDescriptor
Texture::GetDesc

#### 10.9.3.3 ResourceType LLGL::Buffer::GetResourceType ( ) const [final],[override],[virtual]

Returns ResourceType::Buffer.

Implements LLGL::Resource.

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

- Buffer.h

## 10.10 LLGL::BufferArray Class Reference

Hardware buffer container interface.

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

Inheritance diagram for LLGL::BufferArray:

### Public Member Functions

- long GetBindFlags () const

  *Returns the binding flags this buffer array was created with.*

### Protected Member Functions

- BufferArray (long bindFlags)

### 10.10.1 Detailed Description

Hardware buffer container interface.

**Remarks**

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

**See also**

RenderSystem::CreateBufferArray

### 10.10.2 Constructor & Destructor Documentation

#### 10.10.2.1 LLGL::BufferArray::BufferArray ( long *bindFlags* ) [protected]

### 10.10.3 Member Function Documentation

#### 10.10.3.1 long LLGL::BufferArray::GetBindFlags ( ) const [inline]

Returns the binding flags this buffer array was created with.

**See also**

BufferDescriptor::bindFlags

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

- BufferArray.h

## 10.11 LLGL::BufferDescriptor Struct Reference

Hardware buffer descriptor structure.

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

### Classes

- struct IndexBuffer

  *Index buffer specific descriptor structure.*
- struct StorageBuffer

  *Storage buffer specific descriptor structure.*
- struct VertexBuffer

  *Vertex buffer specific descriptor structure.*

### Public Attributes

- std::uint64_t size = 0

  *Buffer size (in bytes). This must not be larger than RenderingLimits::maxBufferSize. By default 0.*
- long bindFlags = 0

  *These flags describe to which resource slots the buffer can be bound. By default 0.*
- long cpuAccessFlags = 0

  *CPU read/write access flags. By default 0.*
- long miscFlags = 0

  *Miscellaneous buffer flags. By default 0.*
- VertexBuffer vertexBuffer

  *Vertex buffer type descriptor appendix.*
- IndexBuffer indexBuffer

  *Index buffer type descriptor appendix.*
- StorageBuffer storageBuffer

  *Storage buffer type descriptor appendix.*

## 10.11.1 Detailed Description

Hardware buffer descriptor structure.

**See also**

RenderSystem::CreateBuffer

## 10.11.2 Member Data Documentation

### 10.11.2.1 long LLGL::BufferDescriptor::bindFlags = 0

These flags describe to which resource slots the buffer can be bound. By default 0.

**Remarks**

When the buffer will be bound to a vertex buffer slot for instance, the BindFlags::VertexBuffer flag is required.

**See also**

BindFlags

#### 10.11.2.2 long LLGL::BufferDescriptor::cpuAccessFlags = 0

CPU read/write access flags. By default 0.

**Remarks**

If this is 0 the buffer cannot be mapped from GPU memory space into CPU memory space and vice versa.

**See also**

CPUAccessFlags
RenderSystem::MapBuffer

#### 10.11.2.3 IndexBuffer LLGL::BufferDescriptor::indexBuffer

Index buffer type descriptor appendix.

#### 10.11.2.4 long LLGL::BufferDescriptor::miscFlags = 0

Miscellaneous buffer flags. By default 0.

**Remarks**

This can be used as a hint for the renderer how frequently the buffer will be updated.

**See also**

MiscFlags

#### 10.11.2.5 std::uint64_t LLGL::BufferDescriptor::size = 0

Buffer size (in bytes). This must not be larger than RenderingLimits::maxBufferSize. By default 0.

**Remarks**

If the buffer has the BindFlags::Sampled or BindFlags::Storage flag, then `size` must be a multiple of `storageBuffer.stride`.

**See also**

RenderingLimits::maxBufferSize
bindFlags
StorageBuffer::stride

#### 10.11.2.6 StorageBuffer LLGL::BufferDescriptor::storageBuffer

Storage buffer type descriptor appendix.

#### 10.11.2.7 VertexBuffer LLGL::BufferDescriptor::vertexBuffer

Vertex buffer type descriptor appendix.

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

- BufferFlags.h

## 10.12 LLGL::Canvas Class Reference

Canvas interface for mobile platforms.

```
#include <Canvas.h>
```

Inheritance diagram for LLGL::Canvas:

### Classes

- class EventListener

  *Interface for all canvas event listeners.*

### Public Member Functions

- virtual void SetTitle (const std::wstring &title)=0

  *Sets the canvas title as UTF16 string. If the OS does not support UTF16 window title, it will be converted to UTF8.*
- virtual std::wstring GetTitle () const =0

  *Returns the canvas title as UTF16 string.*
- virtual bool HasQuit () const

  *Returns true if this canvas is in the 'Quit' state.*
- bool AdaptForVideoMode (VideoModeDescriptor &videoModeDesc) override

  *This default implementation ignores the video mode descriptor completely and always return false.*
- bool ProcessEvents () override final

  *Processes the events for this canvas (i.e. touch input, key presses etc.).*
- void AddEventListener (const std::shared_ptr< EventListener > &eventListener)

  *Adds a new event listener to this canvas.*
- void RemoveEventListener (const EventListener *eventListener)

  *Removes the specified event listener from this canvas.*
- void PostQuit ()

  *Posts a 'Quit' event to all event listeners.*

### Static Public Member Functions

- static std::unique_ptr< Canvas > Create (const CanvasDescriptor &desc)

  *Creates a platform specific instance of the Canvas interface.*

### Protected Member Functions

- virtual void OnProcessEvents ()=0

## 10.12.1 Detailed Description

Canvas interface for mobile platforms.

**Remarks**

This is the main interface for the windowing system in LLGL on mobile platforms. The couterpart is the Window interface for desktop platforms.

**See also**

Window

## 10.12.2 Member Function Documentation

### 10.12.2.1 bool LLGL::Canvas::AdaptForVideoMode ( VideoModeDescriptor & *videoModeDesc* ) [override], [virtual]

This default implementation ignores the video mode descriptor completely and always return false.

Implements LLGL::Surface.

### 10.12.2.2 void LLGL::Canvas::AddEventListener ( const std::shared_ptr< EventListener > & *eventListener* )

Adds a new event listener to this canvas.

### 10.12.2.3 static std::unique_ptr<Canvas> LLGL::Canvas::Create ( const CanvasDescriptor & *desc* ) [static]

Creates a platform specific instance of the Canvas interface.

**Returns**

Unique pointer to a new instance of the platform specific Canvas interface or null if the platform does not support canvas (such as Windows, Linux, and macOS).

**Remarks**

For desktop platforms the interface Window can be used.

**See also**

Window

#### 10.12.2.4 virtual std::wstring LLGL::Canvas::GetTitle ( ) const [pure virtual]

Returns the canvas title as UTF16 string.

#### 10.12.2.5 virtual bool LLGL::Canvas::HasQuit ( ) const [virtual]

Returns true if this canvas is in the 'Quit' state.

**See also**

PostQuit
ProcessEvents

#### 10.12.2.6 virtual void LLGL::Canvas::OnProcessEvents ( ) [protected], [pure virtual]

Called inside the "ProcessEvents" function after all event listeners received the same event.

**See also**

ProcessEvents
EventListener::OnProcessEvents

#### 10.12.2.7 void LLGL::Canvas::PostQuit ( )

Posts a 'Quit' event to all event listeners.

**Remarks**

If at least one event listener returns false within the `OnQuit` callback, the canvas will not quit. If all event listeners return true within the `OnQuit` callback, `ProcessEvents` will return false from now on.

**See also**

EventListener::OnQuit
ProcessEvents
HasQuit

#### 10.12.2.8 bool LLGL::Canvas::ProcessEvents ( ) [final], [override], [virtual]

Processes the events for this canvas (i.e. touch input, key presses etc.).

**Returns**

True, as long as the window can process events. Once the `PostQuit` function has set this canvas to the 'Quit' state, this function returns false.

**See also**

PostQuit

Implements LLGL::Surface.

#### 10.12.2.9 void LLGL::Canvas::RemoveEventListener ( const EventListener * *eventListener* )

Removes the specified event listener from this canvas.

#### 10.12.2.10 virtual void LLGL::Canvas::SetTitle ( const std::wstring & *title* ) [pure virtual]

Sets the canvas title as UTF16 string. If the OS does not support UTF16 window title, it will be converted to UTF8.

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

- Canvas.h

## 10.13 LLGL::CanvasDescriptor Struct Reference

Canvas descriptor structure.

```
#include <CanvasFlags.h>
```

### Public Attributes

- std::wstring title

  *Canvas title as UTF16 string.*
- bool borderless = false

  *Specifies whether the canvas is borderless. This is required for a fullscreen render context.*

### 10.13.1 Detailed Description

Canvas descriptor structure.

### 10.13.2 Member Data Documentation

#### 10.13.2.1 bool LLGL::CanvasDescriptor::borderless = false

Specifies whether the canvas is borderless. This is required for a fullscreen render context.

#### 10.13.2.2 std::wstring LLGL::CanvasDescriptor::title

Canvas title as UTF16 string.

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

- CanvasFlags.h

## 10.14 LLGL::ClearFlags Struct Reference

Command buffer clear flags.

```
#include <CommandBufferFlags.h>
```

### Public Types

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

### 10.14.1 Detailed Description

Command buffer clear flags.

**See also**

CommandBuffer::Clear

### 10.14.2 Member Enumeration Documentation

#### 10.14.2.1 anonymous enum

**Enumerator**

***Color*** Clears the color attachment.

***Depth*** Clears the depth attachment.

***Stencil*** Clears the stencil attachment.

***ColorDepth*** Clears the color and depth attachments.

***DepthStencil*** Clears the depth and stencil attachments.

***All*** Clears the color, depth and stencil attachments.

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

- CommandBufferFlags.h

## 10.15 LLGL::ClearValue Struct Reference

Clear value structure for color, depth, and stencil clear operations.

```
#include <CommandBufferFlags.h>
```

### Public Attributes

- ColorRGBAf color = { 0.0f, 0.0f, 0.0f, 0.0f }

  *Specifies the clear value to clear a color attachment. By default (0.0, 0.0, 0.0, 0.0).*
- float depth = 1.0f

  *Specifies the clear value to clear a depth attachment. By default 1.0.*
- std::uint32_t stencil = 0

  *Specifies the clear value to clear a stencil attachment. By default 0.*

### 10.15.1 Detailed Description

Clear value structure for color, depth, and stencil clear operations.

**See also**

AttachmentClear::clearValue
TextureDescriptor::clearValue

### 10.15.2 Member Data Documentation

#### 10.15.2.1 ColorRGBAf LLGL::ClearValue::color = { 0.0f, 0.0f, 0.0f, 0.0f }

Specifies the clear value to clear a color attachment. By default (0.0, 0.0, 0.0, 0.0).

#### 10.15.2.2 float LLGL::ClearValue::depth = 1.0f

Specifies the clear value to clear a depth attachment. By default 1.0.

#### 10.15.2.3 std::uint32_t LLGL::ClearValue::stencil = 0

Specifies the clear value to clear a stencil attachment. By default 0.

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

- CommandBufferFlags.h

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

Base color class with N components.

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

### Public Member Functions

- Color ()

  *Constructors all attributes with the default color value.*
- Color (const Color< T, N > &rhs)

  *Copy constructor.*
- Color (const T &scalar)

  *Constructs all attributes with the specified scalar value.*
- Color (UninitializeTag)

  *Explicitly uninitialized constructor. All attributes are uninitialized!*
- Color< T, N > & operator+= (const Color< T, N > &rhs)

  *Adds the specified color (component wise) to this color.*
- Color< T, N > & operator-= (const Color< T, N > &rhs)

  *Substracts the specified color (component wise) from this color.*
- Color< T, N > & operator∗= (const Color< T, N > &rhs)

  *Multiplies the specified color (component wise) with this color.*
- Color< T, N > & operator/= (const Color< T, N > &rhs)

  *Divides the specified color (component wise) with this color.*
- Color< T, N > & operator∗= (const T rhs)

  *Multiplies the specified scalar value (component wise) with this color.*
- Color< T, N > & operator/= (const T rhs)

  *Divides the specified scalar value (component wise) with this color.*
- 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

  *Returns the negation of this color.*
- template<typename Dst >
  Color< Dst, N > Cast () const

  *Returns a type casted instance of this color.*
- 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.*

## 10.16.1 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. |

## 10.16.2 Constructor & Destructor Documentation

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

Constructors all attributes with the default color value.

**Remarks**

For default color values the 'MaxColorValue' template is used.

**See also**

MaxColorValue

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

Copy constructor.

### 10.16.2.3 template<typename T, std::size_t N> LLGL::Color< T, N >::Color ( const T & *scalar* ) [inline], [explicit]

Constructs all attributes with the specified scalar value.

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

Explicitly uninitialized constructor. All attributes are uninitialized!

**Remarks**

Only use this constructor when you want to allocate a large amount of color elements that are being initialized later.

## 10.16.3 Member Function Documentation

### 10.16.3.1 template<typename T, std::size_t N> template<typename Dst > Color<Dst, N> LLGL::Color< T, N >::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**

| | |
|---|---|
| *Dst* | Specifies the destination type. |

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

Multiplies the specified color (component wise) with this color.

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

Multiplies the specified scalar value (component wise) with this color.

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

Adds the specified color (component wise) to this color.

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

Returns the negation of this color.

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

Substracts the specified color (component wise) from this color.

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

Divides the specified color (component wise) with this color.

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

Divides the specified scalar value (component wise) with this color.

#### 10.16.3.9 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). |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

#### 10.16.3.10 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). |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

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

Returns a pointer to the first element of this vector.

#### 10.16.3.12 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.

### 10.16.4 Member Data Documentation

#### 10.16.4.1 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

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

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

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

## Public Member Functions

- Color ()

  *Constructors all attributes with the default color value.*
- Color (const Color< T, 3 > &rhs)

  *Copy constructor.*
- Color (const T &scalar)

  *Constructs all attributes with the specified scalar value.*
- Color (const T &r, const T &g, const T &b)

  *Constructs all attributes with the specified color values r (red), g (green), b (blue).*
- Color (UninitializeTag)

  *Explicitly uninitialized constructor. All attributes are uninitialized!*
- Color< T, 3 > & operator+= (const Color< T, 3 > &rhs)

  *Adds the specified color (component wise) to this color.*
- Color< T, 3 > & operator-= (const Color< T, 3 > &rhs)

  *Substracts the specified color (component wise) from this color.*
- Color< T, 3 > & operator∗= (const Color< T, 3 > &rhs)

  *Multiplies the specified color (component wise) with this color.*
- Color< T, 3 > & operator/= (const Color< T, 3 > &rhs)

  *Divides the specified color (component wise) with this color.*
- Color< T, 3 > & operator∗= (const T rhs)

  *Multiplies the specified scalar value (component wise) with this color.*
- Color< T, 3 > & operator/= (const T rhs)

  *Divides the specified scalar value (component wise) with this color.*
- Color< T, 3 > operator- () const

  *Returns the negation of this color.*
- T & operator[ ] (std::size_t component)

  *Returns the specified color component.*
- const T & operator[ ] (std::size_t component) const

  *Returns the specified color component.*
- Color< T, 4 > ToRGBA () const

  *Returns this RGB color as RGBA color.*
- template<typename Dst >
  Color< Dst, 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.*

### 10.17.1 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.

### 10.17.2 Constructor & Destructor Documentation

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

Constructors all attributes with the default color value.

**Remarks**

For default color values the 'MaxColorValue' template is used.

**See also**

MaxColorValue

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

Copy constructor.

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

Constructs all attributes with the specified scalar value.

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

Constructs all attributes with the specified color values r (red), g (green), b (blue).

#### 10.17.2.5 template<typename T > LLGL::Color< T, 3u >::Color ( UninitializeTag ) [inline], [explicit]

Explicitly uninitialized constructor. All attributes are uninitialized!

**Remarks**

Only use this constructor when you want to allocate a large amount of color elements that are being initialized later.

### 10.17.3 Member Function Documentation

#### 10.17.3.1 template<typename T > template<typename Dst > Color<Dst, 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**

| | |
|---|---|
| *Dst* | Specifies the destination type. |

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

Multiplies the specified color (component wise) with this color.

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

Multiplies the specified scalar value (component wise) with this color.

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

Adds the specified color (component wise) to this color.

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

Returns the negation of this color.

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

Substracts the specified color (component wise) from this color.

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

Divides the specified color (component wise) with this color.

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

Divides the specified scalar value (component wise) with this color.

#### 10.17.3.9 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. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

#### 10.17.3.10 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. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

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

Returns a pointer to the first element of this color.

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

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

#### 10.17.3.13 template<typename T > Color<T, 4> LLGL::Color< T, 3u >::ToRGBA ( ) const [inline]

Returns this RGB color as RGBA color.

### 10.17.4 Member Data Documentation

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

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

Specifies the number of color components.

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

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

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

- ColorRGB.h

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

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

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

### Public Member Functions

- Color ()

  *Constructors all attributes with the default color value.*
- Color (const Color< T, 4 > &rhs)

  *Copy constructor.*
- Color (const T &scalar)

  *Constructs all attributes with the specified scalar value.*
- Color (const Color< T, 3u > &rhs)

  *Constructs the RGB attributes with the specified RGB color, and the default value for alpha.*
- Color (const T &r, const T &g, const T &b)

  *Constructs the RGB attributes with the specified color values r (red), g (green), b (blue), and the default value for alpha.*
- Color (const T &r, const T &g, const T &b, const T &a)

  *Constructs all attributes with the specified color values r (red), g (green), b (blue), a (alpha).*
- Color (UninitializeTag)

  *Explicitly uninitialized constructor. All attributes are uninitialized!*
- Color< T, 4 > & operator+= (const Color< T, 4 > &rhs)

  *Adds the specified color (component wise) to this color.*
- Color< T, 4 > & operator-= (const Color< T, 4 > &rhs)

  *Substracts the specified color (component wise) from this color.*
- Color< T, 4 > & operator*= (const Color< T, 4 > &rhs)

  *Multiplies the specified color (component wise) with this color.*
- Color< T, 4 > & operator/= (const Color< T, 4 > &rhs)

  *Divides the specified color (component wise) with this color.*
- Color< T, 4 > & operator*= (const T rhs)

  *Multiplies the specified scalar value (component wise) with this color.*
- Color< T, 4 > & operator/= (const T rhs)

  *Divides the specified scalar value (component wise) with this color.*
- Color< T, 4 > operator- () const

  *Returns the negation of this color.*
- T & operator[ ] (std::size_t component)

  *Returns the specified color component.*
- const T & operator[ ] (std::size_t component) const

*Returns the specified color component.*

- Color< T, 3 > ToRGB () const

  *Returns this RGBA color as RGB color.*

- template<typename Dst >
  Color< Dst, 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.*

### 10.18.1 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.

### 10.18.2 Constructor & Destructor Documentation

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

Constructors all attributes with the default color value.

**Remarks**

For default color values the 'MaxColorValue' template is used.

**See also**

MaxColorValue

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

Copy constructor.

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

Constructs all attributes with the specified scalar value.

#### 10.18.2.4 template<typename T > LLGL::Color< T, 4u >::Color ( const Color< T, 3u > & *rhs* ) [inline], [explicit]

Constructs the RGB attributes with the specified RGB color, and the default value for alpha.

**Remarks**

For default color values the 'MaxColorValue' template is used.

**See also**

MaxColorValue

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

Constructs the RGB attributes with the specified color values r (red), g (green), b (blue), and the default value for alpha.

**Remarks**

For default color values the 'MaxColorValue' template is used.

**See also**

MaxColorValue

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

Constructs all attributes with the specified color values r (red), g (green), b (blue), a (alpha).

#### 10.18.2.7 template<typename T > LLGL::Color< T, 4u >::Color ( UninitializeTag ) [inline], [explicit]

Explicitly uninitialized constructor. All attributes are uninitialized!

**Remarks**

Only use this constructor when you want to allocate a large amount of color elements that are being initialized later.

### 10.18.3 Member Function Documentation

#### 10.18.3.1 template<typename T > template<typename Dst > Color<Dst, 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**

| | |
|---|---|
| *Dst* | Specifies the destination type. |

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

Multiplies the specified color (component wise) with this color.

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

Multiplies the specified scalar value (component wise) with this color.

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

Adds the specified color (component wise) to this color.

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

Returns the negation of this color.

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

Substracts the specified color (component wise) from this color.

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

Divides the specified color (component wise) with this color.

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

Divides the specified scalar value (component wise) with this color.

#### 10.18.3.9 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. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

#### 10.18.3.10 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. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the specified component index is out of range (Only if the macro 'LLGL_DEBUG' is defined). |

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

Returns a pointer to the first element of this color.

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

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

#### 10.18.3.13 template<typename T > Color<T, 3> LLGL::Color< T, 4u >::ToRGB ( ) const [inline]

Returns this RGBA color as RGB color.

### 10.18.4 Member Data Documentation

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

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

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

Specifies the number of color components.

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

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

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

- ColorRGBA.h

## 10.19 LLGL::CommandBuffer Class Reference

Command buffer interface used for storing and encoding GPU commands.

```
#include <CommandBuffer.h>
```

Inheritance diagram for LLGL::CommandBuffer:

### Public Member Functions

- virtual void Begin ()=0

  *Begins with the encoding (also referred to as "recording") of this command buffer.*
- virtual void End ()=0

  *Ends the encoding (also referred to as "recording") of this command buffer.*
- virtual void Execute (CommandBuffer &deferredCommandBuffer)=0

  *Executes the specified deferred command buffer.*
- virtual void UpdateBuffer (Buffer &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint16_t dataSize)=0

  *Updates the data of the specified buffer during encoding the command buffer.*
- virtual void CopyBuffer (Buffer &dstBuffer, std::uint64_t dstOffset, Buffer &srcBuffer, std::uint64_t srcOffset, std::uint64_t size)=0

  *Encodes a buffer copy command for the specified buffer region.*
- virtual void CopyTexture (Texture &dstTexture, const TextureLocation &dstLocation, Texture &srcTexture, const TextureLocation &srcLocation, const Extent3D &extent)=0

  *Encodes a texture copy command for the specified texture regions.*
- virtual void GenerateMips (Texture &texture)=0

  *Generates all MIP-maps for the specified texture.*
- virtual void GenerateMips (Texture &texture, const TextureSubresource &subresource)=0

  *Generates the specified range of MIP-maps for the specified texture.*
- virtual void SetViewport (const Viewport &viewport)=0

*Sets a single viewport.*

- virtual void SetViewports (std::uint32_t numViewports, const Viewport *viewports)=0

  *Sets an array of viewports.*

- virtual void SetScissor (const Scissor &scissor)=0

  *Sets a single scissor rectangle.*

- virtual void SetScissors (std::uint32_t numScissors, const Scissor *scissors)=0

  *Sets an array of scissor rectangles, but only if the scissor test was enabled in the previously set graphics pipeline (otherwise, this function has no effect).*

- 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 (std::uint32_t stencil)=0

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

- virtual void Clear (long flags)=0

  *Clears the specified group of attachments of the active render target.*

- virtual void ClearAttachments (std::uint32_t numAttachments, const AttachmentClear *attachments)=0

  *Clears the specified attachments 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 SetIndexBuffer (Buffer &buffer, const Format format, std::uint64_t offset=0)=0

  *Sets the active index buffer for subsequent drawing operations with a dynamic format and optional buffer offset.*

- 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 SetGraphicsResourceHeap (ResourceHeap &resourceHeap, std::uint32_t firstSet=0)=0

  *Binds the specified resource heap to the graphics pipeline.*

- virtual void SetComputeResourceHeap (ResourceHeap &resourceHeap, std::uint32_t firstSet=0)=0

  *Binds the specified resource heap to the compute pipeline.*

- virtual void SetResource (Resource &resource, std::uint32_t slot, long bindFlags, long stageFlags=StageFlags::AllStages)=0

  *Sets the specified resource to a binding slot.*

- virtual void ResetResourceSlots (const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_t numSlots, long bindFlags, long stageFlags=StageFlags::AllStages)=0

  *Resets the binding slots for the specified resources.*

- virtual void BeginRenderPass (RenderTarget &renderTarget, const RenderPass *renderPass=nullptr, std::uint32_t numClearValues=0, const ClearValue *clearValues=nullptr)=0

  *Begins with a new render pass.*

- virtual void EndRenderPass ()=0

  *Ends the current render pass.*

- 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 SetUniform (UniformLocation location, const void *data, std::uint32_t dataSize)=0

  *Sets the value of a single uniform (a.k.a. shader constant) in the shader program that is currently bound.*
- virtual void SetUniforms (UniformLocation location, std::uint32_t count, const void *data, std::uint32_t dataSize)=0

  *Sets the value of multiple uniforms (a.k.a. shader constants) in the shader program that is currently bound.*
- virtual void BeginQuery (QueryHeap &queryHeap, std::uint32_t query=0)=0

  *Begins a query of the specified query heap.*
- virtual void EndQuery (QueryHeap &queryHeap, std::uint32_t query=0)=0

  *Ends the specified query.*
- virtual void BeginRenderCondition (QueryHeap &queryHeap, std::uint32_t query=0, const RenderConditionMode mode=RenderConditionMode::Wait)=0

  *Begins conditional rendering with the specified query object.*
- virtual void EndRenderCondition ()=0

  *Ends the current render condition.*
- virtual void Draw (std::uint32_t numVertices, std::uint32_t firstVertex)=0

  *Draws the specified amount of primitives from the currently set vertex buffer.*
- virtual void DrawIndexed (std::uint32_t numIndices, std::uint32_t firstIndex)=0
- virtual void DrawIndexed (std::uint32_t numIndices, std::uint32_t firstIndex, std::int32_t vertexOffset)=0

  *Draws the specified amount of primitives from the currently set vertex- and index buffers.*
- virtual void DrawInstanced (std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances)=0
- virtual void DrawInstanced (std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances, std::uint32_t firstInstance)=0

  *Draws the specified amount of instances of primitives from the currently set vertex buffer.*
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset, std::uint32_t firstInstance)=0

  *Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.*
- virtual void DrawIndirect (Buffer &buffer, std::uint64_t offset)=0

  *Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.*
- virtual void DrawIndirect (Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0

  *Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.*
- virtual void DrawIndexedIndirect (Buffer &buffer, std::uint64_t offset)=0

  *Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.*
- virtual void DrawIndexedIndirect (Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0

  *Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.*
- virtual void Dispatch (std::uint32_t numWorkGroupsX, std::uint32_t numWorkGroupsY, std::uint32_t numWorkGroupsZ)=0

  *Dispatches a compute command.*
- virtual void DispatchIndirect (Buffer &buffer, std::uint64_t offset)=0

  *Dispatches a compute command with an unknown amount of thread grounds.*
- virtual void PushDebugGroup (const char *name)=0

  *Pushes the specified name onto a stack of group strings that is used for debug reports.*
- virtual void PopDebugGroup ()=0
- virtual void SetGraphicsAPIDependentState (const void *stateDesc, std::size_t stateDescSize)=0

  *Sets a few low-level graphics API dependent states.*

### Protected Member Functions

- CommandBuffer ()=default

## 10.19.1 Detailed Description

Command buffer interface used for storing and encoding GPU commands.

**Remarks**

This is the main interface to encode graphics, compute, and blit commands to be submitted to the GPU. You can assume that all states that can be changed with a setter function are not persistent across several encoding sections, unless the opposite is mentioned. Before any command can be encoded, the command buffer must be set into encode mode, which is done by the CommandBuffer::Begin function.

**See also**

RenderSystem::CreateCommandBuffer

## 10.19.2 Constructor & Destructor Documentation

### 10.19.2.1 LLGL::CommandBuffer::CommandBuffer ( ) [protected], [default]

## 10.19.3 Member Function Documentation

### 10.19.3.1 virtual void LLGL::CommandBuffer::Begin ( ) [pure virtual]

Begins with the encoding (also referred to as "recording") of this command buffer.

**Remarks**

All functions of the CommandBuffer interface must be used between a call to `Begin` and `End`. This function also resets all previously encoded commands.

**See also**

End

### 10.19.3.2 virtual void LLGL::CommandBuffer::BeginQuery ( QueryHeap & *queryHeap*, std::uint32_t *query* = 0 ) [pure virtual]

Begins a query of the specified query heap.

**Parameters**

| | | |
|---|---|---|
| in | *queryHeap* | Specifies the query heap. |
| in | *query* | Specifies the zero-based index of the query within the heap to begin with. By default 0. This must be in the half-open range [0, QueryHeapDescriptor::numQueries). |

**Remarks**

The `BeginQuery` and `EndQuery` functions can be wrapped around any drawing and/or compute operation. This can be an occlusion query for instance, which determines how many fragments have passed the depth test. The result of a query can be retrieved by the command queue after this command buffer has been submitted.

**See also**

EndQuery
RenderSystem::CreateQueryHeap
CommandQueue::QueryResult

#### 10.19.3.3 virtual void LLGL::CommandBuffer::BeginRenderCondition ( QueryHeap & *queryHeap*, std::uint32_t *query* = 0, const RenderConditionMode *mode* = RenderConditionMode::Wait ) [pure virtual]

Begins conditional rendering with the specified query object.

**Parameters**

| | | |
|---|---|---|
| in | *queryHeap* | Specifies the query heap. This query heap must have been created with the `renderCondition` member set to `true`. |
| in | *query* | Specifies the zero-based index of the query within the heap which is to be used as render condition. By default 0. This must be in the half-open range `[0, QueryHeapDescriptor::numQueries)`. |
| in | *mode* | Specifies the mode of the render condition. |

**Remarks**

Here is a usage example:

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

**See also**

RenderSystem::CreateQueryHeap
QueryHeapDescriptor::renderCondition

#### 10.19.3.4 virtual void LLGL::CommandBuffer::BeginRenderPass ( RenderTarget & *renderTarget*, const RenderPass * *renderPass* = nullptr, std::uint32_t *numClearValues* = 0, const ClearValue * *clearValues* = nullptr ) [pure virtual]

Begins with a new render pass.

**Parameters**

| | | |
|---|---|---|
| in | *renderTarget* | Specifies the render target in which the subsequent draw operations will be stored. |

**Parameters**

| | | |
|---|---|---|
| in | *renderPass* | Specifies an optional render pass object. If this is null, the default render pass for the specified render target will be used. This render pass object must be compatible with the render pass object the specified render target was created with. |
| in | *numClearValues* | Specifies the number of clear values that are specified in the `clearValues` parameter. This *should* be greater than or equal to the number of render pass attachments whose load operation (i.e. AttachmentFormatDescriptor::loadOp) is set to AttachmentLoadOp::Clear. Otherwise, the default values from `SetClearColor`, `SetClearDepth`, and `SetClearStencil` are used. |
| in | *clearValues* | Optional pointer to the array of clear values. If `numClearValues` is not zero, this must be a valid pointer to an array of at least `numClearValues` entries. Each entry in the array is used to clear the attachment whose load operation is set to AttachmentLoadOp::Clear, where the depth attachment (i.e. RenderPassDescriptor::depthAttachment) and the stencil attachment (i.e. RenderPassDescriptor::stencilAttachment) are combined and appear as the last entry. |

**Remarks**

This function starts a new render pass section and must be ended with the `EndRenderPass` function. The following example shows how to use a render pass to clear a render target with two color attachments and a depth-stencil attachment:

```
LLGL::ClearValue myClearValues[3];

// Set clear values for color attachments 0 and 1
myClearValues[0].color = { 1, 0, 0, 1 };
myClearValues[1].color = { 0, 1, 0, 1 };

// Set clear values for depth-stencil attachment
myClearValues[2].depth   = 1.0f;
myClearValues[2].stencil = 0;

// Begin render pass and clear render target
myCmdBuffer->BeginRenderPass(*myRenderTarget, *myRenderPass, 3, myClearValues);
{
    // Draw scene ...
}
myCmdBuffer->EndRenderPass();
```

The following commands can only appear *inside* a render pass section:

- Drawing commands (i.e. `Draw`, `DrawInstanced`, `DrawIndexed`, `DrawIndexedInstanced`, `DrawIndirect` and `DrawIndexedIndirect`).
- Clear attachment commands (i.e. `Clear`, and `ClearAttachments`).
- Query block (i.e. `BeginQuery` and `EndQuery`).
- Conditional render block (i.e. `BeginRenderCondition` and `EndRenderCondition`).

The following commands can only appear *outside* a render pass section:

- Dispatch compute commands (i.e. `Dispatch` and `DispatchIndirect`).

**See also**

RenderSystem::CreateRenderPass
RenderSystem::CreateRenderTarget
RenderTargetDescriptor::renderPass
AttachmentFormatDescriptor::loadOp
EndRenderPass

### 10.19.3.5 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

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

Clears the specified group of attachments 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. If this contains the ClearFlags::Color bit, all color attachments of the active render target are cleared with the color previously set by `SetClearColor`. |

**Remarks**

To specify the clear values for each buffer type, use the respective `SetClear...` function. To clear only a specific render-target color buffer, use the `ClearAttachments` function. Clearing a depth-stencil attachment while the active render target has no depth-stencil buffer is allowed but has no effect. For efficiency reasons, it is recommended to clear the render target attachments when a new render pass begins, i.e. the clear values of the `BeginRenderPass` function should be prefered over this function. For some render systems (e.g. Metal) this function forces the current render pass to stop and start again in order to clear the attachments.

**See also**

ClearFlags
SetClearColor
SetClearDepth
SetClearStencil
ClearAttachments
BeginRenderPass

#### 10.19.3.7 virtual void LLGL::CommandBuffer::ClearAttachments ( std::uint32_t *numAttachments*, const AttachmentClear * *attachments* ) [pure virtual]

Clears the specified attachments of the active render target.

**Parameters**

| | | |
|---|---|---|
| in | *numAttachments* | Specifies the number of attachments to clear. |
| in | *attachments* | Pointer to the array of attachment clear commands. This **must not** be null! |

**Remarks**

To clear all color buffers with the same value, use the `Clear` function. Clearing a depth-stencil attachment while the active render target has no depth-stencil buffer is allowed but has no effect. For efficiency reasons, it is recommended to clear the render target attachments when a new render pass begins, i.e. the clear values of the `BeginRenderPass` function should be prefered over this function. For some render systems (e.g. Metal) this function forces the current render pass to stop and start again in order to clear the attachments.

**See also**

Clear
BeginRenderPass

#### 10.19.3.8 virtual void LLGL::CommandBuffer::CopyBuffer ( Buffer & *dstBuffer,* std::uint64_t *dstOffset,* Buffer & *srcBuffer,* std::uint64_t *srcOffset,* std::uint64_t *size* ) `[pure virtual]`

Encodes a buffer copy command for the specified buffer region.

**Parameters**

| | | |
|---|---|---|
| in | *dstBuffer* | Specifies the destination buffer whose data is to be updated. |
| in | *dstOffset* | Specifies the destination offset (in bytes) at which the destination buffer is to be updated. This offset plus the size (i.e. `dstOffset + size`) must be less than or equal to the size of the destination buffer. |
| in | *srcBuffer* | Specifies the source buffer whose data is to be read from. |
| in | *srcOffset* | Specifies teh source offset (in bytes) at which the source buffer is to be read from. This offset plus the size (i.e. `srcOffset + size`) must be less than or equal to the size of the source buffer. |
| in | *size* | Specifies the size of the buffer region to copy. |

**Remarks**

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

#### 10.19.3.9 virtual void LLGL::CommandBuffer::CopyTexture ( Texture & *dstTexture,* const TextureLocation & *dstLocation,* Texture & *srcTexture,* const TextureLocation & *srcLocation,* const Extent3D & *extent* ) `[pure virtual]`

Encodes a texture copy command for the specified texture regions.

**Parameters**

| | | |
|---|---|---|
| in | *dstTexture* | Specifies the destination texture whose data is to be updated. |
| in | *dstLocation* | Specifies the destination location, including MIP-map level and offset. Its offset plus the extent (i.e. `dstLocation.offset + extent`) must be less than or equal to the size of the destination texture. |
| in | *srcTexture* | Specifies the source texture whose data is to be read from. |
| in | *srcLocation* | Specifies the source location, including MIP-map level and offset. Its offset plus the extent (i.e. `srcLocation.offset + extent`) must be less than or equal to the size of the source texture. |

**Parameters**

| | | |
|---|---|---|
| in | *extent* | Specifies the extent of the texture region to copy (see TextureDescriptor::extent). For this function, the extent also includes the array layers, i.e. `y` component for 1D arrays (TextureType::Texture1DArray), and `z` component for 2D and cube arrays (TextureType::Texture2DArray and TextureType::TextureCubeArray). |

**Remarks**

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

#### 10.19.3.10 virtual void LLGL::CommandBuffer::Dispatch ( std::uint32_t *numWorkGroupsX*, std::uint32_t *numWorkGroupsY*, std::uint32_t *numWorkGroupsZ* ) [pure virtual]

Dispatches a compute command.

**Parameters**

| | | |
|---|---|---|
| in | *numWorkGroupsX* | Specifies the number of worker thread groups in the X-dimension. |
| in | *numWorkGroupsY* | Specifies the number of worker thread groups in the Y-dimension. |
| in | *numWorkGroupsZ* | Specifies the number of worker thread groups in the Z-dimension. |

**See also**

SetComputePipeline
RenderingLimits::maxComputeShaderWorkGroups

#### 10.19.3.11 virtual void LLGL::CommandBuffer::DispatchIndirect ( Buffer & *buffer*, std::uint64_t *offset* ) [pure virtual]

Dispatches a compute command with an unknown amount of thread grounds.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the buffer from which the dispatch command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag. |
| in | *offset* | Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4. |

**See also**

DispatchIndirectArguments

#### 10.19.3.12 virtual void LLGL::CommandBuffer::Draw ( std::uint32_t *numVertices*, std::uint32_t *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. |

**Note**

The parameter `firstVertex` modifies the vertex ID within the shader pipeline differently for `SV_Vertex↩ID` in HLSL and `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan), due to rendering API differences. The system value `SV_VertexID` in HLSL will always start with zero, but the system value `gl_↩VertexID` in GLSL (or `gl_VertexIndex` for Vulkan) will start with the value of `firstVertex`.

#### 10.19.3.13 virtual void LLGL::CommandBuffer::DrawIndexed ( std::uint32_t *numIndices*, std::uint32_t *firstIndex* ) [pure virtual]

**See also**

DrawIndexed(std::uint32_t, std::uint32_t, std::int32_t)

#### 10.19.3.14 virtual void LLGL::CommandBuffer::DrawIndexed ( std::uint32_t *numIndices*, std::uint32_t *firstIndex*, std::int32_t *vertexOffset* ) [pure virtual]

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

**Parameters**

| | | |
|---|---|---|
| in | *numIndices* | Specifies the number of indices 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. |

#### 10.19.3.15 virtual void LLGL::CommandBuffer::DrawIndexedIndirect ( Buffer & *buffer*, std::uint64_t *offset* ) [pure virtual]

Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag. |
| in | *offset* | Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4. |

**See also**

DrawIndexedIndirectArguments

#### 10.19.3.16 virtual void LLGL::CommandBuffer::DrawIndexedIndirect ( Buffer & *buffer*, std::uint64_t *offset*, std::uint32_t *numCommands*, std::uint32_t *stride* ) [pure virtual]

Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag. |
| in | *offset* | Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4. |
| in | *numCommands* | Specifies the number of draw commands that are to be taken from the argument buffer. |
| in | *stride* | Specifies the stride (in bytes) betweeen consecutive sets of arguments, which is commonly greater than or euqal to `sizeof(DrawIndexedIndirectArguments)`. This stride must be a multiple of 4. |

**Remarks**

This is also known as a "multi draw command" which is only natively supported by OpenGL and Vulkan. For other rendering APIs, the recording of multiple draw commands is emulated with a simple loop, which is equivalent to the following example:

```
while (numCommands-- > 0)
{
    DrawIndexedIndirect(buffer, offset);
    offset += stride;
}
```

**See also**

DrawIndexedIndirectArguments

#### 10.19.3.17 virtual void LLGL::CommandBuffer::DrawIndexedInstanced ( std::uint32_t *numIndices*, std::uint32_t *numInstances*, std::uint32_t *firstIndex* ) [pure virtual]

**See also**

DrawIndexedInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::int32_t, std::uint32_t)

#### 10.19.3.18 virtual void LLGL::CommandBuffer::DrawIndexedInstanced ( std::uint32_t *numIndices*, std::uint32_t *numInstances*, std::uint32_t *firstIndex*, std::int32_t *vertexOffset* ) [pure virtual]

**See also**

DrawIndexedInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::int32_t, std::uint32_t)

#### 10.19.3.19 virtual void LLGL::CommandBuffer::DrawIndexedInstanced ( std::uint32_t *numIndices*, std::uint32_t *numInstances*, std::uint32_t *firstIndex*, std::int32_t *vertexOffset*, std::uint32_t *firstInstance* ) [pure virtual]

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

**Parameters**

| | | |
|---|---|---|
| in | *numIndices* | Specifies the number of indices 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 | *firstInstance* | Specifies the zero-based offset of the first instance. |

**Note**

The parameter `firstInstance` modifies the instance ID within the shader pipeline differently for `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan), due to rendering API differences. The system value `SV_InstanceID` in HLSL will always start with zero, but the system value `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan) will start with the value of `firstInstance`.

**See also**

RenderingFeatures::hasInstancing
RenderingFeatures::hasOffsetInstancing

#### 10.19.3.20 virtual void LLGL::CommandBuffer::DrawIndirect ( Buffer & *buffer*, std::uint64_t *offset* ) [pure virtual]

Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag. |
| in | *offset* | Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4. |

**See also**

DrawIndirectArguments
RenderingFeatures::hasIndirectDrawing

#### 10.19.3.21 virtual void LLGL::CommandBuffer::DrawIndirect ( Buffer & *buffer*, std::uint64_t *offset*, std::uint32_t *numCommands*, std::uint32_t *stride* ) [pure virtual]

Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag. |

**Parameters**

| | | |
|---|---|---|
| in | *offset* | Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4. |
| in | *numCommands* | Specifies the number of draw commands that are to be taken from the argument buffer. |
| in | *stride* | Specifies the stride (in bytes) betweeen consecutive sets of arguments, which is commonly greater than or euqal to `sizeof(DrawIndirectArguments)`. This stride must be a multiple of 4. |

**Remarks**

This is also known as a "multi draw command" which is only natively supported by OpenGL and Vulkan. For other rendering APIs, the recording of multiple draw commands is emulated with a simple loop, which is equivalent to the following example:

```
while (numCommands-- > 0)
{
    DrawIndirect(buffer, offset);
    offset += stride;
}
```

**See also**

DrawIndirectArguments
RenderingFeatures::hasIndirectDrawing

#### 10.19.3.22 virtual void LLGL::CommandBuffer::DrawInstanced ( std::uint32_t *numVertices*, std::uint32_t *firstVertex*, std::uint32_t *numInstances* ) [pure virtual]

**See also**

DrawInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::uint32_t)

#### 10.19.3.23 virtual void LLGL::CommandBuffer::DrawInstanced ( std::uint32_t *numVertices*, std::uint32_t *firstVertex*, std::uint32_t *numInstances*, std::uint32_t *firstInstance* ) [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 | *firstInstance* | Specifies the zero-based offset of the first instance. |

**Note**

The parameter `firstVertex` modifies the vertex ID within the shader pipeline differently for `SV_VertexID` in HLSL and `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan), due to rendering API differences. The system value `SV_VertexID` in HLSL will always start with zero, but the system value `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan) will start with the value of `firstVertex`. The

same holds true for the parameter `firstInstance` and the system values `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan).

**See also**

RenderingFeatures::hasInstancing
RenderingFeatures::hasOffsetInstancing

#### 10.19.3.24 virtual void LLGL::CommandBuffer::End ( ) [pure virtual]

Ends the encoding (also referred to as "recording") of this command buffer.

**Remarks**

After this call, the command buffer can be submitted to the CommandQueue or executed by a primary command buffer.

**See also**

Begin
Execute
CommandQueue::Submit(CommandBuffer&)

#### 10.19.3.25 virtual void LLGL::CommandBuffer::EndQuery ( QueryHeap & *queryHeap*, std::uint32_t *query* = 0 ) [pure virtual]

Ends the specified query.

**See also**

BeginQuery

#### 10.19.3.26 virtual void LLGL::CommandBuffer::EndRenderCondition ( ) [pure virtual]

Ends the current render condition.

**See also**

BeginRenderCondition

#### 10.19.3.27 virtual void LLGL::CommandBuffer::EndRenderPass ( ) [pure virtual]

Ends the current render pass.

**See also**

BeginRenderPass

#### 10.19.3.28 virtual void LLGL::CommandBuffer::EndStreamOutput ( ) [pure virtual]

Ends the current stream-output.

**See also**

BeginStreamOutput

#### 10.19.3.29 virtual void LLGL::CommandBuffer::Execute ( CommandBuffer & *deferredCommandBuffer* ) [pure virtual]

Executes the specified deferred command buffer.

**Parameters**

| | | |
|---|---|---|
| in | *deferredCommandBuffer* | Specifies the deferred command buffer which is meant to be executed. This command buffer must have been created with the flag CommandBufferFlags::DeferredSubmit. |

**Remarks**

This function can only be used by primary command buffers, i.e. command buffers that have not been created with the flag CommandBufferFlags::DeferredSubmit.

**See also**

CommandBufferFlags

**Todo** Incomplete for: D3D12, Vulkan, Metal.

#### 10.19.3.30 virtual void LLGL::CommandBuffer::GenerateMips ( Texture & *texture* ) [pure virtual]

Generates all MIP-maps for the specified texture.

**Parameters**

| | | |
|---|---|---|
| in,out | *texture* | Specifies the texture whose MIP-maps are to be generated. This texture must have been created with the BindFlags::ColorAttachment flag. |

**Remarks**

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

**See also**

GenerateMips(Texture&, const TextureSubresource&)

#### 10.19.3.31 virtual void LLGL::CommandBuffer::GenerateMips ( Texture & *texture*, const TextureSubresource & *subresource* ) [pure virtual]

Generates the specified range of MIP-maps for the specified texture.

**Parameters**

| | | |
|---|---|---|
| in,out | *texture* | Specifies the texture whose MIP-maps are to be generated. This texture must have been created with the BindFlags::ColorAttachment flag. |
| in | *subresource* | Specifies the texture subresource, i.e. the range of MIP-maps that are to be updated. |

**Remarks**

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.
This function guarantees to generate only the MIP-maps in the specified range (specified by `subresource`). However, this function may introduce a performance penalty compared to generating MIP-maps using a texture view. It is therefore recommended to use this function only if the range of MIP-maps is significantly smaller than the entire MIP chain, e.g. only a single slice of a large 2D array texture, and use the primary `GenerateMips` function otherwise.

**See also**

GenerateMips(Texture&)

#### 10.19.3.32 virtual void LLGL::CommandBuffer::PopDebugGroup ( ) [pure virtual]

**See also**

PushDebugGroup

#### 10.19.3.33 virtual void LLGL::CommandBuffer::PushDebugGroup ( const char * *name* ) [pure virtual]

Pushes the specified name onto a stack of group strings that is used for debug reports.

**Parameters**

| | | |
|---|---|---|
| in | *name* | Pointer to a null terminated string that specifies the name. This **must not** be null! |

**Remarks**

Here is a usage example:

```
myCmdBuffer->PushDebugGroup("Shadow Map Pass");
myCmdBuffer->BeginRenderPass(...);
// render shadow map ...
myCmdBuffer->EndRenderPass();
myCmdBuffer->PopDebugGroup();

myCmdBuffer->PushDebugGroup("Final Scene Pass");
myCmdBuffer->BeginRenderPass(...);
```

```
// render final scene ...
myCmdBuffer->EndRenderPass();
myCmdBuffer->PopDebugGroup();
```

**Note**

Only supported in debug mode or when the debug layer is enabled. Otherwise, the function has no effect.

**See also**

PopDebugGroup
RenderSystemChild::SetName

#### 10.19.3.34 virtual void LLGL::CommandBuffer::ResetResourceSlots ( const ResourceType *resourceType*, std::uint32_t *firstSlot*, std::uint32_t *numSlots*, long *bindFlags*, long *stageFlags* = StageFlags::AllStages ) [pure virtual]

Resets the binding slots for the specified resources.

**Remarks**

This should be called when a resource is currently bound as shader output and will be bound as shader input for the next draw or compute commands.

**Parameters**

| | | |
|---|---|---|
| in | *resourceType* | Specifies the type of resources to unbind. |
| in | *firstSlot* | Specifies the first binding slot beginning with zero. This must be zero for the following resource types: ResourceType::IndexBuffer, ResourceType::StreamOutputBuffer. |
| in | *numSlots* | Specifies the number of binding slots to reset. If this is zero, the function has no effect. |
| in | *bindFlags* | Specifies which kind of binding slots to reset. This can be a bitwise OR combinations of the BindFlags entries. To reset a vertex buffer slot for instance, it must contain the BindFlags::VertexBuffer flag. |
| in | *stageFlags* | Specifies which shader stages are affected. This can be a bitwise OR combination of the StageFlags entries. By default StageFlags::AllStages. |

**Remarks**

If direct resource binding is not supported by the render system, this function has no effect.

**Note**

Only supported with: OpenGL, Direct3D 11, Metal.

**See also**

BindFlags
StageFlags
RenderingFeatures::hasDirectResourceBinding

#### 10.19.3.35 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).

**Note**

This state is guaranteed to be persistent and can be used outside of command buffer encoding.

**See also**

Clear

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

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

**Note**

This state is guaranteed to be persistent and can be used outside of command buffer encoding.

**See also**

Clear

#### 10.19.3.37 virtual void LLGL::CommandBuffer::SetClearStencil ( std::uint32_t *stencil* ) [pure virtual]

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

**Parameters**

| | | |
|---|---|---|
| in | *stencil* | Specifies the value to clear the stencil buffer. This value is masked with `2^m-1`, where `m` is the number of bits in the stencil buffer (e.g. `stencil & 0xFF` for an 8-bit stencil buffer). |

**Note**

This state is guaranteed to be persistent and can be used outside of command buffer encoding.

**See also**

Clear

#### 10.19.3.38 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

#### 10.19.3.39 virtual void LLGL::CommandBuffer::SetComputeResourceHeap ( ResourceHeap & *resourceHeap*, std::uint32_t *firstSet* = 0 ) [pure virtual]

Binds the specified resource heap to the compute pipeline.

**Parameters**

| | | |
|---|---|---|
| in | *resourceHeap* | Specifies the resource heap that contains all shader resources that will be bound to the shader pipeline. |
| in | *firstSet* | Specifies the set number of the first layout descriptor. |

**Remarks**

This may invalidate the previously bound resource heap for both the graphics and compute pipeline.

**Note**

Parameter 'firstSet' is only supported with: Vulkan.

#### 10.19.3.40 virtual void LLGL::CommandBuffer::SetGraphicsAPIDependentState ( const void * *stateDesc*, std::size_t *stateDescSize* ) [pure virtual]

Sets a few low-level graphics API dependent states.

**Parameters**

| | | |
|---|---|---|
| in | *stateDesc* | Specifies a pointer to the renderer spcific state descriptor. If this is a null pointer, the function has no effect. |
| in | *stateDescSize* | Specifies the size (in bytes) of the renderer spcific state descriptor structure. If this value is not equal to the state descriptor structure that is required for the respective renderer, the function has no effect. |

**Remarks**

This can be used to work around several differences between the low-level graphics APIs, e.g. for a uniform render target behavior between OpenGL and Direct3D. Here is a usage example:

```
LLGL::OpenGLDependentStateDescriptor myOpenGLStateDesc;
myOpenGLStateDesc.invertFrontFace = true;
myCmdBuffer->SetGraphicsAPIDependentState(&myOpenGLStateDesc, sizeof(myOpenGLStateDesc));
```

**Note**

Invalid arguments are ignored by this function silently (except for corrupted pointers).

**See also**

OpenGLDependentStateDescriptor

#### 10.19.3.41 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, and a graphics pipeline can only be set inside a render pass.

```
// First set render target
myCmdBuffer->BeginRenderPass(...);
{
    // Then set graphics pipeline
    myCmdBuffer->SetGraphicsPipeline(...);

    // Then perform drawing operations
    myCmdBuffer->SetGraphicsResourceHeap(...);
    myCmdBuffer->Draw(...);
}
myCmdBuffer->EndRenderPass();
```

**See also**

RenderSystem::CreateGraphicsPipeline

#### 10.19.3.42 virtual void LLGL::CommandBuffer::SetGraphicsResourceHeap ( ResourceHeap & *resourceHeap*, std::uint32_t *firstSet* = 0 ) [pure virtual]

Binds the specified resource heap to the graphics pipeline.

**Parameters**

| | | |
|---|---|---|
| in | *resourceHeap* | Specifies the resource heap that contains all shader resources that will be bound to the shader pipeline. |
| in | *firstSet* | Specifies the set number of the first layout descriptor. |

**Remarks**

This may invalidate the previously bound resource heap for both the graphics and compute pipeline.

**Note**

Parameter 'firstSet' is only supported with: Vulkan.

#### 10.19.3.43 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 binding flag BindFlags::IndexBuffer and its content **must not** be uninitialized. |

**Remarks**

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

**See also**

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
DrawIndexed
DrawIndexedInstanced

#### 10.19.3.44 virtual void LLGL::CommandBuffer::SetIndexBuffer ( Buffer & *buffer*, const Format *format*, std::uint64_t *offset* = 0 ) [pure virtual]

Sets the active index buffer for subsequent drawing operations with a dynamic format and optional buffer offset.

**Parameters**

| | | |
|---|---|---|
| in | *buffer* | Specifies the index buffer to set. This buffer must have been created with the binding flag BindFlags::IndexBuffer and its content **must not** be uninitialized. |
| in | *format* | Specifies the format of each index in the buffer. This must be either Format::R16UInt or Format::R32UInt. |
| in | *offset* | Specifies an optional offset (in bytes) where to start reading the index buffer. By default 0. This has the same effect as setting the `firstIndex` argument in any `DrawIndexed` or `DrawIndexedInstanced` function, except that this offset is byte aligned. |

**Remarks**

The alternative version of this function uses merely the index format that was specified when the buffer was created.

**See also**

BufferDescriptor::IndexBuffer::format
SetIndexBuffer(Buffer&)

#### 10.19.3.45 virtual void LLGL::CommandBuffer::SetResource ( Resource & *resource*, std::uint32_t *slot*, long *bindFlags*, long *stageFlags* = StageFlags::AllStages ) [pure virtual]

Sets the specified resource to a binding slot.

**Parameters**

| | | |
|---|---|---|
| in | *resource* | Specifies the resource to set. |
| in | *slot* | Specifies the slot index where to put the resource. |
| in | *bindFlags* | Specifies to types of binding points for this resource. This can be a bitwise OR combinations of the BindFlags entries. |
| in | *stageFlags* | Specifies at which shader stages the resource is to be set. This can be a bitwise OR combinations of the StageFlags entries. By default StageFlags::AllStages. |

**Remarks**

This function is only supported with the older graphics APIs and only available for convenience. For best performance and complete renderer independence use SetGraphicsResourceHeap and SetComputeResource↩Heap.
The following example binds a constant buffer and texture resource to the fragment shader stage:

```
myCmdBuffer->SetResource(*myConstantBuffer, 1, LLGL::BindFlags::ConstantBuffer
      , LLGL::StageFlags::FragmentStage);
myCmdBuffer->SetResource(*myTexture,        2, LLGL::BindFlags::Sampled,
      LLGL::StageFlags::FragmentStage);
```

If direct resource binding is not supported by the render system, this function has no effect.

**Note**

Only supported with: OpenGL, Direct3D 11, Metal.

**See also**

RenderingFeatures::hasDirectResourceBinding

#### 10.19.3.46 virtual void LLGL::CommandBuffer::SetScissor ( const Scissor & *scissor* ) [pure virtual]

Sets a single scissor rectangle.

**Remarks**

Similar to SetScissors but only a single scissor rectangle is set.

**See also**

SetScissors

#### 10.19.3.47 virtual void LLGL::CommandBuffer::SetScissors ( std::uint32_t *numScissors*, const Scissor * *scissors* ) [pure virtual]

Sets an array of scissor rectangles, but only if the scissor test was enabled in the previously set graphics pipeline (otherwise, this function has no effect).

**Parameters**

| | | |
|---|---|---|
| in | *numScissors* | Specifies the number of scissor rectangles to set. |
| in | *scissors* | 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.screenSpaceOriginLower↩Left` is `false`, the origin of each scissor rectangle is on the upper-left (like for all other render systems). If `stateOpenGL.screenSpaceOriginLowerLeft` is `true`, the origin of each scissor rectangle is on the lower-left.

**See also**

SetGraphicsAPIDependentState
RasterizerDescriptor::scissorTestEnabled

#### 10.19.3.48 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 binding flag BindFlags::StreamOutputBuffer. |

**See also**

RenderSystem::MapBuffer
RenderSystem::UnmapBuffer

#### 10.19.3.49 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

#### 10.19.3.50 virtual void LLGL::CommandBuffer::SetUniform ( UniformLocation *location*, const void * *data*, std::uint32_t *dataSize* ) [pure virtual]

Sets the value of a single uniform (a.k.a. shader constant) in the shader program that is currently bound.

**Parameters**

| | | |
|---|---|---|
| in | *location* | Specifies the location of the uniform. |
| in | *data* | Raw pointer to the data that is to be copied to the uniform. |
| in | *dataSize* | Specifies the size (in bytes) of the input buffer `data`. This must be a multiple of 4. |

**Remarks**

This function must only be called after a graphics or compute pipeline has been set.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 12.

**See also**

ShaderProgram::FindUniformLocation
ShaderProgram::Reflect
SetGraphicsPipeline
SetComputePipeline

#### 10.19.3.51 virtual void LLGL::CommandBuffer::SetUniforms ( UniformLocation *location*, std::uint32_t *count*, const void * *data*, std::uint32_t *dataSize* ) [pure virtual]

Sets the value of multiple uniforms (a.k.a. shader constants) in the shader program that is currently bound.

**Parameters**

| | | |
|---|---|---|
| in | *location* | Specifies the location of the first uniform. |
| in | *count* | Specifies the number of uniforms that are meant to be updated. |
| in | *data* | Raw pointer to the data that is to be copied to the uniforms. |
| in | *dataSize* | Specifies the size (in bytes) of the input buffer `data`. This must be a multiple of 4. |

**Remarks**

This function must only be called after a graphics or compute pipeline has been set. The order of uniforms that come after the first one can be determined by the ShaderReflection::uniform container returned by Shader↩Program::Reflect.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 12.

**See also**

ShaderProgram::FindUniformLocation
ShaderProgram::Reflect
SetGraphicsPipeline
SetComputePipeline

#### 10.19.3.52 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 binding flag BindFlags::VertexBuffer and its content **must not** be uninitialized. |

**See also**

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
SetVertexBufferArray

#### 10.19.3.53 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

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

Sets a single viewport.

**Remarks**

Similar to SetViewports but only a single viewport is set.

**See also**

SetViewports

#### 10.19.3.55 virtual void LLGL::CommandBuffer::SetViewports ( std::uint32_t *numViewports*, const Viewport * *viewports* ) [pure virtual]

Sets an array of viewports.

**Parameters**

| | | |
|---|---|---|
| in | *numViewports* | Specifies the number of viewports to set. Most render system have a limit of 16 viewports. |
| in | *viewports* | 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 OpenGLDependentStateDescriptor::originLowerLeft is `false`, the origin of each viewport is on the upper-left (like for all other render systems). If OpenGL↩DependentStateDescriptor::originLowerLeft is `true`, the origin of each viewport is on the lower-left.

**See also**

SetGraphicsAPIDependentState
RenderingLimits::maxViewports

#### 10.19.3.56 virtual void LLGL::CommandBuffer::UpdateBuffer ( Buffer & *dstBuffer*, std::uint64_t *dstOffset*, const void * *data*, std::uint16_t *dataSize* ) [pure virtual]

Updates the data of the specified buffer during encoding the command buffer.

**Parameters**

| | | |
|---|---|---|
| in | *dstBuffer* | Specifies the destination buffer whose data is to be updated. |
| in | *dstOffset* | Specifies the destination offset (in bytes) at which the buffer is to be updated. This offset plus the data block size (i.e. `dstOffset + dataSize`) must be less than or equal to the size of the buffer. |
| 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 is limited to 2^16 = 65536 bytes, because it may be written to the command buffer itself before it is copied to the destination buffer (depending on the backend). |

**Remarks**

To update buffers larger than 65536 bytes, use RenderSystem::WriteBuffer or RenderSystem::MapBuffer. For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

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

- CommandBuffer.h

## 10.20 LLGL::CommandBufferDescriptor Struct Reference

Command buffer descriptor structure.

```
#include <CommandBufferFlags.h>
```

### Public Attributes

- long flags = 0

  *Specifies the creation flags for the command buffer. By default 0.*
- std::uint32_t numNativeBuffers = 2

  *Specifies the number of internal native command buffers. By default 2.*

### 10.20.1 Detailed Description

Command buffer descriptor structure.

**See also**

RenderSystem::CreateCommandBuffer

### 10.20.2 Member Data Documentation

#### 10.20.2.1 long LLGL::CommandBufferDescriptor::flags = 0

Specifies the creation flags for the command buffer. By default 0.

**Remarks**

If no flags are specified (i.e. the default value), the command buffer must be encoded again after it has been submitted to the command queue.

**See also**

CommandBufferFlags

#### 10.20.2.2 std::uint32_t LLGL::CommandBufferDescriptor::numNativeBuffers = 2

Specifies the number of internal native command buffers. By default 2.

**Remarks**

This is only a hint to the framework, since not all rendering APIs support command buffers natively. For those that do, however, this member specifies how many native command buffers are to be allocated internally. These native command buffers are then switched everytime encoding begins with the CommandBuffer::Begin function. The benefit of having multiple native command buffers is that it reduces the time the GPU is idle because it waits for a command buffer to be completed before it can be reused.

**See also**

CommandBuffer::Begin

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

- CommandBufferFlags.h

## 10.21 LLGL::CommandBufferFlags Struct Reference

Command buffer creation flags.

```
#include <CommandBufferFlags.h>
```

### Public Types

- enum { DeferredSubmit = (1 << 0), MultiSubmit = (1 << 1) }

### 10.21.1 Detailed Description

Command buffer creation flags.

**See also**

CommandBufferDescriptor::flags

### 10.21.2 Member Enumeration Documentation

#### 10.21.2.1 anonymous enum

**Enumerator**

***DeferredSubmit*** Specifies that the encoded command buffer will be submitted as a secondary command buffer.

**Remarks**

If this is specified, the command buffer must be submitted using the `Execute` function of a primary command buffer.

**See also**

CommandBuffer::Execute

***MultiSubmit*** Specifies that the encoded command buffer can be submitted multiple times.

**Remarks**

If this is not specified, the command buffer must be encoded again after it has been submitted to the command queue.

**See also**

CommandQueue::Submit(CommandBuffer&)

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

- CommandBufferFlags.h

## 10.22 LLGL::CommandQueue Class Reference

Command queue interface.

```
#include <CommandQueue.h>
```

Inheritance diagram for LLGL::CommandQueue:

### Public Member Functions

- virtual void Submit (CommandBuffer &commandBuffer)=0

  *Submits the specified command buffer to the command queue.*
- virtual bool QueryResult (QueryHeap &queryHeap, std::uint32_t firstQuery, std::uint32_t numQueries, void *data, std::size_t dataSize)=0

  *Retrieves the result of the specified query objects.*
- virtual void Submit (Fence &fence)=0

  *Submits the specified fence to the command queue for CPU/GPU synchronization.*
- virtual bool WaitFence (Fence &fence, std::uint64_t timeout)=0

  *Blocks the CPU execution until the specified fence has been signaled.*
- virtual void WaitIdle ()=0

  *Blocks the CPU execution until the entire GPU command queue has been completed.*

### Protected Member Functions

- CommandQueue ()=default

### 10.22.1 Detailed Description

Command queue interface.

#### Remarks

This class is mainly used for modern rendering APIs (such as Direct3D 12 and Vulkan) to submit one ore more command buffers (or command lists) into the command queue. For older rendering APIs (such as Direct3D 11 and OpenGL) submitting a command buffer has no effect. It also provides the functionality to submit small sized objects such as fences into the command queue.

### 10.22.2 Constructor & Destructor Documentation

#### 10.22.2.1 LLGL::CommandQueue::CommandQueue ( ) [protected],[default]

### 10.22.3 Member Function Documentation

#### 10.22.3.1 virtual bool LLGL::CommandQueue::QueryResult ( QueryHeap & *queryHeap*, std::uint32_t *firstQuery*, std::uint32_t *numQueries*, void * *data*, std::size_t *dataSize* ) [pure virtual]

Retrieves the result of the specified query objects.

**Parameters**

| | | |
|---|---|---|
| in | *queryHeap* | Specifies the query heap. |
| in | *firstQuery* | Specifies the zero-based index of the first query within the heap. This must be in the half-open range [0, QueryHeapDescriptor::numQueries). |
| in | *numQueries* | Specifies the number of queries to retrieve the result from. This must be less than or equal to (QueryHeapDescriptor::numQueries - firstQuery) and it must not be zero. |
| out | *data* | Specifies the pointer to the output data. This must be a valid pointer to an array of `numQueries` entries. The array entries must have one of the following types:<br>• std::uint32_t<br>• std::uint64_t<br>• QueryPipelineStatistics If the function return false, the content of this array is undefined. |
| in | *dataSize* | Specifies the size (in bytes) of the output data. This must be equal to `numQueries * sizeof(T)` where `T` is the type of the query entries described above. |

**Returns**

True, if all results are available. Otherwise, the results are (partially) unavailable and the content of the output data is undefined.

**Remarks**

Here is a usage example:

```
// Get results of 10 occlusion queries
std::uint64_t occlusionQueryResults[10] = {};
myCmdQueue->QueryResult(*myOcclusionQuery, 0, 10, occlusionQueryResults, sizeof(occlusionQueryResults));

// Get result of a pipeline statistics query
LLGL::QueryPipelineStatistics stats;
myCmdQueue->QueryResult(*myPipelineStatsQuery, 0, 1, &stats, sizeof(stats));
```

#### 10.22.3.2 virtual void LLGL::CommandQueue::Submit ( CommandBuffer & *commandBuffer* ) [pure virtual]

Submits the specified command buffer to the command queue.

**Remarks**

This must only be called with a command buffer that has already been encoded via the `Begin` and `End` functions:

```
myCmdBuffer->Begin();
// Encode/record command buffer ...
myCmdBuffer->End();
myCmdQueue->Submit(*myCmdBuffer);
```

**See also**

CommandBuffer::Begin
CommandBuffer::End
Submit(std::uint32_t, CommandBuffer* const *)

#### 10.22.3.3 virtual void LLGL::CommandQueue::Submit ( Fence & *fence* ) [pure virtual]

Submits the specified fence to the command queue for CPU/GPU synchronization.

#### 10.22.3.4 virtual bool LLGL::CommandQueue::WaitFence ( Fence & *fence*, std::uint64_t *timeout* ) [pure virtual]

Blocks the CPU execution until the specified fence has been signaled.

**Parameters**

| | | |
|---|---|---|
| in | *fence* | Specifies the fence for which the CPU needs to wait to be signaled. |
| in | *timeout* | Specifies the waiting timeout (in nanoseconds). |

**Returns**

True on success, or false if the fence has a timeout (in nanoseconds) or the device is lost.

**Remarks**

To wait for the completion of the entire GPU command queue, use 'WaitIdle'.

**See also**

WaitIdle

#### 10.22.3.5 virtual void LLGL::CommandQueue::WaitIdle ( ) [pure virtual]

Blocks the CPU execution until the entire GPU command queue has been completed.

**Remarks**

To wait for a specific point in the command queue, use fences. Waiting for the queue to be become idle is equivalent to submitting a fence and waiting for that fence to be signaled:

```
myCmdQueue->Submit(myFence);
myCmdQueue->WaitFence(myFence, ~0);
```

**See also**

WaitFence

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

- CommandQueue.h

## 10.23 LLGL::ComputePipeline Class Reference

Compute pipeline interface.

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

Inheritance diagram for LLGL::ComputePipeline:

### Additional Inherited Members

### 10.23.1 Detailed Description

Compute pipeline interface.

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

- ComputePipeline.h

## 10.24 LLGL::ComputePipelineDescriptor Struct Reference

Compute pipeline descriptor structure.

```
#include <ComputePipelineFlags.h>
```

### Public Attributes

- const ShaderProgram * shaderProgram = nullptr

  *Pointer to the shader program for the compute pipeline.*
- const PipelineLayout * pipelineLayout = nullptr

  *Pointer to an optional pipeline layout for the graphics pipeline.*

### 10.24.1 Detailed Description

Compute pipeline descriptor structure.

**See also**

RenderSystem::CreateComputePipeline

### 10.24.2 Member Data Documentation

#### 10.24.2.1 const PipelineLayout* LLGL::ComputePipelineDescriptor::pipelineLayout = nullptr

Pointer to an optional pipeline layout for the graphics pipeline.

**Remarks**

This layout determines at which slots buffer resources can be bound. This is ignored by render systems which do not support pipeline layouts.

**Note**

Only supported with: Vulkan, Direct3D 12

#### 10.24.2.2 const 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:

- ComputePipelineFlags.h

## 10.25 LLGL::CPUAccessFlags Struct Reference

CPU read/write access flag enumeration.

```
#include <ResourceFlags.h>
```

### Public Types

- enum { Read = (1 << 0), Write = (1 << 1), ReadWrite = (Read | Write) }

### 10.25.1 Detailed Description

CPU read/write access flag enumeration.

**See also**

BufferDescriptor::cpuAccessFlags
TextureDescriptor::cpuAccessFlags
CPUAccess

### 10.25.2 Member Enumeration Documentation

#### 10.25.2.1 anonymous enum

**Enumerator**

***Read*** Resource mapping with CPU read access is required.

**See also**

RenderSystem::MapBuffer

***Write*** Resource mapping with CPU write access is required.

**See also**

RenderSystem::MapBuffer

***ReadWrite***

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

- ResourceFlags.h

## 10.26 LLGL::DepthBiasDescriptor Struct Reference

Depth bias descriptor structure to control fragment depth values.

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

### Public Attributes

- float constantFactor = 0.0f

  *Specifies a scalar factor controlling the constant depth value added to each fragment. By default 0.0.*
- float slopeFactor = 0.0f

  *Specifies a scalar factor applied to a fragment's slope in depth bias calculations. By default 0.0.*
- float clamp = 0.0f

  *Specifies the maximum (or minimum) depth bias of a fragment. By default 0.0.*

### 10.26.1 Detailed Description

Depth bias descriptor structure to control fragment depth values.

### 10.26.2 Member Data Documentation

#### 10.26.2.1 float LLGL::DepthBiasDescriptor::clamp = 0.0f

Specifies the maximum (or minimum) depth bias of a fragment. By default 0.0.

**Note**

For OpenGL, this is only supported if the extension `GL_ARB_polygon_offset_clamp` is available (see https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_polygon_offset_clamp.txt).

#### 10.26.2.2 float LLGL::DepthBiasDescriptor::constantFactor = 0.0f

Specifies a scalar factor controlling the constant depth value added to each fragment. By default 0.0.

**Note**

The actual constant factor being added to each fragment is implementation dependent of the respective rendering API. Direct3D 12 for instance only considers the integral part.

#### 10.26.2.3 float LLGL::DepthBiasDescriptor::slopeFactor = 0.0f

Specifies a scalar factor applied to a fragment's slope in depth bias calculations. By default 0.0.

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

- GraphicsPipelineFlags.h

## 10.27 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.*

### 10.27.1 Detailed Description

Depth state descriptor structure.

**See also**

GraphicsPipelineDescriptor::depth

### 10.27.2 Member Data Documentation

#### 10.27.2.1 CompareOp LLGL::DepthDescriptor::compareOp = CompareOp::Less

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

#### 10.27.2.2 bool LLGL::DepthDescriptor::testEnabled = false

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

#### 10.27.2.3 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

## 10.28 LLGL::DispatchIndirectArguments Struct Reference

Format structure for the arguments of an indirect compute command.

```
#include <IndirectArguments.h>
```

### Public Attributes

- std::uint32_t numThreadGroups [3]

  *Number of thread groups in X, Y, and Z dimension.*

### 10.28.1 Detailed Description

Format structure for the arguments of an indirect compute command.

**Remarks**

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

**Note**

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

**See also**

CommandBuffer::DispatchIndirect
OpenGL counterpart `DispatchIndirectCommand`: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glDispatchComputeIndirect.xhtml
Vulkan counterpart `VkDispatchIndirectCommand`: https://www.khronos.org/registry/vulkan/specs1-extensions/man/html/VkDispatchIndirectCommand.html
Direct3D11 counterpart: N/A
Direct3D12 counterpart `D3D12_DISPATCH_ARGUMENTS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d12/ns-d3d12-d3d12_dispatch_arguments
Metal counterpart `MTLDispatchThreadgroupsIndirectArguments`: https://developer.apple.com/documentation/metal/mtldispatchthreadgroupsindirectarguments?language=objc

### 10.28.2 Member Data Documentation

#### 10.28.2.1 std::uint32_t LLGL::DispatchIndirectArguments::numThreadGroups[3]

Number of thread groups in X, Y, and Z dimension.

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

- IndirectArguments.h

## 10.29 LLGL::Display Class Reference

Display interface to query the attributes of all connected displays/monitors.

```
#include <Display.h>
```

Inheritance diagram for LLGL::Display:

### Public Member Functions

- virtual bool IsPrimary () const =0

  *Returns true if this is the primary display, as configured by the host system.*
- virtual std::wstring GetDeviceName () const =0

  *Returns the device name of this display. This may also be empty, if the platform does not support display names.*
- virtual Offset2D GetOffset () const =0

  *Returns the 2D offset relative to the primary display.*
- virtual bool ResetDisplayMode ()=0

  *Resets the display mode to its default value depending on the host system configuration.*
- virtual bool SetDisplayMode (const DisplayModeDescriptor &displayModeDesc)=0

  *Sets the new display mode for this display.*
- virtual DisplayModeDescriptor GetDisplayMode () const =0

  *Returns the current display mode of this display.*
- virtual std::vector< DisplayModeDescriptor > GetSupportedDisplayModes () const =0

  *Returns a list of all modes that are supported by this display.*

### Static Public Member Functions

- static std::vector< std::unique_ptr< Display > > InstantiateList ()

  *Instantiates a list of interfaces for all connected displays.*
- static std::unique_ptr< Display > InstantiatePrimary ()

  *Instantiates an interface for the primary display.*
- static bool ShowCursor (bool show)

  *Shows or hides the cursor for the running application from all displays.*
- static bool IsCursorShown ()

  *Returns true if the cursor is currently being shown on any display.*

### Static Protected Member Functions

- static void FinalizeDisplayModes (std::vector< DisplayModeDescriptor > &displayModeDescs)

  *Sorts the specified list of display modes as described in the GetSupportedDisplayModes function, and removes duplicate entries.*

### Additional Inherited Members

## 10.29.1 Detailed Description

Display interface to query the attributes of all connected displays/monitors.

#### Remarks

Here is an example to print the attributes of all displays:

```
auto myDisplayList = LLGL::Display::InstantiateList();
for (const auto& myDisplay : myDisplayList) {
    auto myDisplayOffset = myDisplay->GetOffset();
    auto myDisplayMode   = myDisplay->GetDisplayMode();
    std::wcout << L"Display: \"  << myDisplay->GetDeviceName() << L"\"" << std::endl;
    std::cout << "|-Primary = " << std::boolalpha << myDisplay->IsPrimary() << std::endl;
    std::cout << "|-X       = " << myDisplayOffset.x << std::endl;
    std::cout << "|-Y       = " << myDisplayOffset.y << std::endl;
    std::cout << "|-Width   = " << myDisplayMode.resolution.width << std::endl;
    std::cout << "|-Height  = " << myDisplayMode.resolution.height << std::endl;
    std::cout << "`-Hz      = " << myDisplayMode.refreshRate << std::endl;
}
```

### 10.29.2 Member Function Documentation

#### 10.29.2.1 static void LLGL::Display::FinalizeDisplayModes ( std::vector< DisplayModeDescriptor > & *displayModeDescs* ) [static],[protected]

Sorts the specified list of display modes as described in the GetSupportedDisplayModes function, and removes duplicate entries.

**See also**

GetSupportedDisplayModes

#### 10.29.2.2 virtual std::wstring LLGL::Display::GetDeviceName ( ) const [pure virtual]

Returns the device name of this display. This may also be empty, if the platform does not support display names.

#### 10.29.2.3 virtual DisplayModeDescriptor LLGL::Display::GetDisplayMode ( ) const [pure virtual]

Returns the current display mode of this display.

**See also**

SetDisplayMode

#### 10.29.2.4 virtual Offset2D LLGL::Display::GetOffset ( ) const [pure virtual]

Returns the 2D offset relative to the primary display.

**Remarks**

This can be used to position your windows accordingly to your displays.

**See also**

Window::SetPosition

#### 10.29.2.5 virtual std::vector<DisplayModeDescriptor> LLGL::Display::GetSupportedDisplayModes ( ) const [pure virtual]

Returns a list of all modes that are supported by this display.

**Remarks**

This list is always sorted in the following manner: The first sorting criterion is the number of pixels (resolution width times resolution height) in ascending order, and the second sorting criterion is the refresh rate in ascending order. To get only the currently active display mode, use GetDisplayMode.

**See also**

GetDisplayMode

#### 10.29.2.6 static std::vector<std::unique_ptr<Display> > LLGL::Display::InstantiateList ( ) [static]

Instantiates a list of interfaces for all connected displays.

#### 10.29.2.7 static std::unique_ptr<Display> LLGL::Display::InstantiatePrimary ( ) [static]

Instantiates an interface for the primary display.

**Returns**

Unique pointer to a Display instance that represents the primary display, or null on failure.

#### 10.29.2.8 static bool LLGL::Display::IsCursorShown ( ) [static]

Returns true if the cursor is currently being shown on any display.

**See also**

ShowCursor

#### 10.29.2.9 virtual bool LLGL::Display::IsPrimary ( ) const [pure virtual]

Returns true if this is the primary display, as configured by the host system.

#### 10.29.2.10 virtual bool LLGL::Display::ResetDisplayMode ( ) [pure virtual]

Resets the display mode to its default value depending on the host system configuration.

**See also**

SetDisplayMode

#### 10.29.2.11 virtual bool LLGL::Display::SetDisplayMode ( const DisplayModeDescriptor & *displayModeDesc* ) [pure virtual]

Sets the new display mode for this display.

**Parameters**

| | | |
|---|---|---|
| in | *displayModeDesc* | Specifies the descriptor of the new display mode. |

**Returns**

True on success, otherwise the specified display mode is not supported by this display and the function has no effect.

**See also**

GetDisplayMode

#### 10.29.2.12 static bool LLGL::Display::ShowCursor ( bool *show* ) [static]

Shows or hides the cursor for the running application from all displays.

**Parameters**

| | | |
|---|---|---|
| in | *show* | Specifies whether to show or hide the cursor. |

**Remarks**

In contrast to the Win32 API, this function only shows or hides the cursor, while the Win32 API function with the same name either increments or decrements an internal visibility counter for the cursor.

**Returns**

True on success, otherwise cursor visibility changes are not supported.

**See also**

IsCursorShown

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

- Display.h

## 10.30 LLGL::DisplayModeDescriptor Struct Reference

Display mode descriptor structure.

```
#include <DisplayFlags.h>
```

### Public Attributes

- Extent2D resolution

  *Display resolution (in pixels).*
- std::uint32_t refreshRate = 0

  *Display refresh rate (in Hz).*

### 10.30.1 Detailed Description

Display mode descriptor structure.

**Remarks**

Describes the attributes of a physical display. The counterpart for a virtual video mode is the VideoMode↩ Descriptor structure.

**See also**

VideoOutputDescriptor::displayModes
VideoModeDescriptor

### 10.30.2 Member Data Documentation

#### 10.30.2.1 std::uint32_t LLGL::DisplayModeDescriptor::refreshRate = 0

Display refresh rate (in Hz).

#### 10.30.2.2 Extent2D LLGL::DisplayModeDescriptor::resolution

Display resolution (in pixels).

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

- DisplayFlags.h

## 10.31 LLGL::DrawIndexedIndirectArguments Struct Reference

Format structure for the arguments of an indirect indexed draw command.

```
#include <IndirectArguments.h>
```

### Public Attributes

- std::uint32_t numIndices

  *Specifies the number of indices per instance.*
- std::uint32_t numInstances

  *Specifies the number of instances to draw.*
- std::uint32_t firstIndex

  *Specifies the zero-based offset of the first index from the index buffer.*
- std::int32_t vertexOffset

  *Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.*
- std::uint32_t firstInstance

  *Specifies the zero-based offset of the first instance.*

### 10.31.1 Detailed Description

Format structure for the arguments of an indirect indexed draw command.

#### Remarks

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

#### Note

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

#### See also

CommandBuffer::DrawIndexedIndirect
OpenGL counterpart `DrawElementsIndirectCommand`: https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_draw_indirect.txt
Vulkan counterpart `VkDrawIndexedIndirectCommand`: https://www.khronos.org/registry/vulkan/sp 1-extensions/man/html/VkDrawIndexedIndirectCommand.html
Direct3D11 counterpart `D3D11_DRAW_INDEXED_INSTANCED_INDIRECT_ARGS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_draw_indexed_instanced_indirect_args
Direct3D12 counterpart `D3D12_DRAW_INDEXED_ARGUMENTS`: https://docs.microsoft.com/en-us/windows/desktop/api/D3D12/ns-d3d12-d3d12_draw_indexed_arguments
Metal counterpart `MTLDrawIndexedPrimitivesIndirectArguments`: https://developer.apple.com/documentation/metal/mtldrawindexedprimitivesindirectarguments?language=obj

### 10.31.2 Member Data Documentation

#### 10.31.2.1 std::uint32_t LLGL::DrawIndexedIndirectArguments::firstIndex

Specifies the zero-based offset of the first index from the index buffer.

#### 10.31.2.2 std::uint32_t LLGL::DrawIndexedIndirectArguments::firstInstance

Specifies the zero-based offset of the first instance.

#### Note

This parameter modifies the instance ID within the shader pipeline differently for `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan), due to rendering API differences. The system value `SV_InstanceID` in HLSL will always start with zero, but the system value `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan) will start with the value of `firstInstance`.

#### 10.31.2.3 std::uint32_t LLGL::DrawIndexedIndirectArguments::numIndices

Specifies the number of indices per instance.

#### 10.31.2.4 std::uint32_t LLGL::DrawIndexedIndirectArguments::numInstances

Specifies the number of instances to draw.

#### 10.31.2.5 std::int32_t LLGL::DrawIndexedIndirectArguments::vertexOffset

Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.

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

- IndirectArguments.h

## 10.32 LLGL::DrawIndirectArguments Struct Reference

Format structure for the arguments of an indirect draw command.

`#include <IndirectArguments.h>`

### Public Attributes

- std::uint32_t numVertices

  *Specifies the number of vertices per instance.*
- std::uint32_t numInstances

  *Specifies the number of instances to draw.*
- std::uint32_t firstVertex

  *Specifies the zero-based offset of the first vertex from the vertex buffer.*
- std::uint32_t firstInstance

  *Specifies the zero-based offset of the first instance.*

### 10.32.1 Detailed Description

Format structure for the arguments of an indirect draw command.

**Remarks**

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

**Note**

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

**See also**

CommandBuffer::DrawIndirect
OpenGL counterpart `DrawArraysIndirectCommand`: https://www.khronos.org/registry/↩ OpenGL/extensions/ARB/ARB_draw_indirect.txt
Vulkan counterpart `VkDrawIndirectCommand`: https://www.khronos.org/registry/vulkan/specs/1.↩ 1-extensions/man/html/VkDrawIndirectCommand.html
Direct3D11 counterpart `D3D11_DRAW_INSTANCED_INDIRECT_ARGS`: https://docs.↩ microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_draw_↩ instanced_indirect_args
Direct3D12 counterpart `D3D12_DRAW_ARGUMENTS`: https://docs.microsoft.com/en-us/windows/deskto D3D12/ns-d3d12-d3d12_draw_arguments
Metal counterpart `MTLDrawPrimitivesIndirectArguments`: https://developer.↩ apple.com/documentation/metal/mtldrawprimitivesindirectarguments?language=objc

### 10.32.2 Member Data Documentation

#### 10.32.2.1 std::uint32_t LLGL::DrawIndirectArguments::firstInstance

Specifies the zero-based offset of the first instance.

**Note**

This parameter modifies the instance ID within the shader pipeline differently for `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan), due to rendering API differences. The system value `SV_InstanceID` in HLSL will always start with zero, but the system value `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan) will start with the value of `firstInstance`.

#### 10.32.2.2 std::uint32_t LLGL::DrawIndirectArguments::firstVertex

Specifies the zero-based offset of the first vertex from the vertex buffer.

**Note**

This parameter modifies the vertex ID within the shader pipeline differently for `SV_VertexID` in HLSL and `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan), due to rendering API differences. The system value `SV_VertexID` in HLSL will always start with zero, but the system value `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan) will start with the value of `firstVertex`.

#### 10.32.2.3 std::uint32_t LLGL::DrawIndirectArguments::numInstances

Specifies the number of instances to draw.

#### 10.32.2.4 std::uint32_t LLGL::DrawIndirectArguments::numVertices

Specifies the number of vertices per instance.

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

- IndirectArguments.h

## 10.33 LLGL::DrawPatchIndirectArgument Struct Reference

Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer.

```
#include <IndirectArguments.h>
```

### Public Attributes

- std::uint32_t numPatches

  *Specifies the number of patches per instance.*
- std::uint32_t numInstances

  *Specifies the number of instances to draw.*
- std::uint32_t firstPatch

  *Specifies the patch start index.*
- std::uint32_t firstInstance

  *Specifies the first instance to draw.*

## 10.33.1 Detailed Description

Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer.

#### Remarks

The Metal API treats the arguments for rendering tessellated patches differently, so this structure is required to fill the buffer that is used for the arguments of an indirect draw command on the Metal backend.

#### Note

Only supported with: Metal.

#### See also

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect
Metal counterpart `MTLDrawPatchIndirectArguments:` https://developer.apple.com/documentation/metal/mtldrawpatchindirectarguments?language=objc

## 10.33.2 Member Data Documentation

### 10.33.2.1 std::uint32_t LLGL::DrawPatchIndirectArgument::firstInstance

Specifies the first instance to draw.

### 10.33.2.2 std::uint32_t LLGL::DrawPatchIndirectArgument::firstPatch

Specifies the patch start index.

### 10.33.2.3 std::uint32_t LLGL::DrawPatchIndirectArgument::numInstances

Specifies the number of instances to draw.

#### 10.33.2.4 std::uint32_t LLGL::DrawPatchIndirectArgument::numPatches

Specifies the number of patches per instance.

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

- IndirectArguments.h

## 10.34 LLGL::DstImageDescriptor Struct Reference

Descriptor structure for an image that is used as destination for writing the image data.

```
#include <ImageFlags.h>
```

### Public Member Functions

- DstImageDescriptor ()=default
- DstImageDescriptor (const DstImageDescriptor &)=default
- DstImageDescriptor (ImageFormat format, DataType dataType, void *data, std::size_t dataSize)

  *Constructor to initialize all attributes.*

### 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. By default DataType::UInt8.*
- void * data = nullptr

  *Pointer to the read/write image data.*
- std::size_t dataSize = 0

  *Specifies the size (in bytes) of the image data. This is primarily used for compressed images and serves for robustness.*

### 10.34.1 Detailed Description

Descriptor structure for an image that is used as destination for writing the image data.

#### Remarks

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

#### See also

SrcImageDescriptor
ConvertImageBuffer
RenderSystem::ReadTexture

### 10.34.2 Constructor & Destructor Documentation

#### 10.34.2.1 LLGL::DstImageDescriptor::DstImageDescriptor ( ) [default]

#### 10.34.2.2 LLGL::DstImageDescriptor::DstImageDescriptor ( const DstImageDescriptor & ) [default]

#### 10.34.2.3 LLGL::DstImageDescriptor::DstImageDescriptor ( ImageFormat *format*, DataType *dataType*, void * *data*, std::size_t *dataSize* ) [inline]

Constructor to initialize all attributes.

### 10.34.3 Member Data Documentation

#### 10.34.3.1 void* LLGL::DstImageDescriptor::data = nullptr

Pointer to the read/write image data.

#### 10.34.3.2 std::size_t LLGL::DstImageDescriptor::dataSize = 0

Specifies the size (in bytes) of the image data. This is primarily used for compressed images and serves for robustness.

#### 10.34.3.3 DataType LLGL::DstImageDescriptor::dataType = DataType::UInt8

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

#### 10.34.3.4 ImageFormat LLGL::DstImageDescriptor::format = ImageFormat::RGBA

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

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

- ImageFlags.h

## 10.35 LLGL::Window::EventListener Class Reference

Interface for all window event listeners.

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

Inheritance diagram for LLGL::Window::EventListener:

### Protected Member Functions

- virtual void OnProcessEvents (Window &sender)

  *Send when the window events are about to be polled. The event listeners receive this event before the window itself.*
- virtual void OnQuit (Window &sender, bool &veto)

  *Send when the window is about to quit.*
- virtual void OnKeyDown (Window &sender, Key keyCode)

  *Send when a key (from keyboard or mouse) has been pushed.*
- virtual void OnKeyUp (Window &sender, Key keyCode)

  *Send when a key (from keyboard or mouse) has been released.*
- virtual void OnDoubleClick (Window &sender, Key keyCode)

  *Send when a mouse button has been double clicked.*
- virtual void OnChar (Window &sender, wchar_t chr)

  *Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.*
- virtual void OnWheelMotion (Window &sender, int motion)

  *Send when the mouse wheel has been moved on the sender window.*
- virtual void OnLocalMotion (Window &sender, const Offset2D &position)

  *Send when the mouse has been moved on the sender window.*
- virtual void OnGlobalMotion (Window &sender, const Offset2D &motion)

  *Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.*
- virtual void OnResize (Window &sender, const Extent2D &clientAreaSize)

  *Send when the window has been resized.*
- virtual void OnGetFocus (Window &sender)

  *Send when the window gets the keyboard focus.*
- virtual void OnLostFocus (Window &sender)

  *Send when the window lost the keyboard focus.*
- virtual void OnTimer (Window &sender, std::uint32_t timerID)

  *Send when the window received a timer event with the specified timer ID number.*

### Friends

- class Window

### Additional Inherited Members

### 10.35.1 Detailed Description

Interface for all window event listeners.

#### Remarks

This is a design exception compared to most other interfaces in LLGL, because it does not inherit from the NonCopyable interface. This is because there is no hidden implementation, so copying an instance of this interface is allowed.

#### See also

Input

### 10.35.2 Member Function Documentation

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

Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.

Reimplemented in LLGL::Input.

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

Send when a mouse button has been double clicked.

Reimplemented in LLGL::Input.

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

Send when the window gets the keyboard focus.

#### 10.35.2.4 virtual void LLGL::Window::EventListener::OnGlobalMotion ( Window & *sender,* const Offset2D & *motion* ) [protected], [virtual]

Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.

Reimplemented in LLGL::Input.

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

Send when a key (from keyboard or mouse) has been pushed.

Reimplemented in LLGL::Input.

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

Send when a key (from keyboard or mouse) has been released.

Reimplemented in LLGL::Input.

#### 10.35.2.7 virtual void LLGL::Window::EventListener::OnLocalMotion ( Window & *sender,* const Offset2D & *position* ) [protected], [virtual]

Send when the mouse has been moved on the sender window.

Reimplemented in LLGL::Input.

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

Send when the window lost the keyboard focus.

Reimplemented in LLGL::Input.

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

Send when the window events are about to be polled. The event listeners receive this event before the window itself.

**See also**

Window::OnProcessEvents

Reimplemented in LLGL::Input.

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

Send when the window is about to quit.

**Parameters**

| | | |
|---|---|---|
| out | *veto* | Specifies whether to cancel the quit event. If set to true, the call to `PostQuit` does not change the state `sender`, only the event listeners get informed. If no event listener sets this parameter to true, `sender` is set to the 'Quit' state and `ProcessEvents` returns false from then on. |

**See also**

Window::ProcessEvents

#### 10.35.2.11 virtual void LLGL::Window::EventListener::OnResize ( Window & *sender,* const Extent2D & *clientAreaSize* ) [protected], [virtual]

Send when the window has been resized.

#### 10.35.2.12 virtual void LLGL::Window::EventListener::OnTimer ( Window & *sender,* std::uint32_t *timerID* ) [protected], [virtual]

Send when the window received a timer event with the specified timer ID number.

**Note**

Only supported on: MS. Windows.

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

Send when the mouse wheel has been moved on the sender window.

Reimplemented in LLGL::Input.

### 10.35.3 Friends And Related Function Documentation

#### 10.35.3.1 friend class Window [friend]

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

- Window.h

## 10.36 LLGL::Canvas::EventListener Class Reference

Interface for all canvas event listeners.

```
#include <Canvas.h>
```

Inheritance diagram for LLGL::Canvas::EventListener:

### Protected Member Functions

- virtual void OnProcessEvents (Canvas &sender)

  *Send when the canvas events are about to be polled. The event listeners receive this event before the canvas itself.*
- virtual void OnQuit (Canvas &sender, bool &veto)

  *Send when the canvas is about to quit.*

### Friends

- class Canvas

### Additional Inherited Members

### 10.36.1 Detailed Description

Interface for all canvas event listeners.

### 10.36.2 Member Function Documentation

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

Send when the canvas events are about to be polled. The event listeners receive this event before the canvas itself.

**See also**

Canvas::OnProcessEvents

#### 10.36.2.2 virtual void LLGL::Canvas::EventListener::OnQuit ( Canvas & *sender,* bool & *veto* ) [protected], [virtual]

Send when the canvas is about to quit.

**Parameters**

| | | |
|---|---|---|
| out | *veto* | Specifies whether to cancel the quit event. If set to true, the call to `PostQuit` does not change the state `sender`, only the event listeners get informed. If no event listener sets this parameter to true, `sender` is set to the 'Quit' state and `ProcessEvents` returns false from then on. |

**See also**

Canvas::ProcessEvents

### 10.36.3 Friends And Related Function Documentation

#### 10.36.3.1 friend class Canvas [friend]

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

- Canvas.h

## 10.37 LLGL::Extent2D Struct Reference

2-Dimensional extent structure.

```
#include <Types.h>
```

### Public Member Functions

- Extent2D ()=default
- Extent2D (const Extent2D &)=default
- Extent2D (std::uint32_t width, std::uint32_t height)

### Public Attributes

- std::uint32_t width = 0

  *Extent X axis, i.e. width.*
- std::uint32_t height = 0

  *Extent Y axis, i.e. height.*

### 10.37.1 Detailed Description

2-Dimensional extent structure.

**Remarks**

Used for unsigned integral 2D extents (for sizes in window-space, screen-space, and texture-space).

### 10.37.2 Constructor & Destructor Documentation

#### 10.37.2.1 LLGL::Extent2D::Extent2D ( ) [default]

#### 10.37.2.2 LLGL::Extent2D::Extent2D ( const Extent2D & ) [default]

#### 10.37.2.3 LLGL::Extent2D::Extent2D ( std::uint32_t *width*, std::uint32_t *height* ) [inline]

### 10.37.3 Member Data Documentation

#### 10.37.3.1 std::uint32_t LLGL::Extent2D::height = 0

Extent Y axis, i.e. height.

#### 10.37.3.2 std::uint32_t LLGL::Extent2D::width = 0

Extent X axis, i.e. width.

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

- Types.h

## 10.38 LLGL::Extent3D Struct Reference

3-Dimensional extent structure.

```
#include <Types.h>
```

### Public Member Functions

- Extent3D ()=default
- Extent3D (const Extent3D &)=default
- Extent3D (std::uint32_t width, std::uint32_t height, std::uint32_t depth)

### Public Attributes

- std::uint32_t width = 0

  *Extent X axis, i.e. width.*
- std::uint32_t height = 0

  *Extent Y axis, i.e. height.*
- std::uint32_t depth = 0

  *Extent Z axis, i.e. depth.*

## 10.38.1 Detailed Description

3-Dimensional extent structure.

**Remarks**

Used for unsigned integral 3D extents (for sizes in texture-space).

## 10.38.2 Constructor & Destructor Documentation

### 10.38.2.1 LLGL::Extent3D::Extent3D ( ) [default]

### 10.38.2.2 LLGL::Extent3D::Extent3D ( const Extent3D & ) [default]

### 10.38.2.3 LLGL::Extent3D::Extent3D ( std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *depth* ) [inline]

## 10.38.3 Member Data Documentation

### 10.38.3.1 std::uint32_t LLGL::Extent3D::depth = 0

Extent Z axis, i.e. depth.

### 10.38.3.2 std::uint32_t LLGL::Extent3D::height = 0

Extent Y axis, i.e. height.

### 10.38.3.3 std::uint32_t LLGL::Extent3D::width = 0

Extent X axis, i.e. width.

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

- Types.h

## 10.39 LLGL::Fence Class Reference

Fence interface for CPU/GPU synchronization.

```
#include <Fence.h>
```

Inheritance diagram for LLGL::Fence:

### Additional Inherited Members

### 10.39.1 Detailed Description

Fence interface for CPU/GPU synchronization.

**See also**

RenderSystem::CreateFence
CommandQueue::Submit(Fence&)
CommandQueue::WaitFence

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

- Fence.h

## 10.40 LLGL::FormatDescriptor Struct Reference

Vertex and texture format descriptor structure.

```
#include <Format.h>
```

### Public Attributes

- std::uint16_t bitSize

  *Size (in bits) of the a pixel block or vertex attribute. This is 0 for invalid formats.*
- std::uint8_t blockWidth: 4

  *Width of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.*
- std::uint8_t blockHeight: 4

  *Height of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.*
- std::uint8_t components: 4

  *Number of components. This is either 1, 2, 3, or 4. This is 0 for invalid formats.*
- ImageFormat format

  *Image pixel format. This is ignored for depth-stencil and compressed formats.*
- DataType dataType

  *Color component data type. This is ignored for depth-stencil and compressed formats.*
- bool sRGB: 1

  *Specifies whether the format is in non-linear sRGB space (e.g. Format::RGBA8UNorm_sRGB).*
- bool compressed: 1

  *Specifies whether the format is compressed (e.g. Format::BC1RGBA).*
- bool depth: 1

  *Specifies whether the format has a depth component (e.g. Format::D16UNorm).*
- bool stencil: 1

  *Specifies whether the format has a stencil component (e.g. Format::D24UNormS8UInt).*
- bool normalized: 1

  *Specifies whether the format has normalized components (e.g. Format::R8UNorm or Format::D16UNorm).*
- bool packed: 1

  *Specifies whether the format is packed (e.g. Format::RGB10A2UNorm or Format::RGB9E5Float).*

## 10.40.1 Detailed Description

Vertex and texture format descriptor structure.

#### Remarks

Describes all attributes of a hardware format, e.g. bit size, color components, compression etc.

#### Note

This descriptor structure has no default initializers to fulfill the requirements of a plain-old-data (POD) structure.

#### See also

GetFormatDesc

## 10.40.2 Member Data Documentation

### 10.40.2.1 std::uint16_t LLGL::FormatDescriptor::bitSize

Size (in bits) of the a pixel block or vertex attribute. This is 0 for invalid formats.

#### 10.40.2.2 std::uint8_t LLGL::FormatDescriptor::blockHeight

Height of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.

#### 10.40.2.3 std::uint8_t LLGL::FormatDescriptor::blockWidth

Width of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.

#### 10.40.2.4 std::uint8_t LLGL::FormatDescriptor::components

Number of components. This is either 1, 2, 3, or 4. This is 0 for invalid formats.

#### 10.40.2.5 bool LLGL::FormatDescriptor::compressed

Specifies whether the format is compressed (e.g. Format::BC1RGBA).

#### 10.40.2.6 DataType LLGL::FormatDescriptor::dataType

Color component data type. This is ignored for depth-stencil and compressed formats.

**Remarks**

For depth-stencil formats, this data type always refers to the depth component, because stencil components are always in 8-bit unsigned integer format.

#### 10.40.2.7 bool LLGL::FormatDescriptor::depth

Specifies whether the format has a depth component (e.g. Format::D16UNorm).

#### 10.40.2.8 ImageFormat LLGL::FormatDescriptor::format

Image pixel format. This is ignored for depth-stencil and compressed formats.

#### 10.40.2.9 bool LLGL::FormatDescriptor::normalized

Specifies whether the format has normalized components (e.g. Format::R8UNorm or Format::D16UNorm).

#### 10.40.2.10 bool LLGL::FormatDescriptor::packed

Specifies whether the format is packed (e.g. Format::RGB10A2UNorm or Format::RGB9E5Float).

#### 10.40.2.11 bool LLGL::FormatDescriptor::sRGB

Specifies whether the format is in non-linear sRGB space (e.g. Format::RGBA8UNorm_sRGB).

#### 10.40.2.12 bool LLGL::FormatDescriptor::stencil

Specifies whether the format has a stencil component (e.g. Format::D24UNormS8UInt).

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

- Format.h

## 10.41 LLGL::FragmentAttribute Struct Reference

Fragment output attribute structure.

```
#include <FragmentAttribute.h>
```

### Public Attributes

- std::string name

  *Fragment attribute name (for GLSL) or semantic name (for HLSL).*
- Format format = Format::RGBA32Float

  *Fragment attribute format. By default Format::RGBA32Float.*
- std::uint32_t semanticIndex = 0

  *Specifies the semantic index when multiple color targets are specified.*
- SystemValue systemValue = SystemValue::Undefined

  *Specifies the system value type for this fragment attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.*

### 10.41.1 Detailed Description

Fragment output attribute structure.

**See also**

FragmentFormat::attributes
ShaderReflection::fragmentAttributes

### 10.41.2 Member Data Documentation

#### 10.41.2.1 Format LLGL::FragmentAttribute::format = Format::RGBA32Float

Fragment attribute format. By default Format::RGBA32Float.

#### 10.41.2.2 std::string LLGL::FragmentAttribute::name

Fragment attribute name (for GLSL) or semantic name (for HLSL).

#### 10.41.2.3 std::uint32_t LLGL::FragmentAttribute::semanticIndex = 0

Specifies the semantic index when multiple color targets are specified.

#### 10.41.2.4 SystemValue LLGL::FragmentAttribute::systemValue = SystemValue::Undefined

Specifies the system value type for this fragment attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.

**Remarks**

This can only be one the following values:

- SystemValue::Undefined
- SystemValue::Color
- SystemValue::Depth
- SystemValue::DepthGreater
- SystemValue::DepthLess
- SystemValue::Stencil

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

- FragmentAttribute.h

## 10.42 LLGL::FrameProfile Struct Reference

Profile of a rendered frame.

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

### Public Member Functions

- FrameProfile ()

  *Default constructor that initializes all counter values to zero.*
- void Clear ()

  *Clears all counter values.*
- void Accumulate (const FrameProfile &rhs)

  *Accumulates the specified profile with this profile.*

## Public Attributes

- union {
  struct {
    std::uint32_t mipMapsGenerations
      *Counter for all MIP-map generations.*
    std::uint32_t vertexBufferBindings
      *Counter for all vertex buffer and vertex buffer array bindings.*
    std::uint32_t indexBufferBindings
      *Counter for all index buffer bindings.*
    std::uint32_t streamOutputBufferBindings
      *Counter for all stream-output buffer and stream-output buffer array bindings.*
    std::uint32_t constantBufferBindings
      *Counter for all individual constant buffer bindings.*
    std::uint32_t sampledBufferBindings
      *Counter for all sampled buffer bindings (i.e. with BindFlags::Sampled flag).*
    std::uint32_t storageBufferBindings
      *Counter for all storage buffer bindings (i.e. with BindFlags::Storage flag).*
    std::uint32_t sampledTextureBindings
      *Counter for all sampled texture bindings (i.e. with BindFlags::Sampled flag).*
    std::uint32_t storageTextureBindings
      *Counter for all sampled texture bindings (i.e. with BindFlags::Storage flag).*
    std::uint32_t samplerBindings
      *Counter for all sampler-state bindings.*
    std::uint32_t graphicsResourceHeapBindings
      *Counter for all resource heap bindings on the graphics pipeline.*
    std::uint32_t computeResourceHeapBindings
      *Counter for all resource heap bindings on the compute pipeline.*
    std::uint32_t graphicsPipelineBindings
      *Counter for all graphics pipeline bindings.*
    std::uint32_t computePipelineBindings
      *Counter for all compute pipeline bindings.*
    std::uint32_t attachmentClears
      *Counter for all framebuffer attachment clear operations.*
    std::uint32_t bufferUpdates
      *Counter for all buffer updates during command encoding.*
    std::uint32_t bufferCopies
      *Counter for all buffer copies during command encoding.*
    std::uint32_t bufferWrites
      *Counter for all buffer write operations outside of command encoding.*
    std::uint32_t bufferReads
      *Counter for all buffer write operations outside of command encoding.*
    std::uint32_t bufferMappings
      *Counter for all buffer map/unmap operations outside of command encoding.*
    std::uint32_t textureCopies
      *Counter for all texture copies during command encoding.*
    std::uint32_t textureWrites
      *Counter for all texture write operations outside of command encoding.*
    std::uint32_t textureReads
      *Counter for all texture write operations outside of command encoding.*
    std::uint32_t textureMappings
      *Counter for all texture write operations outside of command encoding.*
    std::uint32_t renderPassSections
      *Counter for all command buffer sections that are enclosed by a call to* `BeginRenderPass` *and* `EndRenderPass`.
    std::uint32_t streamOutputSections
      *Counter for all command buffer sections that are enclosed by a call to* `BeginStreamOutput` *and* `EndStreamOut`
    std::uint32_t querySections
      *Counter for all command buffer sections that are enclosed by a call to* `BeginQuery` *and* `EndQuery`.
    std::uint32_t renderConditionSections

*Counter for all command buffer sections that are enclosed by a call to* `BeginRenderCondition` *and* `EndRender`

std::uint32_t drawCommands

*Counter for all draw commands.*

std::uint32_t dispatchCommands

*Counter for dispatch compute commands.*

std::uint32_t commandBufferSubmittions

*Counter for all command buffers that were submitted to the queue.*

std::uint32_t commandBufferEncodings

*Counter for all command buffer encodings that are enclosed by a call to* `Begin` *and* `End`.

std::uint32_t fenceSubmissions

*Counter for all fences that were submitted to the queue.*

}

std::uint32_t values [33]

*All proflile values as linear array.*

};

### 10.42.1 Detailed Description

Profile of a rendered frame.

**See also**

RenderingProfiler::NextFrame

### 10.42.2 Constructor & Destructor Documentation

#### 10.42.2.1 LLGL::FrameProfile::FrameProfile ( ) [inline]

Default constructor that initializes all counter values to zero.

### 10.42.3 Member Function Documentation

#### 10.42.3.1 void LLGL::FrameProfile::Accumulate ( const FrameProfile & *rhs* ) [inline]

Accumulates the specified profile with this profile.

#### 10.42.3.2 void LLGL::FrameProfile::Clear ( ) [inline]

Clears all counter values.

### 10.42.4 Member Data Documentation

#### 10.42.4.1 union { ... }

#### 10.42.4.2 std::uint32_t LLGL::FrameProfile::attachmentClears

Counter for all framebuffer attachment clear operations.

**See also**

CommandBuffer::Clear
CommandBuffer::ClearAttachments

#### 10.42.4.3 std::uint32_t LLGL::FrameProfile::bufferCopies

Counter for all buffer copies during command encoding.

**See also**

CommandBuffer::CopyBuffer

#### 10.42.4.4 std::uint32_t LLGL::FrameProfile::bufferMappings

Counter for all buffer map/unmap operations outside of command encoding.

**See also**

RenderSystem::MapBuffer.
RenderSystem::UnmapBuffer.

#### 10.42.4.5 std::uint32_t LLGL::FrameProfile::bufferReads

Counter for all buffer write operations outside of command encoding.

**Todo** Not available yet.

#### 10.42.4.6 std::uint32_t LLGL::FrameProfile::bufferUpdates

Counter for all buffer updates during command encoding.

**See also**

CommandBuffer::UpdateBuffer

#### 10.42.4.7 std::uint32_t LLGL::FrameProfile::bufferWrites

Counter for all buffer write operations outside of command encoding.

**See also**

RenderSystem::WriteBuffer

#### 10.42.4.8 std::uint32_t LLGL::FrameProfile::commandBufferEncodings

Counter for all command buffer encodings that are enclosed by a call to `Begin` and `End`.

**See also**

CommandBuffer::Begin
CommandBuffer::End

#### 10.42.4.9 std::uint32_t LLGL::FrameProfile::commandBufferSubmittions

Counter for all command buffers that were submitted to the queue.

**See also**

CommandQueue::Submit(CommandBuffer&)
CommandQueue::Submit(std::uint32_t, CommandBuffer∗ const ∗)

#### 10.42.4.10 std::uint32_t LLGL::FrameProfile::computePipelineBindings

Counter for all compute pipeline bindings.

**See also**

CommandBuffer::SetComputePipeline

#### 10.42.4.11 std::uint32_t LLGL::FrameProfile::computeResourceHeapBindings

Counter for all resource heap bindings on the compute pipeline.

**See also**

CommandBuffer::SetComputeResourceHeap

#### 10.42.4.12 std::uint32_t LLGL::FrameProfile::constantBufferBindings

Counter for all individual constant buffer bindings.

**See also**

CommandBuffer::SetResource

#### 10.42.4.13 std::uint32_t LLGL::FrameProfile::dispatchCommands

Counter for dispatch compute commands.

**See also**

CommandBuffer::Dispatch

#### 10.42.4.14 std::uint32_t LLGL::FrameProfile::drawCommands

Counter for all draw commands.

**See also**

CommandBuffer::Draw
CommandBuffer::DrawIndexed
CommandBuffer::DrawInstanced
CommandBuffer::DrawIndexedInstanced

#### 10.42.4.15 std::uint32_t LLGL::FrameProfile::fenceSubmissions

Counter for all fences that were submitted to the queue.

**See also**

CommandQueue::Submit(Fence&)

#### 10.42.4.16 std::uint32_t LLGL::FrameProfile::graphicsPipelineBindings

Counter for all graphics pipeline bindings.

**See also**

CommandBuffer::SetGraphicsPipeline

#### 10.42.4.17 std::uint32_t LLGL::FrameProfile::graphicsResourceHeapBindings

Counter for all resource heap bindings on the graphics pipeline.

**See also**

CommandBuffer::SetGraphicsResourceHeap

#### 10.42.4.18 std::uint32_t LLGL::FrameProfile::indexBufferBindings

Counter for all index buffer bindings.

**See also**

CommandBuffer::SetIndexBuffer

#### 10.42.4.19 std::uint32_t LLGL::FrameProfile::mipMapsGenerations

Counter for all MIP-map generations.

**See also**

CommandBuffer::GenerateMips

#### 10.42.4.20 std::uint32_t LLGL::FrameProfile::querySections

Counter for all command buffer sections that are enclosed by a call to `BeginQuery` and `EndQuery`.

**See also**

CommandBuffer::BeginQuery
CommandBuffer::EndQuery

#### 10.42.4.21 std::uint32_t LLGL::FrameProfile::renderConditionSections

Counter for all command buffer sections that are enclosed by a call to `BeginRenderCondition` and `EndRenderCondition`.

**See also**

CommandBuffer::BeginRenderCondition
CommandBuffer::EndRenderCondition

#### 10.42.4.22 std::uint32_t LLGL::FrameProfile::renderPassSections

Counter for all command buffer sections that are enclosed by a call to `BeginRenderPass` and `EndRenderPass`.

**See also**

CommandBuffer::BeginRenderPass
CommandBuffer::EndRenderPass

#### 10.42.4.23 std::uint32_t LLGL::FrameProfile::sampledBufferBindings

Counter for all sampled buffer bindings (i.e. with BindFlags::Sampled flag).

**See also**

CommandBuffer::SetResource

#### 10.42.4.24 std::uint32_t LLGL::FrameProfile::sampledTextureBindings

Counter for all sampled texture bindings (i.e. with BindFlags::Sampled flag).

**See also**

CommandBuffer::SetResource

#### 10.42.4.25 std::uint32_t LLGL::FrameProfile::samplerBindings

Counter for all sampler-state bindings.

**See also**

CommandBuffer::SetResource

#### 10.42.4.26 std::uint32_t LLGL::FrameProfile::storageBufferBindings

Counter for all storage buffer bindings (i.e. with BindFlags::Storage flag).

**See also**

CommandBuffer::SetResource

#### 10.42.4.27 std::uint32_t LLGL::FrameProfile::storageTextureBindings

Counter for all sampled texture bindings (i.e. with BindFlags::Storage flag).

**See also**

CommandBuffer::SetResource

#### 10.42.4.28 std::uint32_t LLGL::FrameProfile::streamOutputBufferBindings

Counter for all stream-output buffer and stream-output buffer array bindings.

**See also**

CommandBuffer::SetStreamOutputBuffer
CommandBuffer::SetStreamOutputBufferArray

#### 10.42.4.29 std::uint32_t LLGL::FrameProfile::streamOutputSections

Counter for all command buffer sections that are enclosed by a call to `BeginStreamOutput` and `EndStream↩Output`.

**See also**

CommandBuffer::BeginStreamOutput
CommandBuffer::EndStreamOutput

#### 10.42.4.30 std::uint32_t LLGL::FrameProfile::textureCopies

Counter for all texture copies during command encoding.

**See also**

CommandBuffer::CopyTexture.

#### 10.42.4.31 std::uint32_t LLGL::FrameProfile::textureMappings

Counter for all texture write operations outside of command encoding.

**Todo** Not available yet.

#### 10.42.4.32 std::uint32_t LLGL::FrameProfile::textureReads

Counter for all texture write operations outside of command encoding.

**See also**

RenderSystem::ReadTexture.

#### 10.42.4.33 std::uint32_t LLGL::FrameProfile::textureWrites

Counter for all texture write operations outside of command encoding.

**See also**

RenderSystem::WriteTexture.

#### 10.42.4.34 std::uint32_t LLGL::FrameProfile::values[33]

All proflile values as linear array.

#### 10.42.4.35 std::uint32_t LLGL::FrameProfile::vertexBufferBindings

Counter for all vertex buffer and vertex buffer array bindings.

**See also**

CommandBuffer::SetVertexBuffer
CommandBuffer::SetVertexBufferArray

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

- RenderingProfiler.h

## 10.43 LLGL::GraphicsPipeline Class Reference

Graphics pipeline interface.

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

Inheritance diagram for LLGL::GraphicsPipeline:

### Additional Inherited Members

### 10.43.1 Detailed Description

Graphics pipeline interface.

**See also**

RenderSystem::CreateGraphicsPipeline
CommandBuffer::SetGraphicsPipeline

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

- GraphicsPipeline.h

## 10.44 LLGL::GraphicsPipelineDescriptor Struct Reference

Graphics pipeline descriptor structure.

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

### Public Attributes

- const ShaderProgram * shaderProgram = nullptr

  *Pointer to the shader program for the graphics pipeline. By default null.*
- const RenderPass * renderPass = nullptr

  *Pointer to a RenderPass object. By default null.*
- const PipelineLayout * pipelineLayout = nullptr

  *Pointer to an optional pipeline layout for the graphics pipeline. By default null.*
- PrimitiveTopology primitiveTopology = PrimitiveTopology::TriangleList

  *Specifies the primitive topology and ordering of the primitive data. By default PrimitiveTopology::TriangleList.*
- std::vector< Viewport > viewports

  *Specifies an optional list of viewports. If empty, the viewports must be set dynamically with the command buffer.*
- std::vector< Scissor > scissors

  *Specifies an optional list of scissor rectangles. If empty, the scissors must be set dynamically with the command buffer.*
- DepthDescriptor depth

  *Specifies the depth state for the depth-stencil stage.*
- StencilDescriptor stencil

  *Specifies the stencil state for the depth-stencil stage.*
- RasterizerDescriptor rasterizer

  *Specifies the state for the rasterizer stage.*
- BlendDescriptor blend

  *Specifies the state descriptor for the blend stage.*

### 10.44.1 Detailed Description

Graphics pipeline descriptor structure.

#### Remarks

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

#### See also

RenderSystem::CreateGraphicsPipeline

### 10.44.2 Member Data Documentation

#### 10.44.2.1 BlendDescriptor LLGL::GraphicsPipelineDescriptor::blend

Specifies the state descriptor for the blend stage.

#### 10.44.2.2 DepthDescriptor LLGL::GraphicsPipelineDescriptor::depth

Specifies the depth state for the depth-stencil stage.

#### 10.44.2.3 const PipelineLayout* LLGL::GraphicsPipelineDescriptor::pipelineLayout = nullptr

Pointer to an optional pipeline layout for the graphics pipeline. By default null.

**Remarks**

This layout determines at which slots buffer resources can be bound. This is ignored by render systems which do not support pipeline layouts.

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

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

**See also**

PrimitiveTopology

#### 10.44.2.5 RasterizerDescriptor LLGL::GraphicsPipelineDescriptor::rasterizer

Specifies the state for the rasterizer stage.

#### 10.44.2.6 const RenderPass* LLGL::GraphicsPipelineDescriptor::renderPass = nullptr

Pointer to a RenderPass object. By default null.

**Remarks**

If this is null, the render pass of the RenderContext that was first created is used. This render pass must be compatible with the one passed to the CommandBuffer::BeginRenderPass function in which the graphics pipeline will be used.

**See also**

CommandBuffer::BeginRenderPass

#### 10.44.2.7 std::vector<Scissor> LLGL::GraphicsPipelineDescriptor::scissors

Specifies an optional list of scissor rectangles. If empty, the scissors must be set dynamically with the command buffer.

**Remarks**

This list must have the same number of entries as `viewports`, unless one of the lists is empty.

**See also**

CommandBuffer::SetScissor
CommandBuffer::SetScissors

#### 10.44.2.8 const ShaderProgram* LLGL::GraphicsPipelineDescriptor::shaderProgram = nullptr

Pointer to the shader program for the graphics pipeline. By default null.

**Remarks**

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

**See also**

RenderSystem::CreateShaderProgram

#### 10.44.2.9 StencilDescriptor LLGL::GraphicsPipelineDescriptor::stencil

Specifies the stencil state for the depth-stencil stage.

#### 10.44.2.10 std::vector<Viewport> LLGL::GraphicsPipelineDescriptor::viewports

Specifies an optional list of viewports. If empty, the viewports must be set dynamically with the command buffer.

**Remarks**

This list must have the same number of entries as `scissors`, unless one of the lists is empty.

**See also**

CommandBuffer::SetViewport
CommandBuffer::SetViewports

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

- GraphicsPipelineFlags.h

## 10.45 LLGL::Image Class Reference

Utility class to manage the storage and attributes of an image.

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

## Public Member Functions

- Image ()=default
- Image (const Extent3D &extent, const ImageFormat format, const DataType dataType)

  *Constructor to initialize the image with a format, data type, and extent.*
- Image (const Extent3D &extent, const ImageFormat format, const DataType dataType, const ColorRGBAd &fillColor)

  *Constructor to initialize the image with a format, data type, and extent. The image buffer will be filled with the specified color.*
- Image (const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data)

  *Constructor to initialize the image with all atributes, including the image buffer specified by the 'data' parameter.*
- Image (const Image &rhs)

  *Copy constructor which copies the entire image buffer from the specified source image.*
- Image (Image &&rhs)

  *Move constructor which takes the ownership of the specified source image.*
- Image & operator= (const Image &rhs)

  *Copy operator which copies the entire image buffer and attributes.*
- Image & operator= (Image &&rhs)

  *Move operator which takes the ownership of the image buffer.*
- void Convert (const ImageFormat format, const DataType dataType, std::size_t threadCount=0)

  *Converts the image format and data type.*
- void Resize (const Extent3D &extent)

  *Resizes the image and resets the image buffer.*
- void Resize (const Extent3D &extent, const ColorRGBAd &fillColor)

  *Resizes the image and initializes the new pixels with the specified color.*
- void Resize (const Extent3D &extent, const ColorRGBAd &fillColor, const Offset3D &offset)

  *Resizes the image, moves the previous pixels by an offset, and initializes the new pixels outside the previous extent with the specified color.*
- void Resize (const Extent3D &extent, const SamplerFilter filter)

  *Resizes the image and resamples the pixels from the previous image buffer.*
- void Swap (Image &rhs)

  *Swaps all attributes with the specified image.*
- void Reset ()

  *Resets all image attributes to its default values.*
- void Reset (const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data)

  *Resets all image attributes to the specified values.*
- ByteBuffer Release ()

  *Releases the ownership of the image buffer and resets all attributes.*
- void Blit (Offset3D dstRegionOffset, const Image &srcImage, Offset3D srcRegionOffset, Extent3D srcRegionExtent)

  *Copies a region of the specified source image into this image.*
- void Fill (Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor)

  *Fills a region of this image by the specified color.*
- void ReadPixels (const Offset3D &offset, const Extent3D &extent, const DstImageDescriptor &imageDesc, std::size_t threadCount=0) const

  *Reads a region of pixels from this image into the destination image buffer specified by 'imageDesc'.*
- void WritePixels (const Offset3D &offset, const Extent3D &extent, const SrcImageDescriptor &imageDesc, std::size_t threadCount=0)

  *Writes a region of pixels to this image from the source image buffer specified by 'imageDesc'.*
- void MirrorYZPlane ()

  *Mirrors the image at the YZ plane.*

- void MirrorXZPlane ()

  *Mirrors the image at the XZ plane.*
- void MirrorXYPlane ()

  *Mirrors the image at the XY plane.*
- SrcImageDescriptor GetSrcDesc () const

  *Returns a source image descriptor for this image with read-only access to the image data.*
- DstImageDescriptor GetDstDesc ()

  *Returns a destination image descriptor for this image with read/write access to the image data.*
- const Extent3D & GetExtent () const

  *Returns the extent of the image as 3D vector.*
- ImageFormat GetFormat () const

  *Returns the format for each pixel. By default ImageFormat::RGBA.*
- DataType GetDataType () const

  *Returns the data type for each pixel component. By default DataType::UInt8.*
- const void * GetData () const

  *Returns the image data buffer as constant raw pointer.*
- void * GetData ()

  *Returns the image data buffer as raw pointer.*
- std::uint32_t GetBytesPerPixel () const

  *Returns the size (in bytes) for each pixel.*
- std::uint32_t GetRowStride () const

  *Returns the stride (in bytes) for each row.*
- std::uint32_t GetDepthStride () const

  *Returns the stride (in bytes) for each depth slice.*
- std::uint32_t GetNumPixels () const

  *Returns the number of pixels this image has.*
- std::uint32_t GetDataSize () const

  *Returns the size (in bytes) of the image buffer.*
- bool IsRegionInside (const Offset3D &offset, const Extent3D &extent) const

  *Returns true if the specified sub-image region is inside the image.*

### 10.45.1 Detailed Description

Utility class to manage the storage and attributes of an image.

This class is not required for any interaction with the render system. It can be used as utility to handle 2D and 3D image data before passing it to a hardware texture.

**Remarks**

This class holds the ownership of an image buffer and its attributes. The primary functions are implemented as global functions like `GenerateImageBuffer` for instance.

**Note**

All image operations of this class do NOT make use of hardware acceleration.

**See also**

GenerateImageBuffer
ConvertImageBuffer

## 10.45.2 Constructor & Destructor Documentation

### 10.45.2.1 LLGL::Image::Image ( ) [default]

### 10.45.2.2 LLGL::Image::Image ( const Extent3D & *extent,* const ImageFormat *format,* const DataType *dataType* )

Constructor to initialize the image with a format, data type, and extent.

**Note**

The image buffer will be uninitialized!

**See also**

Fill

### 10.45.2.3 LLGL::Image::Image ( const Extent3D & *extent,* const ImageFormat *format,* const DataType *dataType,* const ColorRGBAd & *fillColor* )

Constructor to initialize the image with a format, data type, and extent. The image buffer will be filled with the specified color.

**See also**

GenerateImageBuffer

### 10.45.2.4 LLGL::Image::Image ( const Extent3D & *extent,* const ImageFormat *format,* const DataType *dataType,* ByteBuffer && *data* )

Constructor to initialize the image with all atributes, including the image buffer specified by the 'data' parameter.

**Note**

If the specified data does not manage an image buffer of the specified extent and format, the behavior is undefined.

**See also**

Reset(const Extent3D&, const ImageFormat, const DataType, ByteBuffer&&)

### 10.45.2.5 LLGL::Image::Image ( const Image & *rhs* )

Copy constructor which copies the entire image buffer from the specified source image.

### 10.45.2.6 LLGL::Image::Image ( Image && *rhs* )

Move constructor which takes the ownership of the specified source image.

## 10.45.3 Member Function Documentation

### 10.45.3.1 void LLGL::Image::Blit ( Offset3D *dstRegionOffset,* const Image & *srcImage,* Offset3D *srcRegionOffset,* Extent3D *srcRegionExtent* )

Copies a region of the specified source image into this image.

**Parameters**

| | | |
|---|---|---|
| in | *dstRegionOffset* | Specifies the offset within the destination image (i.e. this Image instance). This can also be outside of the image area. |
| in | *srcImage* | Specifies the source image whose region is to be copied. This must have the same format and data type as this image. If the source image is the same object as this image and the destination and source regions overlap, an internal temporary copy is allocated for reading the data. |
| in | *srcRegionOffset* | Specifies the offset within the source image. This will be clamped if it exceeds the source image area. |
| in | *srcRegionExtent* | Specifies the extent of the region to copy. This will be clamped if it exceeds the source or destination image area. |

**Remarks**

If one of the region offsets is clamped, the region extent will be adjusted respectively. If the source image has a different format or data type compared to this image, the function has no effect.

**See also**

ConvertImageBuffer

#### 10.45.3.2 void LLGL::Image::Convert ( const ImageFormat *format*, const DataType *dataType*, std::size_t *threadCount* = 0 )

Converts the image format and data type.

**See also**

ConvertImageBuffer

#### 10.45.3.3 void LLGL::Image::Fill ( Offset3D *offset*, Extent3D *extent*, const ColorRGBAd & *fillColor* )

Fills a region of this image by the specified color.

**Parameters**

| | | |
|---|---|---|
| in | *offset* | Specifies the offset where the region begins. |
| in | *extent* | Specifies the extent of the region. |
| in | *fillColor* | Specifies the color to fill the region with. |

**Todo** Not implemented yet.

#### 10.45.3.4 std::uint32_t LLGL::Image::GetBytesPerPixel ( ) const

Returns the size (in bytes) for each pixel.

**See also**

GetFormat
ImageFormatSize
GetDataType
DataTypeSize

#### 10.45.3.5 const void* LLGL::Image::GetData ( ) const [inline]

Returns the image data buffer as constant raw pointer.

#### 10.45.3.6 void* LLGL::Image::GetData ( ) [inline]

Returns the image data buffer as raw pointer.

#### 10.45.3.7 std::uint32_t LLGL::Image::GetDataSize ( ) const

Returns the size (in bytes) of the image buffer.

**See also**

GetBytesPerPixel
GetNumPixels

#### 10.45.3.8 DataType LLGL::Image::GetDataType ( ) const [inline]

Returns the data type for each pixel component. By default DataType::UInt8.

#### 10.45.3.9 std::uint32_t LLGL::Image::GetDepthStride ( ) const

Returns the stride (in bytes) for each depth slice.

#### 10.45.3.10 DstImageDescriptor LLGL::Image::GetDstDesc ( )

Returns a destination image descriptor for this image with read/write access to the image data.

#### 10.45.3.11 const Extent3D& LLGL::Image::GetExtent ( ) const [inline]

Returns the extent of the image as 3D vector.

#### 10.45.3.12 ImageFormat LLGL::Image::GetFormat ( ) const [inline]

Returns the format for each pixel. By default ImageFormat::RGBA.

#### 10.45.3.13 std::uint32_t LLGL::Image::GetNumPixels ( ) const

Returns the number of pixels this image has.

**Remarks**

This is equivalent to the following code example:

```
const auto& extent = myImage.GetExtent();
return extent.width * extent.height * extent.depth;
```

**See also**

GetExtent

#### 10.45.3.14 std::uint32_t LLGL::Image::GetRowStride ( ) const

Returns the stride (in bytes) for each row.

#### 10.45.3.15 SrcImageDescriptor LLGL::Image::GetSrcDesc ( ) const

Returns a source image descriptor for this image with read-only access to the image data.

#### 10.45.3.16 bool LLGL::Image::IsRegionInside ( const Offset3D & *offset,* const Extent3D & *extent* ) const

Returns true if the specified sub-image region is inside the image.

#### 10.45.3.17 void LLGL::Image::MirrorXYPlane ( )

Mirrors the image at the XY plane.

**Todo** Not implemented yet

#### 10.45.3.18 void LLGL::Image::MirrorXZPlane ( )

Mirrors the image at the XZ plane.

**Todo** Not implemented yet

#### 10.45.3.19 void LLGL::Image::MirrorYZPlane ( )

Mirrors the image at the YZ plane.

**Todo** Not implemented yet

#### 10.45.3.20 Image& LLGL::Image::operator= ( const Image & *rhs* )

Copy operator which copies the entire image buffer and attributes.

#### 10.45.3.21 Image& LLGL::Image::operator= ( Image && *rhs* )

Move operator which takes the ownership of the image buffer.

#### 10.45.3.22 void LLGL::Image::ReadPixels ( const Offset3D & *offset*, const Extent3D & *extent*, const DstImageDescriptor & *imageDesc*, std::size_t *threadCount* = 0 ) const

Reads a region of pixels from this image into the destination image buffer specified by 'imageDesc'.

**Parameters**

| | | |
|---|---|---|
| in | *offset* | Specifies the region offset within this image to read from. |
| in | *extent* | Specifies the region extent within this image to read from. |
| in | *imageDesc* | Specifies the destination image descriptor to write the region to. If the 'data' member of this descriptor is null or if the sub-image region is not inside the image, this function has no effect. |
| in | *threadCount* | Specifies the number of threads to use if the data needs to be converted (see ConvertImageBuffer for more details). By default 0. |

**Remarks**

To read a single pixel, use the following code example:

```
LLGL::ColorRGBAub ReadSinglePixelRGBAub(const LLGL::Image& image, const
      LLGL::Offset3D& position) {
    LLGL::ColorRGBAub pixelColor;
    const DstImageDescriptor imageDesc { LLGL::ImageFormat::RGBA,
      LLGL::DataType::UInt8, &pixelColor, sizeof(pixelColor) };
    image.ReadPixels(position, { 1, 1, 1 }, imageDesc);
    return pixelColor;
}
```

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the 'data' member of the image descriptor is non-null, the sub-image region is inside the image, but the 'dataSize' member of the image descriptor is too small. |

**See also**

IsRegionInside
ConvertImageBuffer

#### 10.45.3.23 ByteBuffer LLGL::Image::Release ( )

Releases the ownership of the image buffer and resets all attributes.

#### 10.45.3.24 void LLGL::Image::Reset ( )

Resets all image attributes to its default values.

#### 10.45.3.25 void LLGL::Image::Reset ( const Extent3D & *extent*, const ImageFormat *format*, const DataType *dataType*, ByteBuffer && *data* )

Resets all image attributes to the specified values.

**Note**

If the specified data does not manage an image buffer of the specified extent and format, the behavior is undefined.

**See also**

GenerateImageBuffer
GenerateEmptyByteBuffer

#### 10.45.3.26 void LLGL::Image::Resize ( const Extent3D & *extent* )

Resizes the image and resets the image buffer.

**Parameters**

| | | |
|---|---|---|
| in | *extent* | Specifies the new image size. |

**Note**

The new image buffer will be uninitialized!

#### 10.45.3.27 void LLGL::Image::Resize ( const Extent3D & *extent*, const ColorRGBAd & *fillColor* )

Resizes the image and initializes the new pixels with the specified color.

**Parameters**

| | | |
|---|---|---|
| in | *extent* | Specifies the new image size. |
| in | *fillColor* | Specifies the color to fill the pixels with. GenerateImageBuffer |

#### 10.45.3.28 void LLGL::Image::Resize ( const Extent3D & *extent*, const ColorRGBAd & *fillColor*, const Offset3D & *offset* )

Resizes the image, moves the previous pixels by an offset, and initializes the new pixels outside the previous extent with the specified color.

**Parameters**

| | | |
|---|---|---|
| in | *extent* | Specifies the new image size. |
| in | *fillColor* | Specifies the color to fill the pixels with that are outside the previous extent. |
| in | *offset* | Specifies the offset to move the previous pixels to. This will be clamped if it exceeds the image area. GenerateImageBuffer |

#### 10.45.3.29 void LLGL::Image::Resize ( const Extent3D & *extent*, const SamplerFilter *filter* )

Resizes the image and resamples the pixels from the previous image buffer.

**Parameters**

| | | |
|---|---|---|
| in | *extent* | Specifies the new image size. |
| in | *filter* | Specifies the sampling filter. GenerateImageBuffer |

**Todo** Not implemented yet.

#### 10.45.3.30 void LLGL::Image::Swap ( Image & *rhs* )

Swaps all attributes with the specified image.

#### 10.45.3.31 void LLGL::Image::WritePixels ( const Offset3D & *offset*, const Extent3D & *extent*, const SrcImageDescriptor & *imageDesc*, std::size_t *threadCount* = 0 )

Writes a region of pixels to this image from the source image buffer specified by 'imageDesc'.

**Parameters**

| | | |
|---|---|---|
| in | *offset* | Specifies the region offset within this image to write to. |
| in | *extent* | Specifies the region extent within this image to write to. |
| in | *imageDesc* | Specifies the source image descriptor to read the region from. If the 'data' member of this descriptor is null or if the sub-image region is not inside the image, this function has no effect. |
| in | *threadCount* | Specifies the number of threads to use if the data needs to be converted (see ConvertImageBuffer for more details). By default 0. |

**See also**

IsRegionInside
ConvertImageBuffer

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

- Image.h

## 10.46 LLGL::BufferDescriptor::IndexBuffer Struct Reference

Index buffer specific descriptor structure.

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

### Public Attributes

- Format format = Format::R32UInt

  *Specifies the format of each index in the buffer. This must be either Format::R16UInt or Format::R32UInt. By default Format::R32UInt.*

### 10.46.1 Detailed Description

Index buffer specific descriptor structure.

### 10.46.2 Member Data Documentation

#### 10.46.2.1 Format LLGL::BufferDescriptor::IndexBuffer::format = Format::R32UInt

Specifies the format of each index in the buffer. This must be either Format::R16UInt or Format::R32UInt. By default Format::R32UInt.

**See also**

Format

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

- BufferFlags.h

## 10.47 LLGL::Input Class Reference

Default window event listener to receive user input.

```
#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 KeyDownRepeated (Key keyCode) const

  *Returns true if the specified key was pressed down in the previous event processing (this event will be repeated, depending on the paltform settings).*
- 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 Offset2D & GetMousePosition () const

  *Returns the local mouse position.*
- const Offset2D & GetMouseMotion () const

  *Returns the global mouse motion.*
- int GetWheelMotion () const

  *Returns the mouse wheel motion.*
- const std::wstring & GetEnteredChars () const

  *Returns the entered characters.*
- std::size_t GetAnyKeyCount () const

  *Returns the number of any keys being pressed.*

## Protected Member Functions

- void OnProcessEvents (Window &sender) override

  *Send when the window events are about to be polled. The event listeners receive this event before the window itself.*
- void OnKeyDown (Window &sender, Key keyCode) override

  *Send when a key (from keyboard or mouse) has been pushed.*
- void OnKeyUp (Window &sender, Key keyCode) override

  *Send when a key (from keyboard or mouse) has been released.*
- void OnDoubleClick (Window &sender, Key keyCode) override

  *Send when a mouse button has been double clicked.*
- void OnChar (Window &sender, wchar_t chr) override

  *Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.*
- void OnWheelMotion (Window &sender, int motion) override

  *Send when the mouse wheel has been moved on the sender window.*
- void OnLocalMotion (Window &sender, const Offset2D &position) override

  *Send when the mouse has been moved on the sender window.*
- void OnGlobalMotion (Window &sender, const Offset2D &motion) override

  *Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.*
- void OnLostFocus (Window &sender) override

  *Send when the window lost the keyboard focus.*

### 10.47.1 Detailed Description

Default window event listener to receive user input.

#### Remarks

This class stores all received user input for a simple evaluation. However, for efficient evaluation, write your own sub class and only respond to user input when the appropriate callback is invoked. Here is an example usage:

```
auto myInput = std::make_shared<LLGL::Input>();
myWindow->AddEventListener(myInput);
while (myWindow->ProcessEvents()) {
    // Quit main loop when user hit the escape key.
    if (myInput->KeyDown(LLGL::Key::Escape))
        break;

    // Rendering goes here ...
}
```

### 10.47.2 Constructor & Destructor Documentation

#### 10.47.2.1 LLGL::Input::Input ( )

### 10.47.3 Member Function Documentation

#### 10.47.3.1 std::size_t LLGL::Input::GetAnyKeyCount ( ) const [inline]

Returns the number of any keys being pressed.

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

Returns the entered characters.

#### 10.47.3.3 const Offset2D& LLGL::Input::GetMouseMotion ( ) const [inline]

Returns the global mouse motion.

#### 10.47.3.4 const Offset2D& LLGL::Input::GetMousePosition ( ) const [inline]

Returns the local mouse position.

#### 10.47.3.5 int LLGL::Input::GetWheelMotion ( ) const [inline]

Returns the mouse wheel motion.

#### 10.47.3.6 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.

#### 10.47.3.7 bool LLGL::Input::KeyDown ( Key *keyCode* ) const

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

#### 10.47.3.8 bool LLGL::Input::KeyDownRepeated ( Key *keyCode* ) const

Returns true if the specified key was pressed down in the previous event processing (this event will be repeated, depending on the paltform settings).

#### 10.47.3.9 bool LLGL::Input::KeyPressed ( Key *keyCode* ) const

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

#### 10.47.3.10 bool LLGL::Input::KeyUp ( Key *keyCode* ) const

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

#### 10.47.3.11 void LLGL::Input::OnChar ( Window & *sender,* wchar_t *chr* ) [override], [protected], [virtual]

Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.12 void LLGL::Input::OnDoubleClick ( Window & *sender,* Key *keyCode* ) [override], [protected], [virtual]

Send when a mouse button has been double clicked.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.13 void LLGL::Input::OnGlobalMotion ( Window & *sender,* const Offset2D & *motion* ) [override], [protected], [virtual]

Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.14 void LLGL::Input::OnKeyDown ( Window & *sender,* Key *keyCode* ) [override], [protected], [virtual]

Send when a key (from keyboard or mouse) has been pushed.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.15 void LLGL::Input::OnKeyUp ( Window & *sender,* Key *keyCode* ) [override], [protected], [virtual]

Send when a key (from keyboard or mouse) has been released.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.16 void LLGL::Input::OnLocalMotion ( Window & *sender,* const Offset2D & *position* ) [override], [protected], [virtual]

Send when the mouse has been moved on the sender window.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.17 void LLGL::Input::OnLostFocus ( Window & *sender* ) [override], [protected], [virtual]

Send when the window lost the keyboard focus.

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.18 void LLGL::Input::OnProcessEvents ( Window & *sender* ) [override], [protected], [virtual]

Send when the window events are about to be polled. The event listeners receive this event before the window itself.

**See also**

Window::OnProcessEvents

Reimplemented from LLGL::Window::EventListener.

#### 10.47.3.19 void LLGL::Input::OnWheelMotion ( Window & *sender,* int *motion* ) [override], [protected], [virtual]

Send when the mouse wheel has been moved on the sender window.

Reimplemented from LLGL::Window::EventListener.

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

- Input.h

## 10.48 LLGL::Interface Class Reference

Base class for all interfaces in LLGL.

```
#include <Interface.h>
```

Inheritance diagram for LLGL::Interface:

### Public Member Functions

- virtual bool IsInstanceOf (int id) const

  *Returns true if this object is an instance of the specified interface.*

### Additional Inherited Members

### 10.48.1 Detailed Description

Base class for all interfaces in LLGL.

#### See also

Display
RenderSystem
RenderSystemChild
Surface
Timer

### 10.48.2 Member Function Documentation

#### 10.48.2.1 virtual bool LLGL::Interface::IsInstanceOf ( int *id* ) const [virtual]

Returns true if this object is an instance of the specified interface.

**Remarks**

Can be used for a more efficient run-time type information (RTTI) than the default C++ mechanism. This can be used like in the following example:

```
LLGL::Surface* mySurface = ...
if (mySurface->IsInstanceOf(LLGL::InterfaceID::Window))
{
    LLGL::Window* myWindow = static_cast<LLGL::Window*>(mySurface);
    ...
}
```

**See also**

LLGL::CastTo

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

- Interface.h

## 10.49 LLGL::InterfaceID Struct Reference

Enumeration of all interfaces in LLGL.

```
#include <InterfaceID.h>
```

### Public Types

- enum {
  Interface, Display, RenderSystem, RenderSystemChild,
  Buffer, BufferArray, CommandBuffer, CommandQueue,
  ComputePipeline, Fence, GraphicsPipeline, PipelineLayout,
  QueryHeap, RenderPass, RenderTarget, RenderContext,
  Resource, ResourceHeap, Sampler, Shader,
  ShaderProgram, Texture, Surface, Canvas,
  Canvas_EventListener, Window, Window_EventListener, Input,
  Timer, Reserved }

### 10.49.1 Detailed Description

Enumeration of all interfaces in LLGL.

**See also**

Interface::IsInstanceOf
Interface::AsInstanceOf

## 10.49.2 Member Enumeration Documentation

### 10.49.2.1 anonymous enum

Enumerator See also

***Interface*** Interface
See also

***Display*** Display
See also

***RenderSystem*** RenderSystem
See also

***RenderSystemChild*** RenderSystemChild
See also

***Buffer*** Buffer
See also

***BufferArray*** BufferArray
See also

***CommandBuffer*** CommandBuffer
See also

***CommandQueue*** CommandQueue
See also

***ComputePipeline*** ComputePipeline
See also

***Fence*** Fence
See also

***GraphicsPipeline*** GraphicsPipeline
See also

***PipelineLayout*** PipelineLayout
See also

***QueryHeap*** QueryHeap
See also

***RenderPass*** RenderPass
See also

***RenderTarget*** RenderTarget
See also

***RenderContext*** RenderContext
See also

***Resource*** Resource
See also

***ResourceHeap*** ResourceHeap
See also

***Sampler*** Sampler
See also

***Shader*** Shader
See also

***ShaderProgram*** ShaderProgram

See also

***Texture*** Texture

See also

***Surface*** Surface

See also

***Canvas*** Canvas

See also

***Canvas_EventListener*** Canvas::EventListener

See also

***Window*** Window

See also

***Window_EventListener*** Window::EventListener

See also

***Input*** Input

See also

***Timer*** Timer

***Reserved*** Maximum reserved ID for interfaces.

Remarks

Use this value to write custom sub classes of an LLGL interface:

```
class MyWindow : public LLGL::Window {
    LLGL_DECLARE_INTERFACE(LLGL::InterfaceID::Reserved + 1
      );
    ...
};
LLGL_IMPLEMENT_INTERFACE(MyWindow, LLGL::Window);
```

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

- InterfaceID.h

## 10.50 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, const std::string &groupName)
- 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.*
- std::string ToReportString (const std::string &reportTypeName) const

  *Returns a report string for this message.*
- const std::string & GetText () const

*Returns the message text.*

- const std::string & GetSource () const

  *Returns the source function where this message occurred.*
- const std::string & GetGroupName () const

  *Returns the debug group name where this message occured.*
- 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

## 10.50.1 Detailed Description

Rendering debugger message class.

**Todo** Rename to "Report"

## 10.50.2 Constructor & Destructor Documentation

#### 10.50.2.1 LLGL::RenderingDebugger::Message::Message ( ) [default]

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

#### 10.50.2.3 LLGL::RenderingDebugger::Message::Message ( const std::string & *text*, const std::string & *source*, const std::string & *groupName* )

## 10.50.3 Member Function Documentation

#### 10.50.3.1 void LLGL::RenderingDebugger::Message::Block ( )

Blocks further occurrences of this message.

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

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

#### 10.50.3.3 const std::string& LLGL::RenderingDebugger::Message::GetGroupName ( ) const [inline]

Returns the debug group name where this message occured.

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

Returns the number of occurrences of this message.

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

Returns the source function where this message occurred.

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

Returns the message text.

#### 10.50.3.7 void LLGL::RenderingDebugger::Message::IncOccurrence ( ) [protected]

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

Returns true if this message has already been blocked.

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

#### 10.50.3.10 std::string LLGL::RenderingDebugger::Message::ToReportString ( const std::string & *reportTypeName* ) const

Returns a report string for this message.

**Parameters**

| | | |
|---|---|---|
| in | *reportTypeName* | Specifies the name of the report type (e.g. "ERROR"). |

**Returns**

Constructed report string containing all information of this message.

### 10.50.4 Friends And Related Function Documentation

#### 10.50.4.1 friend class RenderingDebugger [friend]

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

- RenderingDebugger.h

## 10.51 LLGL::MiscFlags Struct Reference

Miscellaneous resource flags.

`#include <ResourceFlags.h>`

### Public Types

- enum { DynamicUsage = (1 << 0), FixedSamples = (1 << 1), GenerateMips = (1 << 2), NoInitialData = (1 << 3) }

### 10.51.1 Detailed Description

Miscellaneous resource flags.

**See also**

BufferDescriptor::miscFlags
TextureDescriptor::miscFlags

### 10.51.2 Member Enumeration Documentation

#### 10.51.2.1 anonymous enum

**Enumerator**

***DynamicUsage*** Hint to the renderer that the resource will be frequently updated from the CPU.

**Remarks**

This is useful for a constant buffer for instance, that is updated by the host program every frame.

**See also**

RenderSystem::WriteBuffer
RenderSystem::WriteTexture

**Todo** Restriction required to support deferred context in D3D11. This must no longer be just a "hint", it must be a strictly defined attribute for a buffer.

***FixedSamples*** Multi-sampled Texture resource has fixed sample locations.

**Remarks**

This can only be used with multi-sampled Texture resources (i.e. TextureType::Texture2DMS, TextureType::Texture2DMSArray).

***GenerateMips*** Generates MIP-maps at texture creation time with the initial image data (if specified).

**Remarks**

To generate MIP-maps, the texture must be created with the BindFlags::ColorAttachment flag. This can be used to generate all MIP-maps when a new texture is created without explicitly encoding the CommandBuffer::GenerateMips function. The number of MIP-maps being generated depends on the `mipLevels` attribute in TextureDescriptor.

**See also**

TextureDescriptor::mipLevels
CommandBuffer::GenerateMips

***NoInitialData*** Specifies to ignore resource data initialization.

**Remarks**

If this is specified, a texture or buffer resource will stay uninitialized during creation and the content is undefined.

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

- ResourceFlags.h

## 10.52 LLGL::MultiSamplingDescriptor Struct Reference

Multi-sampling descriptor structure.

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

### Public Member Functions

- MultiSamplingDescriptor ()=default
- MultiSamplingDescriptor (std::uint32_t samples, std::uint32_t sampleMask=~0)

  *Constructor to initialize the sample.*
- std::uint32_t 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.*
- std::uint32_t samples = 1

  *Number of samples used for multi-sampling. By default 1.*
- std::uint32_t sampleMask = ~0u

  *Specifies the bitmask for sample coverage. By default `0xFFFFFFFF`.*

### 10.52.1 Detailed Description

Multi-sampling descriptor structure.

**See also**

RasterizerDescriptor::multiSampling

### 10.52.2 Constructor & Destructor Documentation

#### 10.52.2.1 LLGL::MultiSamplingDescriptor::MultiSamplingDescriptor ( ) [default]

#### 10.52.2.2 LLGL::MultiSamplingDescriptor::MultiSamplingDescriptor ( std::uint32_t *samples*, std::uint32_t *sampleMask* = ~0 ) [inline]

Constructor to initialize the sample.

**Parameters**

| | | |
|---|---|---|
| in | *samples* | Specifies the number of samples used for multi-sampling. If this is greater than 1, multi-sampling is enabled. |
| in | *sampleMask* | Specifies the bitmask for sample coverage. |

### 10.52.3 Member Function Documentation

#### 10.52.3.1 std::uint32_t 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.

### 10.52.4 Member Data Documentation

#### 10.52.4.1 bool LLGL::MultiSamplingDescriptor::enabled = false

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

#### 10.52.4.2 std::uint32_t LLGL::MultiSamplingDescriptor::sampleMask = ~0u

Specifies the bitmask for sample coverage. By default `0xFFFFFFFF`.

#### 10.52.4.3 std::uint32_t LLGL::MultiSamplingDescriptor::samples = 1

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

**Remarks**

The equivalent member for multi-sampled textures is TextureDescriptor::samples.

**See also**

TextureDescriptor::samples

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

- GraphicsPipelineFlags.h

## 10.53 LLGL::NativeContextHandle Struct Reference

iOS native context handle structure.

```
#include <IOSNativeHandle.h>
```

### Public Attributes

- UIView * parentView
- ::Display * display
- ::Window parentWindow
- ::XVisualInfo * visual
- ::Colormap colorMap
- int screen
- NSWindow * parentWindow
- HWND parentWindow

## 10.53.1 Detailed Description

iOS native context handle structure.

Win32 native context handle structure.

MacOS native context handle structure.

Linux native context handle structure.

**Remarks**

This must be a POD (Plain-Old-Data) structure, so no default initialization is provided!

## 10.53.2 Member Data Documentation

### 10.53.2.1 ::Colormap LLGL::NativeContextHandle::colorMap

### 10.53.2.2 ::Display* LLGL::NativeContextHandle::display

### 10.53.2.3 UIView* LLGL::NativeContextHandle::parentView

### 10.53.2.4 NSWindow* LLGL::NativeContextHandle::parentWindow

### 10.53.2.5 ::Window LLGL::NativeContextHandle::parentWindow

### 10.53.2.6 HWND LLGL::NativeContextHandle::parentWindow

### 10.53.2.7 int LLGL::NativeContextHandle::screen

### 10.53.2.8 ::XVisualInfo* LLGL::NativeContextHandle::visual

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

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

## 10.54 LLGL::NativeHandle Struct Reference

iOS native handle structure.

```
#include <IOSNativeHandle.h>
```

### Public Attributes

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

### 10.54.1 Detailed Description

iOS native handle structure.

Win32 native handle structure.

MacOS native handle structure.

Linux native handle structure.

**Remarks**

This must be a POD (Plain-Old-Data) structure, so no default initialization is provided!

### 10.54.2 Member Data Documentation

#### 10.54.2.1 ::Display* LLGL::NativeHandle::display

#### 10.54.2.2 UIView* LLGL::NativeHandle::view

#### 10.54.2.3 ::XVisualInfo* LLGL::NativeHandle::visual

#### 10.54.2.4 NSWindow* LLGL::NativeHandle::window

#### 10.54.2.5 ::Window LLGL::NativeHandle::window

#### 10.54.2.6 HWND LLGL::NativeHandle::window

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

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

## 10.55 LLGL::NonCopyable Class Reference

Base class for all classes with virtual function tables in LLGL.

```
#include <NonCopyable.h>
```

Inheritance diagram for LLGL::NonCopyable:

### Public Member Functions

- NonCopyable (const NonCopyable &)=delete
- NonCopyable & operator= (const NonCopyable &)=delete
- virtual ~NonCopyable ()=default

### Protected Member Functions

- NonCopyable ()=default

### 10.55.1 Detailed Description

Base class for all classes with virtual function tables in LLGL.

#### Remarks

Sub classes of this interface cannot be copied on its own, since its copy constructor and copy operator are deleted functions.

#### See also

Interface

### 10.55.2 Constructor & Destructor Documentation

#### 10.55.2.1 LLGL::NonCopyable::NonCopyable ( const NonCopyable & ) [delete]

#### 10.55.2.2 virtual LLGL::NonCopyable::~NonCopyable ( ) [virtual],[default]

#### 10.55.2.3 LLGL::NonCopyable::NonCopyable ( ) [protected],[default]

### 10.55.3 Member Function Documentation

#### 10.55.3.1 NonCopyable& LLGL::NonCopyable::operator= ( const NonCopyable & ) [delete]

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

- NonCopyable.h

## 10.56 LLGL::Offset2D Struct Reference

2-Dimensional offset structure.

```
#include <Types.h>
```

### Public Member Functions

- Offset2D ()=default
- Offset2D (const Offset2D &)=default
- Offset2D (std::int32_t x, std::int32_t y)

### Public Attributes

- std::int32_t x = 0

  *Offset X axis.*
- std::int32_t y = 0

  *Offset Y axis.*

### 10.56.1 Detailed Description

2-Dimensional offset structure.

**Remarks**

Used for signed integral 2D offsets (for coordinates in window-space, screen-space, and texture-space).

### 10.56.2 Constructor & Destructor Documentation

#### 10.56.2.1 LLGL::Offset2D::Offset2D ( ) [default]

#### 10.56.2.2 LLGL::Offset2D::Offset2D ( const Offset2D & ) [default]

#### 10.56.2.3 LLGL::Offset2D::Offset2D ( std::int32_t *x*, std::int32_t *y* ) [inline]

### 10.56.3 Member Data Documentation

#### 10.56.3.1 std::int32_t LLGL::Offset2D::x = 0

Offset X axis.

#### 10.56.3.2 std::int32_t LLGL::Offset2D::y = 0

Offset Y axis.

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

- Types.h

## 10.57 LLGL::Offset3D Struct Reference

3-Dimensional offset structure.

```
#include <Types.h>
```

### Public Member Functions

- Offset3D ()=default
- Offset3D (const Offset3D &)=default
- Offset3D (std::int32_t x, std::int32_t y, std::int32_t z)

### Public Attributes

- std::int32_t x = 0

  *Offset X axis.*
- std::int32_t y = 0

  *Offset Y axis.*
- std::int32_t z = 0

  *Offset Z axis.*

### 10.57.1 Detailed Description

3-Dimensional offset structure.

**Remarks**

Used for signed integral 3D offsets (for coordinates in texture-space).

### 10.57.2 Constructor & Destructor Documentation

#### 10.57.2.1 LLGL::Offset3D::Offset3D ( ) [default]

#### 10.57.2.2 LLGL::Offset3D::Offset3D ( const Offset3D & ) [default]

#### 10.57.2.3 LLGL::Offset3D::Offset3D ( std::int32_t *x*, std::int32_t *y*, std::int32_t *z* ) [inline]

### 10.57.3 Member Data Documentation

#### 10.57.3.1 std::int32_t LLGL::Offset3D::x = 0

Offset X axis.

#### 10.57.3.2 std::int32_t LLGL::Offset3D::y = 0

Offset Y axis.

#### 10.57.3.3 std::int32_t LLGL::Offset3D::z = 0

Offset Z axis.

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

- Types.h

## 10.58 LLGL::OpenGLDependentStateDescriptor Struct Reference

Graphics API dependent state descriptor for the OpenGL renderer.

```
#include <CommandBufferFlags.h>
```

### Public Attributes

- bool originLowerLeft = false

  *Specifies whether the screen-space origin is on the lower-left. By default false.*
- bool invertFrontFace = false

  *Specifies whether to invert front-facing. By default false.*

### 10.58.1 Detailed Description

Graphics API dependent state descriptor for the OpenGL renderer.

**Remarks**

This descriptor is used to compensate a few differences between OpenGL and the other rendering APIs.

**See also**

CommandBuffer::SetGraphicsAPIDependentState

### 10.58.2 Member Data Documentation

#### 10.58.2.1 bool LLGL::OpenGLDependentStateDescriptor::invertFrontFace = false

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`.

#### 10.58.2.2 bool LLGL::OpenGLDependentStateDescriptor::originLowerLeft = false

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:

- CommandBufferFlags.h

## 10.59 LLGL::PipelineLayout Class Reference

Pipeline layout interface.

```
#include <PipelineLayout.h>
```

Inheritance diagram for LLGL::PipelineLayout:

### Additional Inherited Members

### 10.59.1 Detailed Description

Pipeline layout interface.

**Remarks**

An instance of this interface provides all descriptor sets (as called in Vulkan) or descriptor heaps (as called in Direct3D 12) for graphics and compute pipelines.

**See also**

RenderSystem::CreatePipelineLayout
GraphicsPipelineDescriptor::pipelineLayout
ResourceHeapDescriptor::pipelineLayout

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

- PipelineLayout.h

## 10.60 LLGL::PipelineLayoutDescriptor Struct Reference

Pipeline layout descritpor structure.

```
#include <PipelineLayoutFlags.h>
```

### Public Attributes

- std::vector< BindingDescriptor > bindings
  *List of layout resource bindings.*

### 10.60.1 Detailed Description

Pipeline layout descritpor structure.

**Remarks**

Contains all layout bindings that will be used by graphics and compute pipelines.

### 10.60.2 Member Data Documentation

#### 10.60.2.1 std::vector<BindingDescriptor> LLGL::PipelineLayoutDescriptor::bindings

List of layout resource bindings.

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

- PipelineLayoutFlags.h

## 10.61 LLGL::QueryHeap Class Reference

Query heap interface that holds a certain number of queries that are all of the same type.

`#include <QueryHeap.h>`

Inheritance diagram for LLGL::QueryHeap:

### Public Member Functions

- QueryType GetType () const
  *Returns the type of queries within this heap.*

### Protected Member Functions

- QueryHeap (const QueryType type)

### 10.61.1 Detailed Description

Query heap interface that holds a certain number of queries that are all of the same type.

**See also**

RenderSystem::CreateQueryHeap
CommandBuffer::BeginQuery
CommandBuffer::BeginRenderCondition
CommandQueue::QueryResult

### 10.61.2 Constructor & Destructor Documentation

#### 10.61.2.1 LLGL::QueryHeap::QueryHeap ( const QueryType *type* ) [protected]

### 10.61.3 Member Function Documentation

#### 10.61.3.1 QueryType LLGL::QueryHeap::GetType ( ) const [inline]

Returns the type of queries within this heap.

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

- QueryHeap.h

## 10.62 LLGL::QueryHeapDescriptor Struct Reference

Query heap descriptor structure.

```
#include <QueryHeapFlags.h>
```

### Public Attributes

- QueryType type = QueryType::SamplesPassed

  *Specifies the type of queries in the heap. By default QueryType::SamplesPassed.*
- std::uint32_t numQueries = 1

  *Specifies the number of queries in the heap. This must be greater than zero. By default 1.*
- bool renderCondition = false

  *Specifies whether the queries are to be used as render conditions. By default false.*

### 10.62.1 Detailed Description

Query heap descriptor structure.

**See also**

RenderSystem::CreateQueryHeap

### 10.62.2 Member Data Documentation

#### 10.62.2.1 std::uint32_t LLGL::QueryHeapDescriptor::numQueries = 1

Specifies the number of queries in the heap. This must be greater than zero. By default 1.

#### 10.62.2.2 bool LLGL::QueryHeapDescriptor::renderCondition = false

Specifies whether the queries are to be used as render conditions. By default false.

**Remarks**

If this is true, the results of the queries cannot be retrieved by CommandBuffer::QueryResult and the member `type` can only have one of the following values:

- QueryType::SamplesPassed
- QueryType::AnySamplesPassed
- QueryType::AnySamplesPassedConservative
- QueryType::StreamOutOverflow

Render conditions can be used to render complex geometry under the condition that a previous (commonly significantly smaller) geometry has passed the depth and stencil tests.

**See also**

CommandBuffer::BeginRenderCondition
CommandBuffer::EndRenderCondition

**Note**

Only supported with: OpenGL, Direct3D 11, Direct3D 12.

#### 10.62.2.3 QueryType LLGL::QueryHeapDescriptor::type = QueryType::SamplesPassed

Specifies the type of queries in the heap. By default QueryType::SamplesPassed.

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

- QueryHeapFlags.h

## 10.63 LLGL::QueryPipelineStatistics Struct Reference

Query data structure for pipeline statistics.

```
#include <QueryHeapFlags.h>
```

### Public Attributes

- std::uint64_t inputAssemblyVertices = 0

  *Number of vertices submitted to the input-assembly.*
- std::uint64_t inputAssemblyPrimitives = 0

  *Number of primitives submitted to the input-assembly.*
- std::uint64_t vertexShaderInvocations = 0

  *Number of vertex shader invocations.*
- std::uint64_t geometryShaderInvocations = 0

  *Number of geometry shader invocations.*
- std::uint64_t geometryShaderPrimitives = 0

  *Number of primitives generated by the geometry shader.*
- std::uint64_t clippingInvocations = 0

  *Number of primitives that reached the primitive clipping stage.*
- std::uint64_t clippingPrimitives = 0

  *Number of primitives that passed the primitive clipping stage.*
- std::uint64_t fragmentShaderInvocations = 0

  *Number of fragment shader invocations.*
- std::uint64_t tessControlShaderInvocations = 0

  *Number of tessellation-control shader invocations.*
- std::uint64_t tessEvaluationShaderInvocations = 0

  *Number of tessellation-evaluation shader invocations.*
- std::uint64_t computeShaderInvocations = 0

  *Number of compute shader invocations.*

### 10.63.1 Detailed Description

Query data structure for pipeline statistics.

**Remarks**

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

**See also**

QueryType::PipelineStatistics
CommandQueue::QueryResult
RenderingFeatures::hasPipelineStatistics
Direct3D11 counterpart `D3D11_QUERY_DATA_PIPELINE_STATISTICS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_query_data_pipeline_statistics
Direct3D12 counterpart `D3D12_QUERY_DATA_PIPELINE_STATISTICS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d12/ns-d3d12-d3d12_query_data_pipeline_statistics
Vulkan counterpart `VkQueryPipelineStatisticFlagBits`: https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/VkQueryPipelineStatisticFlagBits.html

### 10.63.2 Member Data Documentation

#### 10.63.2.1 std::uint64_t LLGL::QueryPipelineStatistics::clippingInvocations = 0

Number of primitives that reached the primitive clipping stage.

#### 10.63.2.2 std::uint64_t LLGL::QueryPipelineStatistics::clippingPrimitives = 0

Number of primitives that passed the primitive clipping stage.

#### 10.63.2.3 std::uint64_t LLGL::QueryPipelineStatistics::computeShaderInvocations = 0

Number of compute shader invocations.

#### 10.63.2.4 std::uint64_t LLGL::QueryPipelineStatistics::fragmentShaderInvocations = 0

Number of fragment shader invocations.

#### 10.63.2.5 std::uint64_t LLGL::QueryPipelineStatistics::geometryShaderInvocations = 0

Number of geometry shader invocations.

#### 10.63.2.6 std::uint64_t LLGL::QueryPipelineStatistics::geometryShaderPrimitives = 0

Number of primitives generated by the geometry shader.

#### 10.63.2.7 std::uint64_t LLGL::QueryPipelineStatistics::inputAssemblyPrimitives = 0

Number of primitives submitted to the input-assembly.

#### 10.63.2.8 std::uint64_t LLGL::QueryPipelineStatistics::inputAssemblyVertices = 0

Number of vertices submitted to the input-assembly.

#### 10.63.2.9 std::uint64_t LLGL::QueryPipelineStatistics::tessControlShaderInvocations = 0

Number of tessellation-control shader invocations.

#### 10.63.2.10 std::uint64_t LLGL::QueryPipelineStatistics::tessEvaluationShaderInvocations = 0

Number of tessellation-evaluation shader invocations.

#### 10.63.2.11 std::uint64_t LLGL::QueryPipelineStatistics::vertexShaderInvocations = 0

Number of vertex shader invocations.

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

- QueryHeapFlags.h

## 10.64 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.*
- DepthBiasDescriptor depthBias

  *Specifies the parameters to bias fragment depth values.*
- MultiSamplingDescriptor multiSampling

  *(Multi-)sampling descriptor.*
- bool frontCCW = false

  *If enabled, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding. By default disabled.*
- bool discardEnabled = false

  *If enabled, primitives are discarded after optional stream-outputs but before the rasterization stage. By default disabled.*
- bool depthClampEnabled = false

  *If enabled, there is effectively no near and far clipping plane. By default disabled.*
- 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 true, conservative rasterization is enabled. By default disabled.*
- float lineWidth = 1.0f

  *Specifies the width of all generated line primitives. By default 1.0.*

## 10.64.1 Detailed Description

Rasterizer state descriptor structure.

**See also**

GraphicsPipelineDescriptor::rasterizer

## 10.64.2 Member Data Documentation

### 10.64.2.1 bool LLGL::RasterizerDescriptor::antiAliasedLineEnabled = false

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

### 10.64.2.2 bool LLGL::RasterizerDescriptor::conservativeRasterization = false

If true, conservative rasterization is enabled. By default disabled.

**Note**

Only supported with: Direct3D 12, Direct3D 11.3, OpenGL (if the extension `GL_NV_conservative_raster` or `GL_INTEL_conservative_rasterization` is supported).

**See also**

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

#### 10.64.2.3 CullMode LLGL::RasterizerDescriptor::cullMode = CullMode::Disabled

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

#### 10.64.2.4 DepthBiasDescriptor LLGL::RasterizerDescriptor::depthBias

Specifies the parameters to bias fragment depth values.

#### 10.64.2.5 bool LLGL::RasterizerDescriptor::depthClampEnabled = false

If enabled, there is effectively no near and far clipping plane. By default disabled.

#### 10.64.2.6 bool LLGL::RasterizerDescriptor::discardEnabled = false

If enabled, primitives are discarded after optional stream-outputs but before the rasterization stage. By default disabled.

**Note**

Only supported with: OpenGL, Vulkan, Metal.

#### 10.64.2.7 bool LLGL::RasterizerDescriptor::frontCCW = false

If enabled, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding. By default disabled.

#### 10.64.2.8 float LLGL::RasterizerDescriptor::lineWidth = 1.0f

Specifies the width of all generated line primitives. By default 1.0.

**Remarks**

The minimum and maximum supported line width can be determined by the `lineWidthRange` member in the RenderingCapabilities structure. If this line width is out of range, it will be clamped silently during graphics pipeline creation.

**Note**

Only supported with: OpenGL, Vulkan.

**See also**

RenderingLimits::lineWidthRange

#### 10.64.2.9 MultiSamplingDescriptor LLGL::RasterizerDescriptor::multiSampling

(Multi-)sampling descriptor.

#### 10.64.2.10 PolygonMode LLGL::RasterizerDescriptor::polygonMode = PolygonMode::Fill

Polygon render mode. By default PolygonMode::Fill.

#### 10.64.2.11 bool LLGL::RasterizerDescriptor::scissorTestEnabled = false

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

**See also**

CommandBuffer::SetScissor
CommandBuffer::SetScissors

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

- GraphicsPipelineFlags.h

## 10.65 LLGL::RenderContext Class Reference

Render context interface.

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

Inheritance diagram for LLGL::RenderContext:

## Public Member Functions

- Extent2D GetResolution () const final

  *Returns the resolution of the current video mode.*
- std::uint32_t GetNumColorAttachments () const final

  *Returns 1, since each render context has always a single color attachment.*
- bool HasDepthAttachment () const final

  *Returns true if this render context has a depth format.*
- bool HasStencilAttachment () const final

  *Returns true if this render context has a stencil format.*
- virtual void Present ()=0

  *Swaps the back buffer with the front buffer to present it on the screen (or rather on this render context).*
- virtual Format GetColorFormat () const =0

  *Returns the color format of this render context.*
- virtual Format GetDepthStencilFormat () const =0

  *Returns the depth-stencil format of this render context.*
- Surface & GetSurface () const

  *Returns the surface which is used to present the content on the screen.*
- bool SetVideoMode (const VideoModeDescriptor &videoModeDesc)

  *Sets the new video mode for this render context.*
- const VideoModeDescriptor & GetVideoMode () const

  *Returns the video mode for this render context.*
- bool SetVsync (const VsyncDescriptor &vsyncDesc)

  *Sets the new vertical-sychronization (V-sync) configuration for this render context.*
- const VsyncDescriptor & GetVsync () const

  *Returns the V-snyc configuration for this render context.*

## Protected Member Functions

- RenderContext ()=default

  *Default constructor with no effect.*
- RenderContext (const VideoModeDescriptor &initialVideoMode, const VsyncDescriptor &initialVsync)

  *Constructor to initialize the render context with the specified video mode and V-sync.*
- virtual bool OnSetVideoMode (const VideoModeDescriptor &videoModeDesc)=0

  *Callback when the video mode is about to get changed.*
- virtual bool OnSetVsync (const VsyncDescriptor &vsyncDesc)=0

  *Callback when the V-sync is about to get changed.*
- void SetOrCreateSurface (const std::shared_ptr< Surface > &surface, VideoModeDescriptor videoMode↩Desc, const void *windowContext)

  *Sets the render context surface or creates one if 'surface' is null, and switches to fullscreen mode if enabled.*
- void ShareSurfaceAndConfig (RenderContext &other)

  *Shares the surface and video mode with another render context.*
- bool SetDisplayModeByVideoMode (Display &display, const VideoModeDescriptor &videoModeDesc)

  *Sets the display mode for the specified display by the parameters of the video mode descriptor.*
- bool SwitchFullscreenMode (const VideoModeDescriptor &videoModeDesc)

### 10.65.1 Detailed Description

Render context interface.

**Remarks**

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

**See also**

RenderSystem::CreateRenderContext
CommandBuffer::SetRenderTarget(RenderContext&)

### 10.65.2 Constructor & Destructor Documentation

#### 10.65.2.1 LLGL::RenderContext::RenderContext ( ) [protected],[default]

Default constructor with no effect.

#### 10.65.2.2 LLGL::RenderContext::RenderContext ( const VideoModeDescriptor & *initialVideoMode*, const VsyncDescriptor & *initialVsync* ) [protected]

Constructor to initialize the render context with the specified video mode and V-sync.

### 10.65.3 Member Function Documentation

#### 10.65.3.1 virtual Format LLGL::RenderContext::GetColorFormat ( ) const [pure virtual]

Returns the color format of this render context.

**Remarks**

This may depend on the settings specified for the video mode. A common value for a render context color format is Format::BGRA8UNorm.

**See also**

SetVideoMode
AttachmentFormatDescriptor::format
Format

#### 10.65.3.2 virtual Format LLGL::RenderContext::GetDepthStencilFormat ( ) const [pure virtual]

Returns the depth-stencil format of this render context.

**Remarks**

This may depend on the settings specified for the video mode.

**See also**

SetVideoMode
AttachmentFormatDescriptor::format
Format

#### 10.65.3.3 std::uint32_t LLGL::RenderContext::GetNumColorAttachments ( ) const [final], [virtual]

Returns 1, since each render context has always a single color attachment.

Implements LLGL::RenderTarget.

#### 10.65.3.4 Extent2D LLGL::RenderContext::GetResolution ( ) const [final], [virtual]

Returns the resolution of the current video mode.

**See also**

GetVideoMode

Implements LLGL::RenderTarget.

#### 10.65.3.5 Surface& LLGL::RenderContext::GetSurface ( ) const [inline]

Returns the surface which is used to present the content on the screen.

**Remarks**

On desktop platforms, this can be statically casted to 'LLGL::Window&', and on mobile platforms, this can be statically casted to 'LLGL::Canvas&':

```
auto& myWindow = static_cast<LLGL::Window&>(myRenderContext->GetSurface());
```

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

Returns the video mode for this render context.

#### 10.65.3.7 const VsyncDescriptor& LLGL::RenderContext::GetVsync ( ) const [inline]

Returns the V-snyc configuration for this render context.

#### 10.65.3.8 bool LLGL::RenderContext::HasDepthAttachment ( ) const [final], [virtual]

Returns true if this render context has a depth format.

**See also**

GetDepthStencilFormat
IsDepthFormat

Implements LLGL::RenderTarget.

#### 10.65.3.9 bool LLGL::RenderContext::HasStencilAttachment ( ) const [final], [virtual]

Returns true if this render context has a stencil format.

**See also**

GetDepthStencilFormat
IsStencilFormat

Implements LLGL::RenderTarget.

#### 10.65.3.10 virtual bool LLGL::RenderContext::OnSetVideoMode ( const VideoModeDescriptor & *videoModeDesc* ) [protected], [pure virtual]

Callback when the video mode is about to get changed.

**Parameters**

| | | |
|---|---|---|
| in | *videoModeDesc* | Specifies the descriptor of the new video mode. |

**Returns**

True on success, otherwise the previous video mode remains.

**Remarks**

Only if the function returns true, the new video mode will be stored in the member returned by 'GetVideoMode'. This callback is only called when the parameter of 'SetVideoMode' differs from the previous video mode.

**Note**

This video mode may differ from the one passed by 'SetVideoMode' if the platform specific 'Surface' object has modified it for its requirements.

**See also**

SetVideoMode
GetVideoMode
Surface::AdaptForVideoMode

#### 10.65.3.11 virtual bool LLGL::RenderContext::OnSetVsync ( const VsyncDescriptor & *vsyncDesc* ) [protected], [pure virtual]

Callback when the V-sync is about to get changed.

**Parameters**

| | | |
|---|---|---|
| in | *vsyncDesc* | Specifies the descriptor of the new V-sync configuration. |

**Returns**

True on success, otherwise the previous V-sync configuration remains.

#### 10.65.3.12 virtual void LLGL::RenderContext::Present ( ) [pure virtual]

Swaps the back buffer with the front buffer to present it on the screen (or rather on this render context).

#### 10.65.3.13 bool LLGL::RenderContext::SetDisplayModeByVideoMode ( Display & *display*, const VideoModeDescriptor & *videoModeDesc* ) [protected]

Sets the display mode for the specified display by the parameters of the video mode descriptor.

**Returns**

Return value of the Display::SetDisplayMode function.

**See also**

Display::SetDisplayMode

#### 10.65.3.14 void LLGL::RenderContext::SetOrCreateSurface ( const std::shared_ptr< Surface > & *surface*, VideoModeDescriptor *videoModeDesc*, const void * *windowContext* ) [protected]

Sets the render context surface or creates one if 'surface' is null, and switches to fullscreen mode if enabled.

**Parameters**

| | | |
|---|---|---|
| in | *surface* | Optional shared pointer to a surface which will be used as main render target. If this is null, a new surface is created for this render context. |
| in | *videoModeDesc* | Specifies the video mode descriptor. The resolution of this video mode is only used if 'surface' is null, otherwise the resolution is determined by the content size of the specified surface (i.e. with the Surface::GetContentSize function). To determine the final video mode, use the GetVideoMode function. |
| in | *Optional* | pointer to a NativeContextHandle structure. This is only used for desktop platforms. |

**See also**

WindowDescriptor::windowContext
Surface::GetContentSize
GetVideoMode
SwitchFullscreenMode

#### 10.65.3.15 bool LLGL::RenderContext::SetVideoMode ( const VideoModeDescriptor & *videoModeDesc* )

Sets the new video mode for this render context.

**Parameters**

| | | |
|---|---|---|
| in | *videoModeDesc* | Specifies the descriptor of the new video mode. |

**Returns**

True on success, otherwise the specified video mode was invalid (e.g. if the resolution contains a zero).

**Remarks**

When the video mode is changed from fullscreen to non-fullscreen, the previous surface position is restored.

**Note**

This may invalidate the currently set render target if the back buffer is required, so a subsequent call to "↩ CommandBuffer::SetRenderTarget" is necessary!

**See also**

CommandBuffer::SetRenderTarget(RenderContext&)

#### 10.65.3.16 bool LLGL::RenderContext::SetVsync ( const VsyncDescriptor & *vsyncDesc* )

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

**Parameters**

| | | |
|---|---|---|
| in | *vsyncDesc* | Specifies the descriptor of the new V-sync configuration. |

**Returns**

True on success, otherwise the specified V-sync is invalid.

#### 10.65.3.17 void LLGL::RenderContext::ShareSurfaceAndConfig ( RenderContext & *other* ) [protected]

Shares the surface and video mode with another render context.

**Note**

This is only used by the renderer debug layer.

#### 10.65.3.18 bool LLGL::RenderContext::SwitchFullscreenMode ( const VideoModeDescriptor & *videoModeDesc* ) [protected]

Switches the fullscreen mode for the primary display if the specified fullscreen mode is different to the current fullscreen setting.

**See also**

SetDisplayModeByVideoMode
GetVideoMode

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

- RenderContext.h

## 10.66 LLGL::RenderContextDescriptor Struct Reference

Render context descriptor structure.

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

### Public Attributes

- VsyncDescriptor vsync

  *Vertical-synchronization (Vsync) descriptor.*
- MultiSamplingDescriptor multiSampling

  *Multi-sampling descriptor.*
- VideoModeDescriptor videoMode

  *Video mode descriptor.*

### 10.66.1 Detailed Description

Render context descriptor structure.

**See also**

RenderSystem::CreateRenderContext

### 10.66.2 Member Data Documentation

#### 10.66.2.1 MultiSamplingDescriptor LLGL::RenderContextDescriptor::multiSampling

Multi-sampling descriptor.

#### 10.66.2.2 VideoModeDescriptor LLGL::RenderContextDescriptor::videoMode

Video mode descriptor.

#### 10.66.2.3 VsyncDescriptor LLGL::RenderContextDescriptor::vsync

Vertical-synchronization (Vsync) descriptor.

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

- RenderContextFlags.h

## 10.67 LLGL::RendererConfigurationOpenGL Struct Reference

OpenGL profile descriptor structure.

```
#include <RendererConfiguration.h>
```

### Public Attributes

- OpenGLContextProfile contextProfile = OpenGLContextProfile::CoreProfile

  *Specifies the requested OpenGL context profile. By default OpenGLContextProfile::CoreProfile.*
- int majorVersion = 0

  *Specifies the requested OpenGL context major version. By default 0.*
- int minorVersion = 0

  *Specifies the requested OpenGL context minor version. By default 0.*

### 10.67.1 Detailed Description

OpenGL profile descriptor structure.

**Note**

On MacOS the only supported OpenGL profiles are compatibility profile (for lagecy OpenGL before 3.0), 3.2 core profile, or 4.1 core profile.

### 10.67.2 Member Data Documentation

#### 10.67.2.1 OpenGLContextProfile LLGL::RendererConfigurationOpenGL::contextProfile = OpenGLContextProfile::CoreProfile

Specifies the requested OpenGL context profile. By default OpenGLContextProfile::CoreProfile.

#### 10.67.2.2 int LLGL::RendererConfigurationOpenGL::majorVersion = 0

Specifies the requested OpenGL context major version. By default 0.

**Remarks**

If both `majorVersion` and `minorVersion` are 0, the highest OpenGL version that is availabel on the host system will be choosen.
This member is ignored if `contextProfile` is OpenGLContextProfile::CompatibilityProfile.

#### 10.67.2.3 int LLGL::RendererConfigurationOpenGL::minorVersion = 0

Specifies the requested OpenGL context minor version. By default 0.

**Remarks**

If both `majorVersion` and `minorVersion` are 0, the highest OpenGL version that is availabel on the host system will be choosen.
This member is ignored if `contextProfile` is OpenGLContextProfile::CompatibilityProfile.

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

- RendererConfiguration.h

## 10.68 LLGL::RendererConfigurationVulkan Struct Reference

Structure for a Vulkan renderer specific configuration.

```
#include <RendererConfiguration.h>
```

### Public Attributes

- ApplicationDescriptor application

  *Application descriptor used when a Vulkan debug or validation layer is enabled.*
- std::vector< std::string > enabledLayers

  *List of Vulkan layers to enable. The ones that are not supported, will be ignored.*
- std::uint64_t minDeviceMemoryAllocationSize = 1024∗1024

  *Minimal allocation size for a device memory chunk. By default 1024∗1024, i.e. 1 MB of VRAM.*
- bool reduceDeviceMemoryFragmentation = false

  *Specifies whether fragmentation of the device memory blocks shall be kept low. By default false.*

### 10.68.1 Detailed Description

Structure for a Vulkan renderer specific configuration.

**Remarks**

The nomenclature here is "Renderer" instead of "RenderSystem" since the configuration is renderer specific and does not denote a configuration of the entire system.

## 10.68.2 Member Data Documentation

### 10.68.2.1 ApplicationDescriptor LLGL::RendererConfigurationVulkan::application

Application descriptor used when a Vulkan debug or validation layer is enabled.

**See also**

ApplicationDescriptor

### 10.68.2.2 std::vector<std::string> LLGL::RendererConfigurationVulkan::enabledLayers

List of Vulkan layers to enable. The ones that are not supported, will be ignored.

**Remarks**

For example, the layer "`VK_LAYER_KHRONOS_validation`" can be used for a stronger validation.

### 10.68.2.3 std::uint64_t LLGL::RendererConfigurationVulkan::minDeviceMemoryAllocationSize = 1024∗1024

Minimal allocation size for a device memory chunk. By default 1024∗1024, i.e. 1 MB of VRAM.

**Remarks**

Vulkan only allows a limited set of device memory objects (e.g. 4096 on a GPU with 8 GB of VRAM). This member specifies the minimum size used for hardware memory allocation of such a memory chunk. The Vulkan render system automatically manages sub-region allocation and defragmentation.

**Todo** Remove this as soon as Vulkan memory manage has been improved.

### 10.68.2.4 bool LLGL::RendererConfigurationVulkan::reduceDeviceMemoryFragmentation = false

Specifies whether fragmentation of the device memory blocks shall be kept low. By default false.

**Remarks**

If this is true, each buffer and image allocation first tries to find a reusable device memory block within a single VkDeviceMemory chunk (which might be potentially slower). Whenever a VkDeviceMemory chunk is full, the memory manager tries to reduce fragmentation anyways.

**Todo** Remove this as soon as Vulkan memory manage has been improved.

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

- RendererConfiguration.h

## 10.69 LLGL::RendererID Struct Reference

Renderer identification number enumeration.

```
#include <RenderSystemFlags.h>
```

### Static Public Attributes

- static const int Undefined = 0x00000000

  *Undefined ID number.*
- static const int OpenGL = 0x00000001

  *ID number for an OpenGL renderer.*
- static const int OpenGLES1 = 0x00000002

  *ID number for an OpenGL ES 1 renderer.*
- static const int OpenGLES2 = 0x00000003

  *ID number for an OpenGL ES 2 renderer.*
- static const int OpenGLES3 = 0x00000004

  *ID number for an OpenGL ES 3 renderer.*
- static const int Direct3D9 = 0x00000005

  *ID number for a Direct3D 9 renderer.*
- static const int Direct3D10 = 0x00000006

  *ID number for a Direct3D 10 renderer.*
- static const int Direct3D11 = 0x00000007

  *ID number for a Direct3D 11 renderer.*
- static const int Direct3D12 = 0x00000008

  *ID number for a Direct3D 12 renderer.*
- static const int Vulkan = 0x00000009

  *ID number for a Vulkan renderer.*
- static const int Metal = 0x0000000a

  *ID number for a Metal renderer.*
- static const int Reserved = 0x000000ff

  *Highest ID number for reserved future renderers. Value is 0x000000ff.*

### 10.69.1 Detailed Description

Renderer identification number enumeration.

**Remarks**

There are several IDs for reserved future renderes, which are currently not supported (and maybe never supported). You can use an ID greater than 'RendererID::Reserved' (which has a value of 0x000000ff) for your own renderer. Or use one of the pre-defined IDs if you want to implement your own OpenGL/ Direct3D or whatever renderer.

**See also**

RendererInfo::rendererID

### 10.69.2 Member Data Documentation

#### 10.69.2.1 const int LLGL::RendererID::Direct3D10 = 0x00000006 [static]

ID number for a Direct3D 10 renderer.

#### 10.69.2.2 const int LLGL::RendererID::Direct3D11 = 0x00000007 [static]

ID number for a Direct3D 11 renderer.

#### 10.69.2.3 const int LLGL::RendererID::Direct3D12 = 0x00000008 [static]

ID number for a Direct3D 12 renderer.

#### 10.69.2.4 const int LLGL::RendererID::Direct3D9 = 0x00000005 [static]

ID number for a Direct3D 9 renderer.

#### 10.69.2.5 const int LLGL::RendererID::Metal = 0x0000000a [static]

ID number for a Metal renderer.

#### 10.69.2.6 const int LLGL::RendererID::OpenGL = 0x00000001 [static]

ID number for an OpenGL renderer.

#### 10.69.2.7 const int LLGL::RendererID::OpenGLES1 = 0x00000002 [static]

ID number for an OpenGL ES 1 renderer.

#### 10.69.2.8 const int LLGL::RendererID::OpenGLES2 = 0x00000003 [static]

ID number for an OpenGL ES 2 renderer.

#### 10.69.2.9 const int LLGL::RendererID::OpenGLES3 = 0x00000004 [static]

ID number for an OpenGL ES 3 renderer.

#### 10.69.2.10 const int LLGL::RendererID::Reserved = 0x000000ff [static]

Highest ID number for reserved future renderers. Value is 0x000000ff.

#### 10.69.2.11 const int LLGL::RendererID::Undefined = 0x00000000 [static]

Undefined ID number.

#### 10.69.2.12 const int LLGL::RendererID::Vulkan = 0x00000009 [static]

ID number for a Vulkan renderer.

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

- RenderSystemFlags.h

## 10.70 LLGL::RendererInfo Struct Reference

Renderer basic information structure.

```
#include <RenderSystemFlags.h>
```

### Public Attributes

- std::string rendererName

  *Rendering API name and version (e.g. "OpenGL 4.6").*
- 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").*

### 10.70.1 Detailed Description

Renderer basic information structure.

### 10.70.2 Member Data Documentation

#### 10.70.2.1 std::string LLGL::RendererInfo::deviceName

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

#### 10.70.2.2 std::string LLGL::RendererInfo::rendererName

Rendering API name and version (e.g. "OpenGL 4.6").

#### 10.70.2.3 std::string LLGL::RendererInfo::shadingLanguageName

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

#### 10.70.2.4 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

## 10.71 LLGL::RenderingCapabilities Struct Reference

Structure with all attributes describing the rendering capabilities of the render system.

```
#include <RenderSystemFlags.h>
```

### Public Attributes

- ScreenOrigin screenOrigin = ScreenOrigin::UpperLeft

  *Screen coordinate system origin.*
- ClippingRange clippingRange = ClippingRange::ZeroToOne

  *Specifies the clipping depth range.*
- std::vector< ShadingLanguage > shadingLanguages

  *Specifies the list of supported shading languages.*
- std::vector< Format > textureFormats

  *Specifies the list of supported texture formats.*
- RenderingFeatures features

  *Specifies all supported hardware features.*
- RenderingLimits limits

  *Specifies all rendering limitations.*

### 10.71.1 Detailed Description

Structure with all attributes describing the rendering capabilities of the render system.

**See also**

RenderSystem::GetRenderingCaps

### 10.71.2 Member Data Documentation

#### 10.71.2.1 ClippingRange LLGL::RenderingCapabilities::clippingRange = ClippingRange::ZeroToOne

Specifies the clipping depth range.

#### 10.71.2.2 RenderingFeatures LLGL::RenderingCapabilities::features

Specifies all supported hardware features.

**Remarks**

Especially with OpenGL these features can vary between different hardware and GL versions.

#### 10.71.2.3 RenderingLimits LLGL::RenderingCapabilities::limits

Specifies all rendering limitations.

**Remarks**

Especially with OpenGL these features can vary between different hardware and GL versions.

#### 10.71.2.4 ScreenOrigin LLGL::RenderingCapabilities::screenOrigin = ScreenOrigin::UpperLeft

Screen coordinate system origin.

**Remarks**

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

#### 10.71.2.5 std::vector<ShadingLanguage> LLGL::RenderingCapabilities::shadingLanguages

Specifies the list of supported shading languages.

**Remarks**

This also specifies whether shaders can be loaded in source or binary form (using "Compile" or "LoadBinary" functions of the "Shader" interface).

**See also**

Shader::Compile
Shader::LoadBinary

#### 10.71.2.6 std::vector<Format> LLGL::RenderingCapabilities::textureFormats

Specifies the list of supported texture formats.

**See also**

Format

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

- RenderSystemFlags.h

## 10.72 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 SetDebugGroup (const char *name)

  *Sets the new debug group 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

- virtual void OnError (ErrorType type, Message &message)

  *Callback function when an error was posted.*
- virtual void OnWarning (WarningType type, Message &message)

  *Callback function when a warning was posted.*

### 10.72.1 Detailed Description

Rendering debugger interface.

#### Remarks

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

### 10.72.2 Constructor & Destructor Documentation

#### 10.72.2.1 virtual LLGL::RenderingDebugger::~RenderingDebugger ( ) [virtual]

### 10.72.3 Member Function Documentation

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

Callback function when an error was posted.

**Remarks**

Use the 'message' parameter to block further occurences of this error if you like. The following example shows a custom implementation that is equivalent to the default implementation:

```
class MyDebugger : public LLGL::RenderingDebugger {
    void OnError(ErrorType type, Message& message) override {
        LLGL::Log::PostReport(
            LLGL::Log::ReportType::Error,
            "ERROR (" + std::string(LLGL::ToString(type)) + "): in '" + message.GetSource() +
       "': " + message.GetText()
        );
        message.Block();
    }
};
```

**See also**

RenderingDebugger::PostError
OnWarning

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

Callback function when a warning was posted.

**See also**

RenderingDebugger::PostWarning
OnError

#### 10.72.3.3 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. |

#### 10.72.3.4 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. |

#### 10.72.3.5 void LLGL::RenderingDebugger::SetDebugGroup ( const char * *name* )

Sets the new debug group name.

**Parameters**

| | | |
|---|---|---|
| in | *name* | Pointer to a null terminated string that specifies the name. If this is null, the debug group is disabled. |

**Note**

This function only stores the pointer. Hence, the pointer must be valid until a new pointer or a null pointer is set.

#### 10.72.3.6 void LLGL::RenderingDebugger::SetSource ( const char * *source* )

Sets the new source function name.

**Parameters**

| | | |
|---|---|---|
| in | *source* | Pointer to a null terminated string that specifies the name. If this is null, the source is disabled. |

**Note**

This function only stores the pointer. Hence, the pointer must be valid until a new pointer or a null pointer is set.

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

- RenderingDebugger.h

## 10.73 LLGL::RenderingFeatures Struct Reference

Contains the attributes for all supported rendering features.

```
#include <RenderSystemFlags.h>
```

## Public Attributes

- bool hasDirectResourceBinding = false

  *Specifies whether the render system supports binding individual resources directly instead of a resource heap. This does not exclude support of resource heaps, because these are always supported.*
- bool hasRenderTargets = false

  *Specifies whether render targets (also "framebuffer objects") are supported.*
- bool has3DTextures = false

  *Specifies whether 3D textures are supported.*
- bool hasCubeTextures = false

  *Specifies whether cube textures are supported.*
- bool hasArrayTextures = false

  *Specifies whether 1D- and 2D array textures are supported.*
- bool hasCubeArrayTextures = false

  *Specifies whether cube array textures are supported.*
- bool hasMultiSampleTextures = false

  *Specifies whether multi-sample textures are supported.*
- bool hasTextureViews = false

  *Specifies whether texture views are supported.*
- bool hasTextureViewSwizzle = false

  *Specifies whether texture views can have swizzling (a.k.a. component mapping).*
- 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.*
- bool hasGeometryShaders = false

  *Specifies whether geometry shaders are supported.*
- bool hasTessellationShaders = false

  *Specifies whether tessellation shaders are supported.*
- bool hasComputeShaders = false

  *Specifies whether compute shaders are supported.*
- bool hasInstancing = false

  *Specifies whether hardware instancing is supported.*
- bool hasOffsetInstancing = false

  *Specifies whether hardware instancing with instance offsets is supported.*
- bool hasIndirectDrawing = false

  *Specifies whether indirect draw commands are supported.*
- bool hasViewportArrays = false

  *Specifies whether multiple viewports, depth-ranges, and scissors at once are supported.*
- bool hasConservativeRasterization = false

  *Specifies whether conservative rasterization is supported.*
- bool hasStreamOutputs = false

  *Specifies whether stream-output is supported.*
- bool hasLogicOp = false

  *Specifies whether logic fragment operations are supported.*
- bool hasPipelineStatistics = false

  *Specifies whether queries for pipeline statistics are supported.*
- bool hasRenderCondition = false

  *Specifies whether queries for conditional rendering are supported.*

### 10.73.1 Detailed Description

Contains the attributes for all supported rendering features.

**See also**

RenderingCapabilities::features

### 10.73.2 Member Data Documentation

#### 10.73.2.1 bool LLGL::RenderingFeatures::has3DTextures = false

Specifies whether 3D textures are supported.

**See also**

TextureType::Texture3D

#### 10.73.2.2 bool LLGL::RenderingFeatures::hasArrayTextures = false

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

**See also**

TextureType::Texture1DArray
TextureType::Texture2DArray

#### 10.73.2.3 bool LLGL::RenderingFeatures::hasComputeShaders = false

Specifies whether compute shaders are supported.

**See also**

ShaderType::Compute
CommandBuffer::Dispatch
CommandBuffer::DispatchIndirect

#### 10.73.2.4 bool LLGL::RenderingFeatures::hasConservativeRasterization = false

Specifies whether conservative rasterization is supported.

**See also**

RasterizerDescriptor::conservativeRasterization

#### 10.73.2.5 bool LLGL::RenderingFeatures::hasConstantBuffers = false

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

**See also**

BindFlags::ConstantBuffer

#### 10.73.2.6 bool LLGL::RenderingFeatures::hasCubeArrayTextures = false

Specifies whether cube array textures are supported.

**See also**

TextureType::TextureCubeArray

#### 10.73.2.7 bool LLGL::RenderingFeatures::hasCubeTextures = false

Specifies whether cube textures are supported.

**See also**

TextureType::TextureCube

#### 10.73.2.8 bool LLGL::RenderingFeatures::hasDirectResourceBinding = false

Specifies whether the render system supports binding individual resources directly instead of a resource heap. This does not exclude support of resource heaps, because these are always supported.

**See also**

CommandBuffer::SetResource
CommandBuffer::ResetResourceSlots

#### 10.73.2.9 bool LLGL::RenderingFeatures::hasGeometryShaders = false

Specifies whether geometry shaders are supported.

**See also**

ShaderType::Geometry

#### 10.73.2.10 bool LLGL::RenderingFeatures::hasIndirectDrawing = false

Specifies whether indirect draw commands are supported.

**See also**

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect

#### 10.73.2.11 bool LLGL::RenderingFeatures::hasInstancing = false

Specifies whether hardware instancing is supported.

**See also**

CommandBuffer::DrawInstanced(std::uint32_t, std::uint32_t, std::uint32_t)
CommandBuffer::DrawIndexedInstanced(std::uint32_t, std::uint32_t, std::uint32_t)
CommandBuffer::DrawIndexedInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::int32_t)

#### 10.73.2.12 bool LLGL::RenderingFeatures::hasLogicOp = false

Specifies whether logic fragment operations are supported.

**Note**

For Direct3D 11, feature level 11.1 is required.

**See also**

BlendDescriptor::logicOp

#### 10.73.2.13 bool LLGL::RenderingFeatures::hasMultiSampleTextures = false

Specifies whether multi-sample textures are supported.

**See also**

TextureType::Texture2DMS
TextureType::Texture2DMSArray

#### 10.73.2.14 bool LLGL::RenderingFeatures::hasOffsetInstancing = false

Specifies whether hardware instancing with instance offsets is supported.

**See also**

CommandBuffer::DrawInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::uint32_t)
CommandBuffer::DrawIndexedInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::int32_t, std::uint32_t)

#### 10.73.2.15 bool LLGL::RenderingFeatures::hasPipelineStatistics = false

Specifies whether queries for pipeline statistics are supported.

**See also**

QueryType::PipelineStatistics
QueryPipelineStatistics

#### 10.73.2.16 bool LLGL::RenderingFeatures::hasRenderCondition = false

Specifies whether queries for conditional rendering are supported.

**See also**

QueryHeapDescriptor::renderCondition
CommandBuffer:BeginRenderCondition

#### 10.73.2.17 bool LLGL::RenderingFeatures::hasRenderTargets = false

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

#### 10.73.2.18 bool LLGL::RenderingFeatures::hasSamplers = false

Specifies whether samplers are supported.

#### 10.73.2.19 bool LLGL::RenderingFeatures::hasStorageBuffers = false

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

**See also**

BindFlags::Sampled
BindFlags::Storage

#### 10.73.2.20 bool LLGL::RenderingFeatures::hasStreamOutputs = false

Specifies whether stream-output is supported.

**See also**

ShaderSource::streamOutput
CommandBuffer::BeginStreamOutput

#### 10.73.2.21 bool LLGL::RenderingFeatures::hasTessellationShaders = false

Specifies whether tessellation shaders are supported.

**See also**

ShaderType::TessControl
ShaderType::TessEvaluation

#### 10.73.2.22 bool LLGL::RenderingFeatures::hasTextureViews = false

Specifies whether texture views are supported.

**Remarks**

Texture views can share their image data with another texture resource.

**See also**

TextureViewDescriptor

#### 10.73.2.23 bool LLGL::RenderingFeatures::hasTextureViewSwizzle = false

Specifies whether texture views can have swizzling (a.k.a. component mapping).

**Remarks**

This feature implies that `hasTextureViews` is true.

**See also**

TextureViewDescriptor::swizzle

#### 10.73.2.24 bool LLGL::RenderingFeatures::hasUniforms = false

Specifies whether individual shader uniforms are supported.

**Note**

Only supported with: OpenGL.

**See also**

CommandBuffer::SetUniform
CommandBuffer::SetUniforms

#### 10.73.2.25 bool LLGL::RenderingFeatures::hasViewportArrays = false

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

**See also**

RenderingLimits::maxViewports

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

- RenderSystemFlags.h

## 10.74 LLGL::RenderingLimits Struct Reference

Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.

```
#include <RenderSystemFlags.h>
```

### Public Attributes

- float lineWidthRange [2] = { 1.0f, 1.0f }

  *Specifies the range for rasterizer line widths. By default [1, 1].*
- std::uint32_t maxTextureArrayLayers = 0

  *Specifies the maximum number of texture array layers (for 1D-, 2D-, and cube textures).*
- std::uint32_t maxColorAttachments = 0

  *Specifies the maximum number of color attachments for each render target.*
- std::uint32_t maxPatchVertices = 0

  *Specifies the maximum number of patch control points.*
- std::uint32_t max1DTextureSize = 0

  *Specifies the maximum size of each 1D texture.*
- std::uint32_t max2DTextureSize = 0

  *Specifies the maximum size of each 2D texture (for width and height).*
- std::uint32_t max3DTextureSize = 0

  *Specifies the maximum size of each 3D texture (for width, height, and depth).*
- std::uint32_t maxCubeTextureSize = 0

  *Specifies the maximum size of each cube texture (for width and height).*
- std::uint32_t maxAnisotropy = 0

  *Specifies the maximum anisotropy texture filter.*
- std::uint32_t maxComputeShaderWorkGroups [3] = { 0, 0, 0 }

  *Specifies the maximum number of work groups in a compute shader.*
- std::uint32_t maxComputeShaderWorkGroupSize [3] = { 0, 0, 0 }

  *Specifies the maximum work group size in a compute shader.*
- std::uint32_t maxViewports = 0

  *Specifies the maximum number of viewports and scissor rectangles. Most render systems have a maximum of 16.*
- std::uint32_t maxViewportSize [2] = { 0, 0 }

  *Specifies the maximum width and height of each viewport and scissor rectangle.*
- std::uint64_t maxBufferSize = 0

  *Specifies the maximum size (in bytes) that is supported for hardware buffers (vertex, index, storage buffers).*
- std::uint64_t maxConstantBufferSize = 0

  *Specifies the maximum size (in bytes) that is supported for hardware constant buffers.*

### 10.74.1 Detailed Description

Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.

**See also**

RenderingCapabilities::limits

### 10.74.2 Member Data Documentation

#### 10.74.2.1 float LLGL::RenderingLimits::lineWidthRange[2] = { 1.0f, 1.0f }

Specifies the range for rasterizer line widths. By default [1, 1].

**Note**

Only supported with: OpenGL, Vulkan.

**See also**

RasterizerDescriptor::lineWidth

#### 10.74.2.2 std::uint32_t LLGL::RenderingLimits::max1DTextureSize = 0

Specifies the maximum size of each 1D texture.

**See also**

TextureDescriptor::extent

#### 10.74.2.3 std::uint32_t LLGL::RenderingLimits::max2DTextureSize = 0

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

**See also**

TextureDescriptor::extent

#### 10.74.2.4 std::uint32_t LLGL::RenderingLimits::max3DTextureSize = 0

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

**See also**

TextureDescriptor::extent

#### 10.74.2.5 std::uint32_t LLGL::RenderingLimits::maxAnisotropy = 0

Specifies the maximum anisotropy texture filter.

**See also**

SamplerDescriptor::maxAnisotropy

#### 10.74.2.6 std::uint64_t LLGL::RenderingLimits::maxBufferSize = 0

Specifies the maximum size (in bytes) that is supported for hardware buffers (vertex, index, storage buffers).

**Remarks**

Constant buffers are a special case for which 'maxConstantBufferSize' can be used.

**See also**

BufferDescriptor::size
maxConstantBufferSize

#### 10.74.2.7 std::uint32_t LLGL::RenderingLimits::maxColorAttachments = 0

Specifies the maximum number of color attachments for each render target.

**Remarks**

This value must not be greater than 8.

**See also**

RenderTargetDescriptor::attachments
RenderPassDescriptor::colorAttachments
BlendDescriptor::targets

#### 10.74.2.8 std::uint32_t LLGL::RenderingLimits::maxComputeShaderWorkGroups[3] = { 0, 0, 0 }

Specifies the maximum number of work groups in a compute shader.

**See also**

CommandBuffer::Dispatch

#### 10.74.2.9 std::uint32_t LLGL::RenderingLimits::maxComputeShaderWorkGroupSize[3] = { 0, 0, 0 }

Specifies the maximum work group size in a compute shader.

#### 10.74.2.10 std::uint64_t LLGL::RenderingLimits::maxConstantBufferSize = 0

Specifies the maximum size (in bytes) that is supported for hardware constant buffers.

**Remarks**

This is typically a lot smaller than the maximum size for other types of buffers.

**See also**

BufferDescriptor::size

#### 10.74.2.11 std::uint32_t LLGL::RenderingLimits::maxCubeTextureSize = 0

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

**See also**

TextureDescriptor::extent

#### 10.74.2.12 std::uint32_t LLGL::RenderingLimits::maxPatchVertices = 0

Specifies the maximum number of patch control points.

**See also**

PrimitiveTopology::Patches1
PrimitiveTopology::Patches32

#### 10.74.2.13 std::uint32_t LLGL::RenderingLimits::maxTextureArrayLayers = 0

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

**See also**

TextureDescriptor::arrayLayers

#### 10.74.2.14 std::uint32_t LLGL::RenderingLimits::maxViewports = 0

Specifies the maximum number of viewports and scissor rectangles. Most render systems have a maximum of 16.

**See also**

CommandBuffer::SetViewports
CommandBuffer::SetScissors
GraphicsPipelineDescriptor::viewports
GraphicsPipelineDescriptor::scissors
RenderingFeatures::hasViewportArrays

#### 10.74.2.15 std::uint32_t LLGL::RenderingLimits::maxViewportSize[2] = { 0, 0 }

Specifies the maximum width and height of each viewport and scissor rectangle.

**See also**

Viewport::width
Viewport::height
Scissor::width
Scissor::height

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

- RenderSystemFlags.h

## 10.75 LLGL::RenderingProfiler Class Reference

Rendering profiler model class.

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

### Public Member Functions

- void NextProfile (FrameProfile *outputProfile=nullptr)

  *Returns the current frame profile and resets the counters for the next frame.*
- void Accumulate (const FrameProfile &profile)

  *Accumulates the specified profile with the current values.*

### Public Attributes

- FrameProfile frameProfile

  *Current frame profile with all counter values.*

### 10.75.1 Detailed Description

Rendering profiler model class.

**Remarks**

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

**Todo** Refactor this for the new ResourceHeap and RenderPass interfaces.

### 10.75.2 Member Function Documentation

#### 10.75.2.1 void LLGL::RenderingProfiler::Accumulate ( const FrameProfile & *profile* )

Accumulates the specified profile with the current values.

**Parameters**

| | | |
|---|---|---|
| in | *profile* | Specifies the input profile whose values are to be merged with the current values. |

**See also**

FrameProfile::Accumulate

#### 10.75.2.2 void LLGL::RenderingProfiler::NextProfile ( FrameProfile * *outputProfile* = `nullptr` )

Returns the current frame profile and resets the counters for the next frame.

**Parameters**

| | | |
|---|---|---|
| out | *outputProfile* | Optional pointer to an output profile to retrieve the current values. By default null. |

### 10.75.3 Member Data Documentation

#### 10.75.3.1 FrameProfile LLGL::RenderingProfiler::frameProfile

Current frame profile with all counter values.

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

- RenderingProfiler.h

## 10.76 LLGL::RenderPass Class Reference

Render pass interface.

```
#include <RenderPass.h>
```

Inheritance diagram for LLGL::RenderPass:

### Additional Inherited Members

### 10.76.1 Detailed Description

Render pass interface.

**Remarks**

A render pass is a high level construct adopted from Vulkan and Metal. It is used to tell the driver the various segments of a frame and which render target attachments are used and which depend on each other.

**See also**

RenderSystem::CreateRenderPass
CommandBuffer::BeginRenderPass
CommandBuffer::EndRenderPass

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

- RenderPass.h

## 10.77 LLGL::RenderPassDescriptor Struct Reference

Render pass descriptor structure.

```
#include <RenderPassFlags.h>
```

### Public Attributes

- std::vector< AttachmentFormatDescriptor > colorAttachments

  *Specifies the color attachments used within the render pass.*
- AttachmentFormatDescriptor depthAttachment

  *Specifies the depth attachment used within the render pass.*
- AttachmentFormatDescriptor stencilAttachment

  *Specifies the stencil attachment used within the render pass.*

### 10.77.1 Detailed Description

Render pass descriptor structure.

**Remarks**

A render pass object can be used across multiple render targets. Moreover, a render target can be created with a different render pass object than the one used for CommandBuffer::BeginRenderPass as long as they are compatible. Two render passes are considered compatible when all color-, depth-, and stencil attachments are compatible.

**See also**

RenderSystem::CreateRenderPass
CommandBuffer::BeginRenderPass
AttachmentFormatDescriptor

## 10.77.2 Member Data Documentation

### 10.77.2.1 std::vector<AttachmentFormatDescriptor> LLGL::RenderPassDescriptor::colorAttachments

Specifies the color attachments used within the render pass.

**Remarks**

A render context usually uses a BGRA format instead of an RGBA format.

**See also**

RenderingLimits::maxColorAttachments
Format::BGRA8UNorm
Format::BGRA8sRGB

**Todo** Change to static array with 8 elements, i.e. `AttachmentFormatDescriptor color↩ Attachments[8]`.

### 10.77.2.2 AttachmentFormatDescriptor LLGL::RenderPassDescriptor::depthAttachment

Specifies the depth attachment used within the render pass.

**Remarks**

The depth attachment and stencil attachment usually share the same format (e.g. Format::D24UNormS8UInt). They are separated here to specify different load and store operations.

### 10.77.2.3 AttachmentFormatDescriptor LLGL::RenderPassDescriptor::stencilAttachment

Specifies the stencil attachment used within the render pass.

**Remarks**

The depth attachment and stencil attachment usually share the same format (e.g. Format::D24UNormS8UInt). They are separated here to specify different load and store operations.

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

- RenderPassFlags.h

## 10.78 LLGL::RenderSystem Class Reference

Render system interface.

```
#include <RenderSystem.h>
```

Inheritance diagram for LLGL::RenderSystem:

### Public Member Functions

- int GetRendererID () const

  *Rendering API identification number.*
- const std::string & GetName () const

  *Returns the name of this render system.*
- const RendererInfo & GetRendererInfo () const

  *Returns basic renderer information.*
- const RenderingCapabilities & 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< Surface > &surface={})=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 CommandQueue * GetCommandQueue ()=0

  *Returns the single instance of the command queue.*
- virtual CommandBuffer * CreateCommandBuffer (const CommandBufferDescriptor &desc={})=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 (std::uint32_t 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 &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint64_t dataSize)=0

*Updates the data of the specified buffer.*

- virtual void * MapBuffer (Buffer &buffer, const CPUAccess 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 SrcImageDescriptor *image↩Desc=nullptr)=0

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

  *Releases the specified texture object. After this call, the specified object must no longer be used.*
- virtual void WriteTexture (Texture &texture, const TextureRegion &textureRegion, const SrcImageDescriptor &imageDesc)=0

  *Updates the image data of the specified texture.*
- virtual void ReadTexture (const Texture &texture, std::uint32_t mipLevel, const DstImageDescriptor &image↩Desc)=0

  *Reads the image data from the specified texture.*
- virtual Sampler * CreateSampler (const SamplerDescriptor &desc)=0

  *Creates a new Sampler object.*
- virtual void Release (Sampler &sampler)=0

  *Releases the specified Sampler object. After this call, the specified object must no longer be used.*
- virtual ResourceHeap * CreateResourceHeap (const ResourceHeapDescriptor &desc)=0

  *Creates a new resource heap.*
- virtual void Release (ResourceHeap &resourceHeap)=0

  *Releases the specified ResourceHeap object. After this call, the specified object must no longer be used.*
- virtual RenderPass * CreateRenderPass (const RenderPassDescriptor &desc)=0

  *Creates a new RenderPass object.*
- virtual void Release (RenderPass &renderPass)=0

  *Releases the specified RenderPass 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 ShaderDescriptor &desc)=0

  *Creates a new and Shader object and compiles the specified source.*
- virtual ShaderProgram * CreateShaderProgram (const ShaderProgramDescriptor &desc)=0

  *Creates a new shader program and links all specified shaders.*
- 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 PipelineLayout * CreatePipelineLayout (const PipelineLayoutDescriptor &desc)=0

  *Creates a new and initialized pipeline layout object, if and only if the renderer supports pipeline layouts.*
- virtual void Release (PipelineLayout &pipelineLayout)=0

  *Releases the specified PipelineLayout 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 QueryHeap * CreateQueryHeap (const QueryHeapDescriptor &desc)=0

  *Creates a new query heap.*

- virtual void Release (QueryHeap &queryHeap)=0

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

- virtual Fence * CreateFence ()=0

  *Creates a new fence (used for CPU/GPU synchronization).*

- virtual void Release (Fence &fence)=0

  *Releases the specified Fence 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.*

- static std::unique_ptr< RenderSystem > Load (const RenderSystemDescriptor &renderSystemDesc, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr)

  *Loads a new render system from the specified module.*

- static void Unload (std::unique_ptr< RenderSystem > &&renderSystem)

  *Unloads the specified render system and the internal module.*

## Protected Member Functions

- RenderSystem ()=default
- void SetRendererInfo (const RendererInfo &info)

  *Sets the renderer information.*

- void SetRenderingCaps (const RenderingCapabilities &caps)

  *Sets the rendering capabilities.*

- void AssertCreateBuffer (const BufferDescriptor &desc, std::uint64_t maxSize)

  *Validates the specified buffer descriptor to be used for buffer creation.*

- void AssertCreateBufferArray (std::uint32_t numBuffers, Buffer *const *bufferArray)

  *Validates the specified arguments to be used for buffer array creation.*

- void AssertCreateShader (const ShaderDescriptor &desc)

  *Validates the specified shader descriptor.*

- void AssertCreateShaderProgram (const ShaderProgramDescriptor &desc)

  *Validates the specified shader program descriptor.*

- void AssertCreateRenderTarget (const RenderTargetDescriptor &desc)

  *Validates the specified render target descriptor.*

- void AssertCreateRenderPass (const RenderPassDescriptor &desc)

  *Validates the specified render pass descriptor.*

- void AssertImageDataSize (std::size_t dataSize, std::size_t requiredDataSize, const char *info=nullptr)

  *Validates the specified image data size against the required size (in bytes).*

### 10.78.1 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...", "Read...", "Map...", "Unmap...", 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);
```

### 10.78.2 Constructor & Destructor Documentation

#### 10.78.2.1 LLGL::RenderSystem::RenderSystem ( ) [protected],[default]

### 10.78.3 Member Function Documentation

#### 10.78.3.1 void LLGL::RenderSystem::AssertCreateBuffer ( const BufferDescriptor & *desc*, std::uint64_t *maxSize* ) [protected]

Validates the specified buffer descriptor to be used for buffer creation.

#### 10.78.3.2 void LLGL::RenderSystem::AssertCreateBufferArray ( std::uint32_t *numBuffers*, Buffer *const * *bufferArray* ) [protected]

Validates the specified arguments to be used for buffer array creation.

#### 10.78.3.3 void LLGL::RenderSystem::AssertCreateRenderPass ( const RenderPassDescriptor & *desc* ) [protected]

Validates the specified render pass descriptor.

#### 10.78.3.4 void LLGL::RenderSystem::AssertCreateRenderTarget ( const RenderTargetDescriptor & *desc* ) [protected]

Validates the specified render target descriptor.

#### 10.78.3.5 void LLGL::RenderSystem::AssertCreateShader ( const ShaderDescriptor & *desc* ) [protected]

Validates the specified shader descriptor.

#### 10.78.3.6 void LLGL::RenderSystem::AssertCreateShaderProgram ( const ShaderProgramDescriptor & *desc* ) [protected]

Validates the specified shader program descriptor.

#### 10.78.3.7 void LLGL::RenderSystem::AssertImageDataSize ( std::size_t *dataSize*, std::size_t *requiredDataSize*, const char * *info* = nullptr ) [protected]

Validates the specified image data size against the required size (in bytes).

#### 10.78.3.8 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

#### 10.78.3.9 virtual BufferArray* LLGL::RenderSystem::CreateBufferArray ( std::uint32_t *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**

All buffers within this array must have the same binding flags. 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. |
| *std::invalid_argument* | If bufferArray is null. |
| *std::invalid_argument* | If any of the pointers in the array are null. |
| *std::invalid_argument* | If not all buffers have the same binding flags. |

**See also**

BufferDescriptor::bindFlags

#### 10.78.3.10 virtual CommandBuffer* LLGL::RenderSystem::CreateCommandBuffer ( const CommandBufferDescriptor & *desc* = {} ) [pure virtual]

Creates a new command buffer.

**Remarks**

All render systems can create multiple command buffers, but especially for the legacy graphics APIs such as OpenGL and Direct3D 11, this doesn't provide any benefit, since all graphics and compute commands are submitted sequentially to the GPU.

#### 10.78.3.11 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

#### 10.78.3.12 virtual Fence* LLGL::RenderSystem::CreateFence ( ) [pure virtual]

Creates a new fence (used for CPU/GPU synchronization).

**See also**

CommandBuffer::SubmitFence
CommandBuffer::WaitFence

#### 10.78.3.13 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

#### 10.78.3.14 virtual PipelineLayout* LLGL::RenderSystem::CreatePipelineLayout ( const PipelineLayoutDescriptor & *desc* ) [pure virtual]

Creates a new and initialized pipeline layout object, if and only if the renderer supports pipeline layouts.

**Parameters**

| | | |
|---|---|---|
| in | *desc* | Specifies the pipeline layout descriptor with all layout bindings. |

**Remarks**

A pipeline layout is required in combination with a ResourceHeap to bind multiple resources at once. For modern graphics APIs (i.e. Direct3D 12 and Vulkan), this is only way to bind shader resources. For legacy graphics APIs (i.e. Direct3D 11 and OpenGL), shader resources can also be bound individually with the extended command buffer.

**Returns**

Pointer to the new PipelineLayout object or null if the renderer does not support pipeline layouts.

**See also**

CreateResourceHeap

#### 10.78.3.15 virtual QueryHeap* LLGL::RenderSystem::CreateQueryHeap ( const QueryHeapDescriptor & *desc* ) [pure virtual]

Creates a new query heap.

#### 10.78.3.16 virtual RenderContext* LLGL::RenderSystem::CreateRenderContext ( const RenderContextDescriptor & *desc,* const std::shared_ptr< Surface > & *surface* = { } ) [pure virtual]

Creates a new render context and returns the raw pointer.

**Parameters**

| | | |
|---|---|---|
| in | *desc* | Specifies the render context descriptor, which contains the video mode, vsync, multi-sampling settings etc. |
| in | *surface* | Optional shared pointer to a surface for the render context. If this is null, the render context will create its own platform specific surface, which can be accessed by RenderContext::GetSurface. The default surface is not shown automatically. |

**See also**

RenderContext::GetSurface

#### 10.78.3.17 virtual RenderPass* LLGL::RenderSystem::CreateRenderPass ( const RenderPassDescriptor & *desc* ) [pure virtual]

Creates a new RenderPass object.

**Returns**

Pointer to the new RenderPass object or null if the render system does not use render passes. In the case of the latter, null pointers are allowed for render passes.

**See also**

RenderTargetDescriptor::renderPass
GraphicsPipelineDescriptor::renderPass
CommandBuffer::BeginRenderPass
CommandBuffer::EndRenderPass

#### 10.78.3.18 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). |

#### 10.78.3.19 virtual ResourceHeap* LLGL::RenderSystem::CreateResourceHeap ( const ResourceHeapDescriptor & *desc* ) [pure virtual]

Creates a new resource heap.

**Parameters**

| | | |
|---|---|---|
| in | *desc* | Specifies the descriptor which determines all shader resource. |

**Remarks**

Resources heaps are used in combination with a pipeline layout. The pipeline layout determines to which binding points the resources are bound.

**See also**

CreatePipelineLayout
CommandBuffer::SetGraphicsResourceHeap
CommandBuffer::SetComputeResourceHeap

#### 10.78.3.20 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**

GetRenderingCaps

#### 10.78.3.21 virtual Shader* LLGL::RenderSystem::CreateShader ( const ShaderDescriptor & *desc* ) [pure virtual]

Creates a new and Shader object and compiles the specified source.

**Remarks**

To check whether the compilation was successful or not, use the `HasErrors` and `GetReport` functions of the Shader interface.

**See also**

Shader::HasErrors
Shader::GetReport
ShaderDescriptor
ShaderDescFromFile

#### 10.78.3.22 virtual ShaderProgram* LLGL::RenderSystem::CreateShaderProgram ( const ShaderProgramDescriptor & *desc* ) [pure virtual]

Creates a new shader program and links all specified shaders.

**Remarks**

To check whether the linking was successful or not, use the `HasErrors` and `GetReport` functions of the ShaderProgram interface.

**See also**

ShaderProgram::HasErrors
ShaderProgram::GetReport
ShaderProgramDescriptor
ShaderProgramDesc

#### 10.78.3.23 virtual Texture* LLGL::RenderSystem::CreateTexture ( const TextureDescriptor & *textureDesc*, const SrcImageDescriptor * *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 feature is enabled). 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

#### 10.78.3.24 static std::vector<std::string> LLGL::RenderSystem::FindModules ( ) `[static]`

Returns the list of all available render system modules for the current platform.

**Remarks**

For example, on Win32 this might be { "OpenGL", "Direct3D11", "Direct3D12" }, but on MacOS it might be only { "OpenGL" }.

#### 10.78.3.25 virtual CommandQueue* LLGL::RenderSystem::GetCommandQueue ( ) `[pure virtual]`

Returns the single instance of the command queue.

#### 10.78.3.26 const RenderSystemConfiguration& LLGL::RenderSystem::GetConfiguration ( ) const `[inline]`

Returns the basic configuration.

**See also**

SetConfiguration

#### 10.78.3.27 const std::string& LLGL::RenderSystem::GetName ( ) const [inline]

Returns the name of this render system.

#### 10.78.3.28 int LLGL::RenderSystem::GetRendererID ( ) const [inline]

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 have its own ID number.

**See also**

RendererID

#### 10.78.3.29 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!

#### 10.78.3.30 const RenderingCapabilities& 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!

#### 10.78.3.31 static std::unique_ptr<RenderSystem> LLGL::RenderSystem::Load ( const RenderSystemDescriptor & *renderSystemDesc*, RenderingProfiler * *profiler* = nullptr, RenderingDebugger * *debugger* = nullptr ) [static]

Loads a new render system from the specified module.

**Parameters**

| | | |
|---|---|---|
| in | *renderSystemDesc* | Specifies the render system descriptor structure. The 'moduleName' member of this strucutre must not be empty. |
| in | *profiler* | Optional pointer to a rendering profiler. This is only supported if LLGL was compiled with the LLGL_ENABLE_DEBUG_LAYER flag. If this is used, the counters of the profiler must be reset manually. |
| in | *debugger* | Optional pointer to a rendering debugger. This is only supported if LLGL was compiled with the LLGL_ENABLE_DEBUG_LAYER flag. If the default debugger is used (i.e. no sub class of RenderingDebugger), then all reports will be send to the Log. In order to see any reports from the Log, use either |

#### Remarks

The descriptor structure can be initialized by only the module name like shown in the following example:

```
// Load the "OpenGL" render system module
auto myRenderSystem = LLGL::RenderSystem::Load("OpenGL");
```

The debugger and profiler can be used like this:

```
// Forward all log reports to the standard output stream for errors
LLGL::Log::SetReportCallbackStd(std::cerr);

// Declare profiler and debugger (these classes can also be extended)
LLGL::RenderingProfiler myProfiler;
LLGL::RenderingDebugger myDebugger;

// Load the "Direct3D11" render system module
auto myRenderSystem = LLGL::RenderSystem::Load("Direct3D11", &myProfiler, &
      myDebugger);
```

#### Exceptions

| | |
|---|---|
| *std::runtime_error* | If loading the render system from the specified module failed. |

#### See also

RenderSystemDescriptor::moduleName

### 10.78.3.32 virtual void* LLGL::RenderSystem::MapBuffer ( Buffer & *buffer*, const CPUAccess *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

### 10.78.3.33 virtual void LLGL::RenderSystem::ReadTexture ( const Texture & *texture*, std::uint32_t *mipLevel*, const DstImageDescriptor & *imageDesc* ) [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 texture data. |
| out | *imageDesc* | Specifies the destination image descriptor to write the texture data to. |

**Remarks**

The required size for a successful texture read operation depends on the image format, data type, and texture size. The Texture::GetDesc or Texture::GetMipExtent functions can be used to determine the texture dimensions.

```
// Query texture size attribute
auto myTextureExtent = myTexture->GetMipExtent(0);

// Allocate image buffer with elements in all dimensions
std::vector<LLGL::ColorRGBAub> myImage(myTextureExtent.width * myTextureExtent.height * myTextureExtent.
      depth);

// Initialize destination image descriptor
const DstImageDescriptor myImageDesc {
    LLGL::ImageFormat::RGBA,                    // RGBA image format, since we used
       LLGL::ColorRGBAub
    LLGL::DataType::UInt8,                      // 8-bit unsigned integral data type:
       <std::uint8_t> or <unsigned char>
    myImage.data(),                             // Output image buffer
    myImage.size() * sizeof(LLGL::ColorRGBAub)  // Image buffer size: number of color
       elements and size of each color element
};

// Read texture data from first MIP-map level (index 0)
myRenderSystem->ReadTexture(*myTexture, 0, myImageDesc);
```

**Note**

The behavior is undefined if `imageDesc.data` points to an invalid buffer, or `imageDesc.data` points to a buffer that is smaller than specified by `imageDesc.dataSize`, or `imageDesc.dataSize` is less than the required size.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If `imageDesc.data` is null. |

**See also**

Texture::GetDesc
Texture::GetMipExtent

**Todo** Replace `mipLevel` parameter with `textureRegion` just like with the `WriteTexture` function.

#### 10.78.3.34 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.

#### 10.78.3.35 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.

**See also**

CreateCommandBuffer

#### 10.78.3.36 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.

#### 10.78.3.37 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.

#### 10.78.3.38 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.

#### 10.78.3.39 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.

#### 10.78.3.40 virtual void LLGL::RenderSystem::Release ( ResourceHeap & *resourceHeap* ) [pure virtual]

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

#### 10.78.3.41 virtual void LLGL::RenderSystem::Release ( RenderPass & *renderPass* ) [pure virtual]

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

#### 10.78.3.42 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.

#### 10.78.3.43 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.

#### 10.78.3.44 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.

#### 10.78.3.45 virtual void LLGL::RenderSystem::Release ( PipelineLayout & *pipelineLayout* ) [pure virtual]

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

#### 10.78.3.46 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.

#### 10.78.3.47 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.

#### 10.78.3.48 virtual void LLGL::RenderSystem::Release ( QueryHeap & *queryHeap* ) [pure virtual]

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

#### 10.78.3.49 virtual void LLGL::RenderSystem::Release ( Fence & *fence* ) [pure virtual]

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

#### 10.78.3.50 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

#### 10.78.3.51 void LLGL::RenderSystem::SetRendererInfo ( const RendererInfo & *info* ) [protected]

Sets the renderer information.

#### 10.78.3.52 void LLGL::RenderSystem::SetRenderingCaps ( const RenderingCapabilities & *caps* ) [protected]

Sets the rendering capabilities.

#### 10.78.3.53 static void LLGL::RenderSystem::Unload ( std::unique_ptr< RenderSystem > && *renderSystem* ) [static]

Unloads the specified render system and the internal module.

**Remarks**

After this call, the specified render system and all the objects associated to it must no longer be used!

#### 10.78.3.54 virtual void LLGL::RenderSystem::UnmapBuffer ( Buffer & *buffer* ) [pure virtual]

Unmaps the specified buffer.

**See also**

MapBuffer

#### 10.78.3.55 virtual void LLGL::RenderSystem::WriteBuffer ( Buffer & *dstBuffer*, std::uint64_t *dstOffset*, const void * *data*, std::uint64_t *dataSize* ) [pure virtual]

Updates the data of the specified buffer.

**Parameters**

| | | |
|---|---|---|
| in | *dstBuffer* | Specifies the destination buffer whose data is to be updated. |
| in | *dstOffset* | 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. |
| 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. |

**Remarks**

To update a small buffer (maximum of 65536 bytes) during encoding a command buffer, use CommandBuffer::UpdateBuffer.

#### 10.78.3.56 virtual void LLGL::RenderSystem::WriteTexture ( Texture & *texture*, const TextureRegion & *textureRegion*, const SrcImageDescriptor & *imageDesc* ) [pure virtual]

Updates the image data of the specified texture.

**Parameters**

| | | |
|---|---|---|
| in | *texture* | Specifies the texture whose data is to be updated. |
| in | *textureRegion* | Specifies the texture region where the texture is to be updated. The field TextureRegion::numMipLevels **must** be 1. |
| 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

## 10.79 LLGL::RenderSystemChild Class Reference

Base class for all interfaces whoes instances are owned by the RenderSystem.

```
#include <RenderSystemChild.h>
```

Inheritance diagram for LLGL::RenderSystemChild:

### Public Member Functions

- virtual void SetName (const char *name)

  *Sets the name of this class instance for debugging purposes.*

### Additional Inherited Members

### 10.79.1 Detailed Description

Base class for all interfaces whoes instances are owned by the RenderSystem.

### 10.79.2 Member Function Documentation

#### 10.79.2.1 virtual void LLGL::RenderSystemChild::SetName ( const char * *name* ) [virtual]

Sets the name of this class instance for debugging purposes.

**Parameters**

| | | |
|---|---|---|
| in | *name* | Pointer to a null terminated string that specifies the new name of this instance. Specifying a null pointer effectively removes the name from the object. The implementation of this function may alter the actual name depending on how many internal objects need to be labeled. |

**Remarks**

This is used for debugging purposes only and the implementation is undefined, i.e. if the respective render system does not support debug labels this function call will be ignored silently.
This function is especially useful in conjunction with diagnistoc tools (such as `RenderDoc`) to better identify objects in the event history.
The default implementation has no effect.

**See also**

CommandBuffer::PushDebugGroup
CommandBuffer::PopDebugGroup
RenderSystemDescriptor::debugCallback

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

- RenderSystemChild.h

## 10.80 LLGL::RenderSystemConfiguration Struct Reference

Render system configuration structure.

```
#include <RenderSystemFlags.h>
```

### Public Attributes

- std::size_t threadCount = Constants::maxThreadCount

  *Specifies the number of threads that will be used internally by the render system. By default Constants::maxThread↩ Count.*

### 10.80.1 Detailed Description

Render system configuration structure.

**See also**

RenderSystem::SetConfiguration

### 10.80.2 Member Data Documentation

#### 10.80.2.1 std::size_t LLGL::RenderSystemConfiguration::threadCount = Constants::maxThreadCount

Specifies the number of threads that will be used internally by the render system. By default Constants::max↩ ThreadCount.

**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**

Constants::maxThreadCount

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

- RenderSystemFlags.h

## 10.81 LLGL::RenderSystemDescriptor Struct Reference

Render system descriptor structure.

```
#include <RenderSystemFlags.h>
```

### Public Member Functions

- RenderSystemDescriptor ()=default
- RenderSystemDescriptor (const RenderSystemDescriptor &)=default
- RenderSystemDescriptor & operator= (const RenderSystemDescriptor &)=default
- RenderSystemDescriptor (const std::string &moduleName)

  *Constructor to initialize the descriptor with the module name form an std::string.*
- RenderSystemDescriptor (const char *moduleName)

  *Constructor to initialize the descriptor with the module name form a null terminated string.*

### Public Attributes

- std::string moduleName

  *Specifies the name from which the new render system is to be loaded.*
- DebugCallback debugCallback

  *Debuging callback function object.*
- const void * rendererConfig = nullptr

  *Optional raw pointer to a renderer specific configuration structure.*
- std::size_t rendererConfigSize = 0

  *Specifies the size (in bytes) of the structure where the 'rendererConfig' member points to (use 'sizeof' with the respective structure). By default 0.*

## 10.81.1 Detailed Description

Render system descriptor structure.

#### Remarks

This can be used for some refinements of a specific renderer, e.g. to configure the Vulkan device memory manager.

#### See also

RenderSystem::Load

## 10.81.2 Constructor & Destructor Documentation

### 10.81.2.1 LLGL::RenderSystemDescriptor::RenderSystemDescriptor ( ) [default]

### 10.81.2.2 LLGL::RenderSystemDescriptor::RenderSystemDescriptor ( const RenderSystemDescriptor & ) [default]

### 10.81.2.3 LLGL::RenderSystemDescriptor::RenderSystemDescriptor ( const std::string & *moduleName* ) [inline]

Constructor to initialize the descriptor with the module name form an std::string.

### 10.81.2.4 LLGL::RenderSystemDescriptor::RenderSystemDescriptor ( const char * *moduleName* ) [inline]

Constructor to initialize the descriptor with the module name form a null terminated string.

## 10.81.3 Member Function Documentation

### 10.81.3.1 RenderSystemDescriptor& LLGL::RenderSystemDescriptor::operator= ( const RenderSystemDescriptor & ) [default]

## 10.81.4 Member Data Documentation

### 10.81.4.1 DebugCallback LLGL::RenderSystemDescriptor::debugCallback

Debuging callback function object.

#### 10.81.4.2 std::string LLGL::RenderSystemDescriptor::moduleName

Specifies the name from which the new render system is to be loaded.

**Remarks**

This denotes a shared 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. If LLGL was built with the `LLGL_BUILD_STATIC_LIB` option, this member is ignored.

#### 10.81.4.3 const void∗ LLGL::RenderSystemDescriptor::rendererConfig = nullptr

Optional raw pointer to a renderer specific configuration structure.

**Remarks**

This can be used to pass some refinement configurations to the render system when the module is loaded. Example usage (for Vulkan renderer):

```
// Initialize Vulkan specific configurations (e.g. always allocate at least 1GB of VRAM for each device
      memory chunk).
LLGL::RendererConfigurationVulkan config;
config.minDeviceMemoryAllocationSize = 1024*1024*1024;

// Initialize render system descriptor
LLGL::RenderSystemDescriptor rendererDesc;
rendererDesc.moduleName         = "Vulkan";
rendererDesc.rendererConfig     = &config;
rendererDesc.rendererConfigSize = sizeof(config);

// Load Vulkan render system
auto renderer = LLGL::RenderSystem::Load(rendererDesc);
```

**See also**

rendererConfigSize
RendererConfigurationVulkan
RendererConfigurationOpenGL

#### 10.81.4.4 std::size_t LLGL::RenderSystemDescriptor::rendererConfigSize = 0

Specifies the size (in bytes) of the structure where the 'rendererConfig' member points to (use 'sizeof' with the respective structure). By default 0.

**Remarks**

If 'rendererConfig' is null then this member is ignored.

**See also**

rendererConfig

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

- RenderSystemFlags.h

## 10.82 LLGL::RenderTarget Class Reference

Render target interface.

```
#include <RenderTarget.h>
```

Inheritance diagram for LLGL::RenderTarget:

### Public Member Functions

- bool IsRenderContext () const

  *Returns true if this render target is an instance of RenderContext.*
- virtual Extent2D GetResolution () const =0

  *Returns the render target resolution.*
- virtual std::uint32_t GetNumColorAttachments () const =0

  *Returns the number of color attachments of this render target. This can also be zero.*
- virtual bool HasDepthAttachment () const =0

  *Returns true if this render target has a depth or depth-stencil attachment.*
- virtual bool HasStencilAttachment () const =0

  *Returns true if this render target has a stencil or depth-stencil attachment.*
- virtual const RenderPass * GetRenderPass () const =0

  *Returns the RenderPass object this render target is associated with, or null if render passes are optional for the the render system.*

### Protected Member Functions

- void ValidateResolution (const Extent2D &resolution)

  *Applies the specified resolution.*
- void ValidateMipResolution (const Texture &texture, std::uint32_t mipLevel)

  *Applies the resolution of the texture MIP level.*

### Friends

- class RenderContext

### 10.82.1 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. The only interface that inherits from this interface is RenderContext, a special case of render targets used to present the result on the screen.

**See also**

RenderSystem::CreateRenderTarget
CommandBuffer::SetRenderTarget(RenderTarget&)
RenderContext

### 10.82.2 Member Function Documentation

#### 10.82.2.1 virtual std::uint32_t LLGL::RenderTarget::GetNumColorAttachments ( ) const [pure virtual]

Returns the number of color attachments of this render target. This can also be zero.

**Remarks**

For a render context, this will always be 1.

**See also**

RenderContext::GetColorFormat

Implemented in LLGL::RenderContext.

#### 10.82.2.2 virtual const RenderPass* LLGL::RenderTarget::GetRenderPass ( ) const [pure virtual]

Returns the RenderPass object this render target is associated with, or null if render passes are optional for the the render system.

**Remarks**

This is either the RenderPass object that was passed to the descriptor when this render target was created, or it is the default RenderPass object that was created by the render target itself.

**See also**

RenderTargetDescriptor::renderPass

#### 10.82.2.3 virtual Extent2D LLGL::RenderTarget::GetResolution ( ) const [pure virtual]

Returns the render target resolution.

**Remarks**

This is either determined by the resolution specified in the render target descriptor, or by the video mode of the render context.

**See also**

RenderContext::GetVideoMode
RenderTargetDescriptor::resolution
VideoModeDescriptor::resolution

Implemented in LLGL::RenderContext.

#### 10.82.2.4 virtual bool LLGL::RenderTarget::HasDepthAttachment ( ) const [pure virtual]

Returns true if this render target has a depth or depth-stencil attachment.

**Remarks**

The return value depends on whether the rendering API supports depth-stencil formats where the depth and stencil components can be strictly separated. For example, if the render target was created with only a stencil attachment, LLGL may still create a depth-stencil buffer that results in both a depth and stencil component in one attachment.

**See also**

RenderContext::GetDepthStencilFormat

Implemented in LLGL::RenderContext.

#### 10.82.2.5 virtual bool LLGL::RenderTarget::HasStencilAttachment ( ) const [pure virtual]

Returns true if this render target has a stencil or depth-stencil attachment.

**Remarks**

The return value depends on whether the rendering API supports depth-stencil formats where the depth and stencil components can be strictly separated. For example, if the render target was created with only a stencil attachment, LLGL may still create a depth-stencil buffer that results in both a depth and stencil component in one attachment.

**See also**

RenderContext::GetDepthStencilFormat

Implemented in LLGL::RenderContext.

#### 10.82.2.6 bool LLGL::RenderTarget::IsRenderContext ( ) const

Returns true if this render target is an instance of RenderContext.

**Todo** Replace by IsInstanceOf

#### 10.82.2.7 void LLGL::RenderTarget::ValidateMipResolution ( const Texture & *texture*, std::uint32_t *mipLevel* ) [protected]

Applies the resolution of the texture MIP level.

**See also**

Texture::GetMipExtent
ValidateResolution

#### 10.82.2.8 void LLGL::RenderTarget::ValidateResolution ( const Extent2D & *resolution* ) [protected]

Applies the specified resolution.

**Remarks**

This shoudl be called for each attachment.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If one of the resolution components is zero. |
| *std::invalid_argument* | If the internal resolution has already been set and the input resolution is not equal to that previous resolution. |

### 10.82.3 Friends And Related Function Documentation

#### 10.82.3.1 friend class RenderContext [friend]

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

- RenderTarget.h

## 10.83 LLGL::RenderTargetDescriptor Struct Reference

Render target descriptor structure.

```
#include <RenderTargetFlags.h>
```

## Public Attributes

- const RenderPass * renderPass = nullptr

  *Optional render pass object that will be used with the render target. By default null.*
- Extent2D resolution

  *Specifies the resolution of the render targets.*
- MultiSamplingDescriptor multiSampling

  *Multi-sampling descriptor. By default, multi-sampling is disabled.*
- bool customMultiSampling = false

  *Specifies whether custom multi-sampling is used or not. By default false.*
- std::vector< AttachmentDescriptor > attachments

  *Specifies all render target attachment descriptors.*

### 10.83.1 Detailed Description

Render target descriptor structure.

#### Remarks

Here is a small example of a render target descriptor with a color attachmnet and an anonymous depth attachment (i.e. without a texture reference, which is only allowed for depth/stencil attachments):

```
LLGL::RenderTargetDescriptor myRenderTargetDesc;

auto myRenderTargetSize = myColorTexture->GetMipExtent(0);
myRenderTargetDesc.resolution = { myRenderTargetSize.width, myRenderTargetSize.height };

myRenderTargetDesc.attachments = {
    LLGL::AttachmentDescriptor {
      LLGL::AttachmentType::Color, myColorTexture },
    LLGL::AttachmentDescriptor {
      LLGL::AttachmentType::Depth },
};

auto myRenderTarget = myRenderer->CreateRenderTarget(myRenderTargetDesc);
```

#### See also

RenderSystem::CreateRenderTarget

### 10.83.2 Member Data Documentation

#### 10.83.2.1 std::vector<AttachmentDescriptor> LLGL::RenderTargetDescriptor::attachments

Specifies all render target attachment descriptors.

#### Remarks

This container can also be empty, if the respective fragment shader has no direct output but writes into a storage texture instead (e.g. `image3D` in GLSL, or `RWTexture3D<float4>` in HLSL). If the respective rendering API does not support render targets without any attachments, LLGL will generate a dummy texture.

#### 10.83.2.2 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.

#### 10.83.2.3 MultiSamplingDescriptor LLGL::RenderTargetDescriptor::multiSampling

Multi-sampling descriptor. By default, multi-sampling is disabled.

#### 10.83.2.4 const RenderPass* LLGL::RenderTargetDescriptor::renderPass = nullptr

Optional render pass object that will be used with the render target. By default null.

**Remarks**

If this is null, a default render pass is created for the render target. The default render pass determines the attachment formats by the render target attachments and keeps the load and store operations at its default values.

**See also**

RenderSystem::CreateRenderPass
AttachmentFormatDescriptor::loadOp
AttachmentFormatDescriptor::storeOp

#### 10.83.2.5 Extent2D LLGL::RenderTargetDescriptor::resolution

Specifies the resolution of the render targets.

**Remarks**

All attachments with a reference to a texture must have the same resolution, i.e. the specified array layer and MIP-map level must have the same extent.

**See also**

Texture::GetMipExtent

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

- RenderTargetFlags.h

## 10.84 LLGL::Resource Class Reference

Base class for all hardware resource interfaces.

`#include <Resource.h>`

Inheritance diagram for LLGL::Resource:

### Public Member Functions

- virtual ResourceType GetResourceType () const =0

  *Returns the type of this resource object.*

### Additional Inherited Members

### 10.84.1 Detailed Description

Base class for all hardware resource interfaces.

**See also**

Buffer
Texture
Sampler

### 10.84.2 Member Function Documentation

#### 10.84.2.1 virtual ResourceType LLGL::Resource::GetResourceType ( ) const [pure virtual]

Returns the type of this resource object.

**Remarks**

This is queried by a virtual function call, so the resource type does not need to be stored per instance.

**See also**

ResourceType

**Todo** (Maybe) replace by IsInstanceOf

Implemented in LLGL::Texture, LLGL::Buffer, and LLGL::Sampler.

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

- Resource.h

## 10.85 LLGL::ResourceHeap Class Reference

Resource heap interface.

```
#include <ResourceHeap.h>
```

Inheritance diagram for LLGL::ResourceHeap:

### Additional Inherited Members

### 10.85.1 Detailed Description

Resource heap interface.

#### Remarks

An instance of this interface provides a descriptor set (as called in Vulkan) or descriptor heap (as called in Direct3D 12) for graphics and compute pipelines.

#### See also

RenderSystem::CreateResourceHeap
CommandBuffer::SetGraphicsResourceHeap
CommandBuffer::SetComputeResourceHeap

**Todo** Maybe rename to "ResourceViewHeap" again?

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

- ResourceHeap.h

## 10.86 LLGL::ResourceHeapDescriptor Struct Reference

Resource heap descriptor structure.

```
#include <ResourceHeapFlags.h>
```

### Public Attributes

- PipelineLayout * pipelineLayout = nullptr

  *Reference to the pipeline layout. This must not be null, when a resource heap is created.*
- std::vector< ResourceViewDescriptor > resourceViews

  *List of all resource view descriptors.*

## 10.86.1 Detailed Description

Resource heap descriptor structure.

**Remarks**

For the render systems of modern graphics APIs (i.e. Vulkan and Direct3D 12), a resource heap is the only way to bind hardware resources to a shader pipeline. The resource heap is a container for one or more resources such as textures, samplers, constant buffers etc.

**See also**

RenderSystem::CreateResourceHeap

## 10.86.2 Member Data Documentation

### 10.86.2.1 PipelineLayout* LLGL::ResourceHeapDescriptor::pipelineLayout = nullptr

Reference to the pipeline layout. This must not be null, when a resource heap is created.

### 10.86.2.2 std::vector<ResourceViewDescriptor> LLGL::ResourceHeapDescriptor::resourceViews

List of all resource view descriptors.

**Remarks**

These resources must be specified in the same order as they were specified when the pipeline layout was created.

**See also**

PipelineLayoutDescriptor::bindings

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

- ResourceHeapFlags.h

## 10.87 LLGL::ResourceViewDescriptor Struct Reference

Resource view descriptor structure.

```
#include <ResourceHeapFlags.h>
```

### Public Member Functions

- ResourceViewDescriptor ()=default

  *Default constructor to initialize the resource view with a null pointer.*
- ResourceViewDescriptor (Resource *resource)

  *Constructor to initialize the descriptor with a resource. The resource view will access the entire resource.*
- ResourceViewDescriptor (Texture *texture, const TextureViewDescriptor &subresourceDesc)

  *Constructor to initialize a descriptor with a texture subresource view.*

### Public Attributes

- Resource * resource = nullptr

  *Pointer to the hardware resoudce.*
- TextureViewDescriptor textureView

  *Optional texture view descriptor.*

### 10.87.1 Detailed Description

Resource view descriptor structure.

**See also**

ResourceHeapDescriptor::resourceViews

### 10.87.2 Constructor & Destructor Documentation

#### 10.87.2.1 LLGL::ResourceViewDescriptor::ResourceViewDescriptor ( ) [default]

Default constructor to initialize the resource view with a null pointer.

#### 10.87.2.2 LLGL::ResourceViewDescriptor::ResourceViewDescriptor ( Resource * *resource* ) [inline]

Constructor to initialize the descriptor with a resource. The resource view will access the entire resource.

#### 10.87.2.3 LLGL::ResourceViewDescriptor::ResourceViewDescriptor ( Texture * *texture*, const TextureViewDescriptor & *subresourceDesc* ) [inline]

Constructor to initialize a descriptor with a texture subresource view.

### 10.87.3 Member Data Documentation

#### 10.87.3.1 Resource* LLGL::ResourceViewDescriptor::resource = nullptr

Pointer to the hardware resoudce.

#### 10.87.3.2 TextureViewDescriptor LLGL::ResourceViewDescriptor::textureView

Optional texture view descriptor.

**Remarks**

Can be used to declare a subresource view of a texture resource.
This attribute is ignored if either `numMipLevels` or `numArrayLayers` of the TextureSubresource structure is zero.

**Todo** Not implemented yet.

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

- ResourceHeapFlags.h

## 10.88 LLGL::Sampler Class Reference

Sampler interface.

```
#include <Sampler.h>
```

Inheritance diagram for LLGL::Sampler:

### Public Member Functions

- ResourceType GetResourceType () const override final

  *Returns ResourceType::Sampler.*

### Protected Member Functions

- Sampler ()=default

### 10.88.1 Detailed Description

Sampler interface.

**See also**

RenderSystem::CreateSampler

### 10.88.2 Constructor & Destructor Documentation

#### 10.88.2.1 LLGL::Sampler::Sampler ( ) [protected],[default]

### 10.88.3 Member Function Documentation

#### 10.88.3.1 ResourceType LLGL::Sampler::GetResourceType ( ) const [final],[override],[virtual]

Returns ResourceType::Sampler.

Implements LLGL::Resource.

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

- Sampler.h

## 10.89 LLGL::SamplerDescriptor Struct Reference

Texture sampler descriptor structure.

```
#include <SamplerFlags.h>
```

### Public Attributes

- SamplerAddressMode addressModeU = SamplerAddressMode::Repeat

  *Sampler address mode in U direction (also X axis). By default SamplerAddressMode::Repeat.*
- SamplerAddressMode addressModeV = SamplerAddressMode::Repeat

  *Sampler address mode in V direction (also Y axis). By default SamplerAddressMode::Repeat.*
- SamplerAddressMode addressModeW = SamplerAddressMode::Repeat

  *Sampler address mode in W direction (also Z axis). By default SamplerAddressMode::Repeat.*
- SamplerFilter minFilter = SamplerFilter::Linear

  *Minification filter. By default SamplerFilter::Linear.*
- SamplerFilter magFilter = SamplerFilter::Linear

  *Magnification filter. By default SamplerFilter::Linear.*
- SamplerFilter mipMapFilter = SamplerFilter::Linear

  *MIP-mapping filter. By default SamplerFilter::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.*
- std::uint32_t maxAnisotropy = 1

  *Maximal anisotropy in the range [1, 16].*
- bool compareEnabled = 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).*

## 10.89.1 Detailed Description

Texture sampler descriptor structure.

**See also**

RenderSystem::CreateSampler

## 10.89.2 Member Data Documentation

### 10.89.2.1 SamplerAddressMode LLGL::SamplerDescriptor::addressModeU = SamplerAddressMode::Repeat

Sampler address mode in U direction (also X axis). By default SamplerAddressMode::Repeat.

### 10.89.2.2 SamplerAddressMode LLGL::SamplerDescriptor::addressModeV = SamplerAddressMode::Repeat

Sampler address mode in V direction (also Y axis). By default SamplerAddressMode::Repeat.

#### 10.89.2.3 SamplerAddressMode LLGL::SamplerDescriptor::addressModeW = SamplerAddressMode::Repeat

Sampler address mode in W direction (also Z axis). By default SamplerAddressMode::Repeat.

#### 10.89.2.4 ColorRGBAf LLGL::SamplerDescriptor::borderColor = { 0.0f, 0.0f, 0.0f, 0.0f }

Border color. By default black (0, 0, 0, 0).

**Note**

For Vulkan and Metal, only three predefined border colors are supported:

- Transparenty black: `{0,0,0,0}`
- Opaque black: `{0,0,0,1}`
- Opaque white: `{1,1,1,1}`

#### 10.89.2.5 bool LLGL::SamplerDescriptor::compareEnabled = false

Specifies whether the compare operation for depth textures is to be used or not. By default false.

#### 10.89.2.6 CompareOp LLGL::SamplerDescriptor::compareOp = CompareOp::Less

Compare operation for depth textures. By default CompareOp::Less.

#### 10.89.2.7 SamplerFilter LLGL::SamplerDescriptor::magFilter = SamplerFilter::Linear

Magnification filter. By default SamplerFilter::Linear.

#### 10.89.2.8 std::uint32_t LLGL::SamplerDescriptor::maxAnisotropy = 1

Maximal anisotropy in the range [1, 16].

#### 10.89.2.9 float LLGL::SamplerDescriptor::maxLOD = 1000.0f

Upper end of the MIP-map range. Must be greater than or equal to `minLOD`. By default 1000.

#### 10.89.2.10 SamplerFilter LLGL::SamplerDescriptor::minFilter = SamplerFilter::Linear

Minification filter. By default SamplerFilter::Linear.

#### 10.89.2.11 float LLGL::SamplerDescriptor::minLOD = 0.0f

Lower end of the MIP-map range. By default 0.

#### 10.89.2.12 SamplerFilter LLGL::SamplerDescriptor::mipMapFilter = SamplerFilter::Linear

MIP-mapping filter. By default SamplerFilter::Linear.

#### 10.89.2.13 float LLGL::SamplerDescriptor::mipMapLODBias = 0.0f

MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.
For Metal, the LOD bias can only be specified within the shader code.

#### 10.89.2.14 bool LLGL::SamplerDescriptor::mipMapping = true

Specifies whether MIP-maps are used or not. By default true.

**Remarks**

The number of MIP-maps a texture has is specified by the TextureDescriptor::mipLevels attribute.

**See also**

TextureDescriptor::mipLevels

**Todo** Rename to `mipMapEnabled`.

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

- SamplerFlags.h

## 10.90 LLGL::Scissor Struct Reference

Scissor dimensions.

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

### Public Member Functions

- Scissor ()=default
- Scissor (const Scissor &)=default
- Scissor (std::int32_t x, std::int32_t y, std::int32_t width, std::int32_t height)

  *Scissor constructor with parameters for all attributes.*
- Scissor (const Offset2D &offset, const Extent2D &extent)

  *Scissor constructor with offset and extent parameters.*

### Public Attributes

- std::int32_t x = 0

  *Left-top X coordinate.*
- std::int32_t y = 0

  *Left-top Y coordinate.*
- std::int32_t width = 0

  *Right-bottom width.*
- std::int32_t height = 0

  *Right-bottom height.*

## 10.90.1 Detailed Description

Scissor dimensions.

**Remarks**

A scissor is in screen coordinates where the origin is in the left-top corner.

**See also**

CommandBuffer::SetScissor
CommandBuffer::SetScissors
GraphicsPipelineDescriptor::scissors

## 10.90.2 Constructor & Destructor Documentation

### 10.90.2.1 LLGL::Scissor::Scissor ( ) [default]

### 10.90.2.2 LLGL::Scissor::Scissor ( const Scissor & ) [default]

### 10.90.2.3 LLGL::Scissor::Scissor ( std::int32_t *x*, std::int32_t *y*, std::int32_t *width*, std::int32_t *height* ) [inline]

Scissor constructor with parameters for all attributes.

### 10.90.2.4 LLGL::Scissor::Scissor ( const Offset2D & *offset*, const Extent2D & *extent* ) [inline]

Scissor constructor with offset and extent parameters.

## 10.90.3 Member Data Documentation

### 10.90.3.1 std::int32_t LLGL::Scissor::height = 0

Right-bottom height.

#### 10.90.3.2 std::int32_t LLGL::Scissor::width = 0

Right-bottom width.

#### 10.90.3.3 std::int32_t LLGL::Scissor::x = 0

Left-top X coordinate.

#### 10.90.3.4 std::int32_t LLGL::Scissor::y = 0

Left-top Y coordinate.

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

- GraphicsPipelineFlags.h

## 10.91 LLGL::Shader Class Reference

Shader interface.

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

Inheritance diagram for LLGL::Shader:

### Public Member Functions

- virtual bool HasErrors () const =0

  *Returns true if this shader has any errors. Otherwise, the compilation was successful.*
- virtual std::string GetReport () const =0

  *Returns the report message after the shader compilation or an empty string if there is no report.*
- long GetStageFlags () const

  *Returns the shader stage bitmask for this shader object.*
- ShaderType GetType () const

  *Returns the type of this shader.*

### Protected Member Functions

- Shader (const ShaderType type)

## 10.91.1 Detailed Description

Shader interface.

**See also**

RenderSystem::CreateShader

## 10.91.2 Constructor & Destructor Documentation

### 10.91.2.1 LLGL::Shader::Shader ( const ShaderType *type* ) [protected]

## 10.91.3 Member Function Documentation

### 10.91.3.1 virtual std::string LLGL::Shader::GetReport ( ) const [pure virtual]

Returns the report message after the shader compilation or an empty string if there is no report.

**Todo** Change return value to std::unique_ptr<LLGL::Blob>

**See also**

ShaderProgram::GetReport

### 10.91.3.2 long LLGL::Shader::GetStageFlags ( ) const

Returns the shader stage bitmask for this shader object.

**See also**

StageFlags

### 10.91.3.3 ShaderType LLGL::Shader::GetType ( ) const [inline]

Returns the type of this shader.

#### 10.91.3.4 virtual bool LLGL::Shader::HasErrors ( ) const [pure virtual]

Returns true if this shader has any errors. Otherwise, the compilation was successful.

**Remarks**

If the compilation failed, this shader can not be used for a graphics or compute pipeline. However, the details about the failure can be queried by the GetReport function.

**See also**

GetReport

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

- Shader.h

## 10.92 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) }

### 10.92.1 Detailed Description

Shader compilation flags enumeration.

**Note**

Only supported with: Direct3D 11, Direct3D 12.

### 10.92.2 Member Enumeration Documentation

#### 10.92.2.1 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

## 10.93 LLGL::ShaderDescriptor Struct Reference

Shader source and binary code descriptor structure.

```
#include <ShaderFlags.h>
```

### Classes

- struct StreamOutput

  *Additional descriptor for stream outputs.*

### Public Member Functions

- ShaderDescriptor ()=default
- ShaderDescriptor (const ShaderDescriptor &)=default
- ShaderDescriptor & operator= (const ShaderDescriptor &)=default
- ShaderDescriptor (const ShaderType type, const char *source)

  *Constructor to initialize the shader descriptor with a source filename.*
- ShaderDescriptor (const ShaderType type, const char *source, const char *entryPoint, const char *profile, long flags=0)

  *Constructor to initialize the shader descriptor with a source filename, entry point, profile, and optional flags.*

### Public Attributes

- ShaderType type = ShaderType::Undefined

  *Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like. By default ShaderType::Undefined.*
- const char * source = nullptr

  *Pointer to the shader source. This is either a null terminated string or a raw byte buffer (depending on the 'sourceType' member).*
- std::size_t sourceSize = 0

  *Specifies the size of the shader source (excluding the null terminator).*
- ShaderSourceType sourceType = ShaderSourceType::CodeFile

  *Specifies the type of the shader source. By default ShaderSourceType::CodeFile.*
- const char * entryPoint = nullptr

  *Shader entry point (shader main function). If this is null, the empty string is used. By default null.*
- const char * profile = nullptr

  *Shader target profile. If this is null, the empty string is used. By default null.*
- const ShaderMacro * defines = nullptr

  *Optional array of macro definitions. By default null.*
- long flags = 0

  *Optional compilation flags. By default 0.*
- StreamOutput streamOutput

  *Optional stream output descriptor for a geometry shader (or a vertex shader when used with OpenGL).*

### 10.93.1 Detailed Description

Shader source and binary code descriptor structure.

**See also**

RenderSystem::CreateShader

### 10.93.2 Constructor & Destructor Documentation

#### 10.93.2.1 LLGL::ShaderDescriptor::ShaderDescriptor ( ) [default]

#### 10.93.2.2 LLGL::ShaderDescriptor::ShaderDescriptor ( const ShaderDescriptor & ) [default]

#### 10.93.2.3 LLGL::ShaderDescriptor::ShaderDescriptor ( const ShaderType *type*, const char * *source* ) [inline]

Constructor to initialize the shader descriptor with a source filename.

#### 10.93.2.4 LLGL::ShaderDescriptor::ShaderDescriptor ( const ShaderType *type*, const char * *source*, const char * *entryPoint*, const char * *profile*, long *flags* = 0 ) [inline]

Constructor to initialize the shader descriptor with a source filename, entry point, profile, and optional flags.

### 10.93.3 Member Function Documentation

#### 10.93.3.1 ShaderDescriptor& LLGL::ShaderDescriptor::operator= ( const ShaderDescriptor & ) [default]

### 10.93.4 Member Data Documentation

#### 10.93.4.1 const ShaderMacro* LLGL::ShaderDescriptor::defines = nullptr

Optional array of macro definitions. By default null.

**Remarks**

This must either be null or a null-terminated array of ShaderMacro entries. For those shader compilers that provide a mechanism to add external macro definitions, this can be used to generate multiple shader permutations.
Here is a brief example how to use:

```
const LLGL::ShaderMacro myDefines[] = {
    { "ENABLE_SHADER_PASS_FOO", "1" }, // first macro
    { "ENABLE_SHADER_PASS_BAR", "0" }, // second macro
    { nullptr, nullptr },              // terminate array
};
LLGL::ShaderDescriptor myShaderDesc;
myShaderDesc.defines = myDefines;
```

**Note**

Only supported with: HLSL, Metal.

#### 10.93.4.2 const char* LLGL::ShaderDescriptor::entryPoint = nullptr

Shader entry point (shader main function). If this is null, the empty string is used. By default null.

**Note**

Only supported with: HLSL, SPIR-V, Metal.

#### 10.93.4.3 long LLGL::ShaderDescriptor::flags = 0

Optional compilation flags. By default 0.

**Remarks**

This can be a bitwise OR combination of the ShaderCompileFlags enumeration entries.

**Note**

Only supported with: HLSL.

**See also**

ShaderCompileFlags

#### 10.93.4.4 const char* LLGL::ShaderDescriptor::profile = nullptr

Shader target profile. If this is null, the empty string is used. By default null.

**Remarks**

This is renderer API dependent and is forwarded to the respective shader compiler.
Here are a few examples:

- For HLSL: `"vs_5_0"` specifies vertex shader model 5.0.
- For Metal: `"2.1"` specifies shader version 2.1.

**Note**

Only supported with: HLSL, Metal.

**See also**

https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.↩
85).aspx

#### 10.93.4.5 const char* LLGL::ShaderDescriptor::source = nullptr

Pointer to the shader source. This is either a null terminated string or a raw byte buffer (depending on the 'source↩ Type' member).

**Remarks**

This must not be null when passed to the RenderSystem::CreateShader function. If this is raw byte buffer rather than a null terminated string, the 'sourceSize' member must not be zero!

**See also**

sourceSize
sourceType

#### 10.93.4.6 std::size_t LLGL::ShaderDescriptor::sourceSize = 0

Specifies the size of the shader source (excluding the null terminator).

**Remarks**

If this is zero, the 'source' member is expected to point to a null terminated string and the size is automatically determined. For the binrary buffer source type (i.e. ShaderSourceType::BinaryBuffer), this must not be zero!

**See also**

source

#### 10.93.4.7 ShaderSourceType LLGL::ShaderDescriptor::sourceType = ShaderSourceType::CodeFile

Specifies the type of the shader source. By default ShaderSourceType::CodeFile.

**Remarks**

With the filename source types (i.e. ShaderSourceType::CodeFile and ShaderSourceType::BinaryFile), the shader source or binary code will be loaded from file using the standard C++ file streams (i.e. std::ifstream). Only the binary buffer source type (i.e. ShaderSourceType::BinaryBuffer) does not require a null terminator for the 'source' pointer.

**See also**

ShaderSourceType
source

#### 10.93.4.8 StreamOutput LLGL::ShaderDescriptor::streamOutput

Optional stream output descriptor for a geometry shader (or a vertex shader when used with OpenGL).

#### 10.93.4.9 ShaderType LLGL::ShaderDescriptor::type = ShaderType::Undefined

Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like. By default ShaderType::Undefined.

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

- ShaderFlags.h

## 10.94 LLGL::ShaderMacro Struct Reference

Shader macro structure with name and optional body definition.

```
#include <ShaderFlags.h>
```

### Public Member Functions

- ShaderMacro ()=default
- ShaderMacro (const ShaderMacro &)=default
- ShaderMacro & operator= (const ShaderMacro &)=default
- ShaderMacro (const char *name, const char *definition=nullptr)

  *Constructor to initialize the shader macro with a name and an optional body definition.*

### Public Attributes

- const char * name = nullptr

  *Specifies the name of the macro. This must not be null.*
- const char * definition = nullptr

  *Specifies the macro definition. If this is null, the macro has no body definition.*

### 10.94.1 Detailed Description

Shader macro structure with name and optional body definition.

**See also**

ShaderDescriptor::defines

### 10.94.2 Constructor & Destructor Documentation

#### 10.94.2.1 LLGL::ShaderMacro::ShaderMacro ( ) [default]

#### 10.94.2.2 LLGL::ShaderMacro::ShaderMacro ( const ShaderMacro & ) [default]

#### 10.94.2.3 LLGL::ShaderMacro::ShaderMacro ( const char * *name*, const char * *definition* = nullptr ) [inline]

Constructor to initialize the shader macro with a name and an optional body definition.

### 10.94.3 Member Function Documentation

#### 10.94.3.1 ShaderMacro& LLGL::ShaderMacro::operator= ( const ShaderMacro & ) [default]

### 10.94.4 Member Data Documentation

#### 10.94.4.1 const char* LLGL::ShaderMacro::definition = nullptr

Specifies the macro definition. If this is null, the macro has no body definition.

#### 10.94.4.2 const char* LLGL::ShaderMacro::name = nullptr

Specifies the name of the macro. This must not be null.

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

- ShaderFlags.h

## 10.95 LLGL::ShaderProgram Class Reference

Shader program interface.

```
#include <ShaderProgram.h>
```

Inheritance diagram for LLGL::ShaderProgram:

### Public Member Functions

- virtual bool HasErrors () const =0

  *Returns true if this shader program has any errors. Otherwise, the linking was successful.*
- virtual std::string GetReport () const =0

  *Returns the report message after the shader linkage or an empty string if there is no report.*
- virtual bool Reflect (ShaderReflection &reflection) const =0

  *Returns a reflection of the shader pipeline layout with all required shader resources.*
- virtual UniformLocation FindUniformLocation (const char *name) const =0

  *Returns the location of a single shader uniform by its name.*
- virtual bool SetWorkGroupSize (const Extent3D &workGroupSize)=0

  *Sets the work group size of a compute shader, i.e. the number of threads per thread-group. By default (1, 1, 1).*
- virtual bool GetWorkGroupSize (Extent3D &workGroupSize) const =0

  *Retrieves the work group size of a compute shader, i.e. the number of threads per thread-group.*

### Protected Types

- enum LinkError {
  LinkError::NoError, LinkError::InvalidComposition, LinkError::InvalidByteCode, LinkError::TooMany↩ Attachments,
  LinkError::IncompleteAttachments }

  *Linker error codes for internal error checking.*

### Static Protected Member Functions

- static bool ValidateShaderComposition (Shader ∗const ∗shaders, std::size_t numShaders)

  *Validates the composition of the specified shader attachments.*
- static void ClearShaderReflection (ShaderReflection &reflection)

  *Clears all members in the specified shader reflection.*
- static void FinalizeShaderReflection (ShaderReflection &reflection)

  *Sorts the resource views of the specified shader reflection descriptor as described in the Reflect function.*
- static const char ∗ LinkErrorToString (const LinkError errorCode)

  *Returns a string representation for the specified shader linker error, or null if the no error is entered (i.e. LinkError::↩ NoError).*

### Additional Inherited Members

## 10.95.1 Detailed Description

Shader program interface.

#### Remarks

A shader program combines multiple instances of the Shader class to be used in a complete shader pipeline.

#### See also

RenderSystem::CreateShaderProgram

## 10.95.2 Member Enumeration Documentation

### 10.95.2.1 enum LLGL::ShaderProgram::LinkError [strong],[protected]

Linker error codes for internal error checking.

**Todo** Turn this into a global common enumeration for error codes.

#### Enumerator

***NoError***

***InvalidComposition***

***InvalidByteCode***

***TooManyAttachments***

***IncompleteAttachments***

### 10.95.3 Member Function Documentation

#### 10.95.3.1 static void LLGL::ShaderProgram::ClearShaderReflection ( ShaderReflection & *reflection* ) [static], [protected]

Clears all members in the specified shader reflection.

**Remarks**

Use this to start reflecting a shader program as shown in the following example:

```
bool MyShaderProgram::Reflect(ShaderReflection& reflection)
{
    ShaderProgram::ClearShaderReflection(reflection);
    auto succeded = // reflection code here ...
    FinalizeShaderReflection(reflection);
    return succeded;
}
```

**See also**

Reflect
FinalizeShaderReflection

#### 10.95.3.2 static void LLGL::ShaderProgram::FinalizeShaderReflection ( ShaderReflection & *reflection* ) [static], [protected]

Sorts the resource views of the specified shader reflection descriptor as described in the Reflect function.

**See also**

Reflect
ClearShaderReflection

#### 10.95.3.3 virtual UniformLocation LLGL::ShaderProgram::FindUniformLocation ( const char * *name* ) const [pure virtual]

Returns the location of a single shader uniform by its name.

**Returns**

Uniform location of the specified uniform, or -1 if there is no such uniform in the shader program.

**Remarks**

This is a helper function when only one or a few number of uniform locations are meant to be determined. If more uniforms are involved, use the Reflect function.

**See also**

Reflect

**Note**

Only supported with: OpenGL, Vulkan, Direct3D 12.

#### 10.95.3.4 virtual std::string LLGL::ShaderProgram::GetReport ( ) const [pure virtual]

Returns the report message after the shader linkage or an empty string if there is no report.

**See also**

Shader::GetReport

#### 10.95.3.5 virtual bool LLGL::ShaderProgram::GetWorkGroupSize ( Extent3D & *workGroupSize* ) const [pure virtual]

Retrieves the work group size of a compute shader, i.e. the number of threads per thread-group.

**Parameters**

| | | |
|---|---|---|
| out | *workGroupSize* | Specifies the number of threads per thread-group in X, Y, and Z direction. This output parameter is not modified, if the function returns false. |

**Returns**

True, if the work group size could be determined. Otherwise, the rendering API does not support shader reflection to query the work group size, or the shader program does not contain a compute shader.

#### 10.95.3.6 virtual bool LLGL::ShaderProgram::HasErrors ( ) const [pure virtual]

Returns true if this shader program has any errors. Otherwise, the linking was successful.

**Remarks**

If the linking failed, this shader program can not be used for a graphics or compute pipeline. However, the details about the failure can be queried by the GetReport function.

**See also**

GetReport

#### 10.95.3.7 static const char* LLGL::ShaderProgram::LinkErrorToString ( const LinkError *errorCode* ) [static], [protected]

Returns a string representation for the specified shader linker error, or null if the no error is entered (i.e. LinkError↩ ::NoError).

#### 10.95.3.8 virtual bool LLGL::ShaderProgram::Reflect ( ShaderReflection & *reflection* ) const [pure virtual]

Returns a reflection of the shader pipeline layout with all required shader resources.

**Parameters**

| | | |
|---|---|---|
| out | *reflection* | Specifies the output shader reflection. If the function returns `false`, the content of this parameter is **undefined**! |

**Remarks**

The list of resources in the reflection output is always sorted by the following attributes (lower number means higher priority for sorting):

1. Resource type in ascending order (see BindingDescriptor::type).
2. Binding flags in ascending order (see BindingDescriptor::bindFlags).
3. Binding slot in ascending order (see BindingDescriptor::slot).

Here is an example of such a sorted list (pseudocode):

```
resources[0] = { type: ResourceType::Buffer,  bindFlags:
      BindFlags::ConstantBuffer, slot: 0 }
resources[1] = { type: ResourceType::Buffer,  bindFlags:
      BindFlags::ConstantBuffer, slot: 2 }
resources[2] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 0 }
resources[3] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 1 }
resources[4] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 2 }
resources[5] = { type: ResourceType::Sampler, bindFlags: 0,
      slot: 2 }
```

The `instanceDivisor` and `offset` members of the vertex attributes are ignored by this function.

**Returns**

True, if the reflection was successful. Otherwise, the shader reflection failed and the content of the output parameter `reflection` is undefined.

**See also**

ShaderReflection::resources
ShaderReflection::vertexAttributes
VertexAttribute::instanceDivisor
VertexAttribute::offset

#### 10.95.3.9 virtual bool LLGL::ShaderProgram::SetWorkGroupSize ( const Extent3D & *workGroupSize* ) [pure virtual]

Sets the work group size of a compute shader, i.e. the number of threads per thread-group. By default (1, 1, 1).

**Parameters**

| | | |
|---|---|---|
| in | *workGroupSize* | Specifies the number of threads per thread-group in X, Y, and Z direction. Each component must be greater than zero. |

**Returns**

True, if the work group size can be dynamically set and the values are valid. Otherwise, the work group size must be specified within the shader code or the values are invalid. If the return value is false, the function call has no effect.

**Remarks**

Only the Metal backend supports dispatch compute kernels with dynamic work group sizes. For all other renderers, the work group size must be specified within the shader code:

- For GLSL: `layout(local_size_x = X, local_size_y = Y, local_size_z = Z)`
- For HLSL: `[numthreads(X, Y, Z)]`

**Note**

Only supported with: Metal.

#### 10.95.3.10 static bool LLGL::ShaderProgram::ValidateShaderComposition ( Shader *const * *shaders*, std::size_t *numShaders* ) [static],[protected]

Validates the composition of the specified shader attachments.

**Parameters**

| | | |
|---|---|---|
| in | *shaders* | Array of Shader objects that belong to this shader program. Null pointers within the array are ignored. |
| in | *numShaders* | Specifies the number of entries in the array 'shaders'. This must not be larger than the number of entries in the 'shaders' array. |

**Returns**

True if the shader composition is valid, otherwise false.

**Remarks**

For example, a composition of a compute shader and a fragment shader is invalid, but a composition of a vertex shader and a fragment shader is valid.

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

- ShaderProgram.h

## 10.96 LLGL::ShaderProgramDescriptor Struct Reference

Descriptor structure for shader programs.

```
#include <ShaderProgramFlags.h>
```

### Public Attributes

- std::vector< VertexFormat > vertexFormats

  *Vertex format list. This may also be empty, if the vertex shader has no input attributes or only a compute shader is specified.*
- Shader * vertexShader = nullptr

  *Specifies the vertex shader.*
- Shader * tessControlShader = nullptr

  *Specifies the tessellation-control shader (also referred to as "Hull Shader").*
- Shader * tessEvaluationShader = nullptr

  *Specifies the tessellation-evaluation shader (also referred to as "Domain Shader").*
- Shader * geometryShader = nullptr

  *Specifies an optional geometry shader.*
- Shader * fragmentShader = nullptr

  *Specifies an optional fragment shader (also referred to as "Pixel Shader").*
- Shader * computeShader = nullptr

  *Specifies the compute shader.*

## 10.96.1 Detailed Description

Descriptor structure for shader programs.

**See also**

RenderSystem::CreateShaderProgram
RenderSystem::CreateShader

## 10.96.2 Member Data Documentation

### 10.96.2.1 Shader* LLGL::ShaderProgramDescriptor::computeShader = nullptr

Specifies the compute shader.

**Remarks**

This shader cannot be used in conjunction with any other shaders.

### 10.96.2.2 Shader* LLGL::ShaderProgramDescriptor::fragmentShader = nullptr

Specifies an optional fragment shader (also referred to as "Pixel Shader").

**Remarks**

If no fragment shader is specified, generated fragments are discarded by the output merger and only the stream-output functionality is used by either the vertex or geometry shader.

#### 10.96.2.3 Shader* LLGL::ShaderProgramDescriptor::geometryShader = nullptr

Specifies an optional geometry shader.

**Remarks**

This shader may also have a stream output.

**See also**

ShaderDescriptor::streamOutput

#### 10.96.2.4 Shader* LLGL::ShaderProgramDescriptor::tessControlShader = nullptr

Specifies the tessellation-control shader (also referred to as "Hull Shader").

**Remarks**

If this is used, the counter part must also be specified (i.e. `tessEvaluationShader`).

**See also**

tessEvaluationShader

#### 10.96.2.5 Shader* LLGL::ShaderProgramDescriptor::tessEvaluationShader = nullptr

Specifies the tessellation-evaluation shader (also referred to as "Domain Shader").

**Remarks**

If this is used, the counter part must also be specified (i.e. `tessControlShader`).

**See also**

tessControlShader

#### 10.96.2.6 std::vector<VertexFormat> LLGL::ShaderProgramDescriptor::vertexFormats

Vertex format list. This may also be empty, if the vertex shader has no input attributes or only a compute shader is specified.

**See also**

VertexFormat

#### 10.96.2.7 Shader* LLGL::ShaderProgramDescriptor::vertexShader = nullptr

Specifies the vertex shader.

**Remarks**

Each graphics shader program must have at least a vertex shader. For a compute shader program, only a compute shader must be specified. With OpenGL, this shader may also have a stream output.

**See also**

ShaderDescriptor::streamOutput

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

- ShaderProgramFlags.h

## 10.97 LLGL::ShaderReflection Struct Reference

Shader reflection structure.

```
#include <ShaderProgramFlags.h>
```

### Public Attributes

- std::vector< VertexAttribute > vertexAttributes

  *List of all vertex input attributes.*
- std::vector< StreamOutputAttribute > streamOutputAttributes

  *List of all stream-output attributes.*
- std::vector< ShaderResource > resources

  *List of all shader reflection resource views.*
- std::vector< ShaderUniform > uniforms

  *List of all uniforms (a.k.a. shader constants).*

### 10.97.1 Detailed Description

Shader reflection structure.

**Remarks**

Contains all information of resources and attributes that can be queried from a shader program. This is not a "descriptor", because it is only used as output from an interface rather than a description to create something.

**See also**

ShaderProgram::Reflect

### 10.97.2 Member Data Documentation

#### 10.97.2.1 std::vector<ShaderResource> LLGL::ShaderReflection::resources

List of all shader reflection resource views.

#### 10.97.2.2 std::vector<StreamOutputAttribute> LLGL::ShaderReflection::streamOutputAttributes

List of all stream-output attributes.

#### 10.97.2.3 std::vector<ShaderUniform> LLGL::ShaderReflection::uniforms

List of all uniforms (a.k.a. shader constants).

**Note**

Only supported with: OpenGL, Vulkan.

#### 10.97.2.4 std::vector<VertexAttribute> LLGL::ShaderReflection::vertexAttributes

List of all vertex input attributes.

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

- ShaderProgramFlags.h

## 10.98 LLGL::ShaderResource Struct Reference

Shader reflection resource structure.

```
#include <ShaderProgramFlags.h>
```

### Public Attributes

- BindingDescriptor binding

  *Binding descriptor with resource name, binding slot, flags, and array size.*
- std::uint32_t constantBufferSize = 0

  *Specifies the size (in bytes) for a constant buffer resource.*
- StorageBufferType storageBufferType = StorageBufferType::Undefined

  *Specifies the sub-type of a storage buffer resource.*

### 10.98.1 Detailed Description

Shader reflection resource structure.

**See also**

ShaderReflection::resources
BindingDescriptor

### 10.98.2 Member Data Documentation

#### 10.98.2.1 BindingDescriptor LLGL::ShaderResource::binding

Binding descriptor with resource name, binding slot, flags, and array size.

**Remarks**

Although the `name` attribute in the BindingDescriptor structure is optional for pipeline layouts, the shader reflection always queries this attribute as well.

#### 10.98.2.2 std::uint32_t LLGL::ShaderResource::constantBufferSize = 0

Specifies the size (in bytes) for a constant buffer resource.

**Remarks**

Additional attribute exclusively used for constant buffer resources. For all other resources, i.e. when 'type' is not equal to 'ResourceType::ConstantBuffer', this attribute is zero.

**See also**

ResourceType::ConstantBuffer

#### 10.98.2.3 StorageBufferType LLGL::ShaderResource::storageBufferType = StorageBufferType::Undefined

Specifies the sub-type of a storage buffer resource.

**Remarks**

Additional attribute exclusively used for storage buffer resources.

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

- ShaderProgramFlags.h

## 10.99 LLGL::ShaderUniform Struct Reference

Shader reflection uniform structure.

```
#include <ShaderProgramFlags.h>
```

### Public Attributes

- std::string name

  *Name of the uniform inside the shader.*
- UniformType type = UniformType::Undefined

  *Data type of the uniform. By default UniformType::Undefined.*
- UniformLocation location = 0

  *Internal location of the uniform within a shader program.*
- std::uint32_t size = 0

  *Array size of the uniform.*

### 10.99.1 Detailed Description

Shader reflection uniform structure.

**See also**

ShaderReflection::uniforms

### 10.99.2 Member Data Documentation

#### 10.99.2.1 UniformLocation LLGL::ShaderUniform::location = 0

Internal location of the uniform within a shader program.

#### 10.99.2.2 std::string LLGL::ShaderUniform::name

Name of the uniform inside the shader.

#### 10.99.2.3 std::uint32_t LLGL::ShaderUniform::size = 0

Array size of the uniform.

#### 10.99.2.4 UniformType LLGL::ShaderUniform::type = UniformType::Undefined

Data type of the uniform. By default UniformType::Undefined.

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

- ShaderProgramFlags.h

## 10.100 LLGL::SrcImageDescriptor Struct Reference

Descriptor structure for an image that is used as source for reading the image data.

```
#include <ImageFlags.h>
```

### Public Member Functions

- SrcImageDescriptor ()=default
- SrcImageDescriptor (const SrcImageDescriptor &)=default
- SrcImageDescriptor (ImageFormat format, DataType dataType, const void *data, std::size_t dataSize)

  *Constructor to initialize all attributes.*

### 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. By default DataType::UInt8.*
- const void * data = nullptr

  *Pointer to the read-only image data.*
- std::size_t dataSize = 0

  *Specifies the size (in bytes) of the image data. This is primarily used for compressed images and serves for robustness.*

### 10.100.1 Detailed Description

Descriptor structure for an image that is used as source for reading the image data.

**Remarks**

This kind of 'Image' is mainly used to fill a MIP-map within a hardware texture by reading from a source image. The counterpart for reading a MIP-map from a hardware texture by writing to a destination image is the DstImageDescriptor structure.

**See also**

DstImageDescriptor
ConvertImageBuffer
RenderSystem::CreateTexture
RenderSystem::WriteTexture

### 10.100.2 Constructor & Destructor Documentation

#### 10.100.2.1 LLGL::SrcImageDescriptor::SrcImageDescriptor ( ) [default]

#### 10.100.2.2 LLGL::SrcImageDescriptor::SrcImageDescriptor ( const SrcImageDescriptor & ) [default]

#### 10.100.2.3 LLGL::SrcImageDescriptor::SrcImageDescriptor ( ImageFormat *format*, DataType *dataType*, const void * *data*, std::size_t *dataSize* ) [inline]

Constructor to initialize all attributes.

### 10.100.3 Member Data Documentation

#### 10.100.3.1 const void* LLGL::SrcImageDescriptor::data = nullptr

Pointer to the read-only image data.

#### 10.100.3.2 std::size_t LLGL::SrcImageDescriptor::dataSize = 0

Specifies the size (in bytes) of the image data. This is primarily used for compressed images and serves for robustness.

#### 10.100.3.3 DataType LLGL::SrcImageDescriptor::dataType = DataType::UInt8

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

#### 10.100.3.4 ImageFormat LLGL::SrcImageDescriptor::format = ImageFormat::RGBA

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

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

- ImageFlags.h

## 10.101 LLGL::StageFlags Struct Reference

Shader stage flags enumeration.

```
#include <ShaderFlags.h>
```

### Public Types

- enum {
  VertexStage = (1 << 0), TessControlStage = (1 << 1), TessEvaluationStage = (1 << 2), GeometryStage = (1 << 3),
  FragmentStage = (1 << 4), ComputeStage = (1 << 5), AllTessStages = (TessControlStage | Tess↩EvaluationStage), AllGraphicsStages = (VertexStage | AllTessStages | GeometryStage | FragmentStage),
  AllStages = (AllGraphicsStages | ComputeStage) }

### 10.101.1 Detailed Description

Shader stage flags enumeration.

**Remarks**

Specifies which shader stages are affected by a state change, e.g. to which shader stages a constant buffer is bound.

**See also**

BindingDescriptor::stageFlags

### 10.101.2 Member Enumeration Documentation

#### 10.101.2.1 anonymous enum

**Enumerator**

***VertexStage*** Specifies the vertex shader stage.

***TessControlStage*** Specifies the tessellation-control shader stage (also referred to as "Hull Shader").

***TessEvaluationStage*** Specifies the tessellation-evaluation shader stage (also referred to as "Domain Shader").

***GeometryStage*** Specifies the geometry shader stage.

***FragmentStage*** Specifies the fragment shader stage (also referred to as "Pixel Shader").

***ComputeStage*** Specifies the compute shader stage.

***AllTessStages*** Specifies all tessellation stages, i.e. tessellation-control-, tessellation-evaluation shader stages.

***AllGraphicsStages*** Specifies all graphics pipeline shader stages, i.e. vertex-, tessellation-, geometry-, and fragment shader stages.

***AllStages*** Specifies all shader stages.

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

- ShaderFlags.h

## 10.102 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. By default disabled.*
- StencilFaceDescriptor front

  *Specifies the front face settings for the stencil test.*
- StencilFaceDescriptor back

  *Specifies the back face settings for the stencil test.*

### 10.102.1 Detailed Description

Stencil state descriptor structure.

**See also**

GraphicsPipelineDescriptor::stencil

### 10.102.2 Member Data Documentation

#### 10.102.2.1 StencilFaceDescriptor LLGL::StencilDescriptor::back

Specifies the back face settings for the stencil test.

#### 10.102.2.2 StencilFaceDescriptor LLGL::StencilDescriptor::front

Specifies the front face settings for the stencil test.

**Note**

For Direct3D 11 and Direct3D 12, the members `readMask`, `writeMask`, and `reference` are only supported for the front face.

**See also**

StencilFaceDescriptor::readMask
StencilFaceDescriptor::writeMask
StencilFaceDescriptor::reference

#### 10.102.2.3 bool LLGL::StencilDescriptor::testEnabled = false

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

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

- GraphicsPipelineFlags.h

## 10.103 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. By default StencilOp::Keep.*
- StencilOp depthFailOp = StencilOp::Keep

  *Specifies the operation to take when the stencil test passes but the depth test fails. By default StencilOp::Keep.*
- StencilOp depthPassOp = StencilOp::Keep

  *Specifies the operation to take when both the stencil test and the depth test pass. By default StencilOp::Keep.*
- CompareOp compareOp = CompareOp::Less

  *Specifies the stencil compare operation. By default CompareOp::Less.*
- std::uint32_t readMask = ~0u

  *Specifies the portion of the depth-stencil buffer for reading stencil data. By default `0xFFFFFFFF`.*
- std::uint32_t writeMask = ~0u

  *Specifies the portion of the depth-stencil buffer for writing stencil data. By default `0xFFFFFFFF`.*
- std::uint32_t reference = 0u

  *Specifies the stencil reference value. By default 0.*

### 10.103.1 Detailed Description

Stencil face descriptor structure.

**See also**

StencilDescriptor::front
StencilDescriptor::back

### 10.103.2 Member Data Documentation

#### 10.103.2.1 CompareOp LLGL::StencilFaceDescriptor::compareOp = CompareOp::Less

Specifies the stencil compare operation. By default CompareOp::Less.

#### 10.103.2.2 StencilOp LLGL::StencilFaceDescriptor::depthFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test passes but the depth test fails. By default StencilOp::Keep.

#### 10.103.2.3 StencilOp LLGL::StencilFaceDescriptor::depthPassOp = StencilOp::Keep

Specifies the operation to take when both the stencil test and the depth test pass. By default StencilOp::Keep.

#### 10.103.2.4 std::uint32_t LLGL::StencilFaceDescriptor::readMask = ~0u

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 (i.e. `readMask & 0xFF`) of the read mask value of the front face will be used.

**See also**

StencilDescriptor::front

#### 10.103.2.5 std::uint32_t LLGL::StencilFaceDescriptor::reference = 0u

Specifies the stencil reference value. By default 0.

**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

#### 10.103.2.6 StencilOp LLGL::StencilFaceDescriptor::stencilFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test fails. By default StencilOp::Keep.

#### 10.103.2.7 std::uint32_t LLGL::StencilFaceDescriptor::writeMask = ~0u

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 (i.e. `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

## 10.104 LLGL::BufferDescriptor::StorageBuffer Struct Reference

Storage buffer specific descriptor structure.

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

### Public Attributes

- StorageBufferType storageType = StorageBufferType::Undefined

  *Specifies the storage buffer type. By defalut StorageBufferType::Undefined.*
- Format format = Format::Undefined

  *Specifies the vector format of a typed buffer. By default Format::Undefined.*
- std::uint32_t stride = 0

  *Specifies the stride (in bytes) of each element in a storage buffer. By default 0.*

### 10.104.1 Detailed Description

Storage buffer specific descriptor structure.

**Todo** Move this into a BufferViewDescriptor.

### 10.104.2 Member Data Documentation

#### 10.104.2.1 Format LLGL::BufferDescriptor::StorageBuffer::format = Format::Undefined

Specifies the vector format of a typed buffer. By default Format::Undefined.

**Remarks**

This is only used if the storage type is either StorageBufferType::Buffer or StorageBufferType::RWBuffer.

**See also**

IsTypedBuffer

#### 10.104.2.2 StorageBufferType LLGL::BufferDescriptor::StorageBuffer::storageType = StorageBufferType::Undefined

Specifies the storage buffer type. By defalut StorageBufferType::Undefined.

**Remarks**

In OpenGL there are only generic storage buffers (or rather "Shader Storage Buffer Objects"). However, a valid type should always be specified when a storage buffer is created.

#### 10.104.2.3 std::uint32_t LLGL::BufferDescriptor::StorageBuffer::stride = 0

Specifies the stride (in bytes) of each element in a storage buffer. By default 0.

**Remarks**

If the buffer has the BindFlags::Sampled or BindFlags::Storage flag, then `stride` must not be zero.

**See also**

BufferDescriptor::bindFlags

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

- BufferFlags.h

## 10.105 LLGL::ShaderDescriptor::StreamOutput Struct Reference

Additional descriptor for stream outputs.

```
#include <ShaderFlags.h>
```

### Public Attributes

- StreamOutputFormat format

  *Stream-output buffer format.*

### 10.105.1 Detailed Description

Additional descriptor for stream outputs.

### 10.105.2 Member Data Documentation

#### 10.105.2.1 StreamOutputFormat LLGL::ShaderDescriptor::StreamOutput::format

Stream-output buffer format.

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

- ShaderFlags.h

## 10.106 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).*
- std::uint32_t stream = 0

  *Zero-based stream number. By default 0.*
- std::uint8_t startComponent = 0

  *Start vector component index, which is to be written. Must be 0, 1, 2, or 3. By default 0.*
- std::uint8_t components = 4

  *Number of vector components, which are to be written. Must be 1, 2, 3, or 4.*
- std::uint32_t semanticIndex = 0

  *Semantic index.*
- std::uint8_t outputSlot = 0

  *Stream-output buffer output slot.*

### 10.106.1 Detailed Description

Stream-output attribute structure.

### 10.106.2 Constructor & Destructor Documentation

#### 10.106.2.1 LLGL::StreamOutputAttribute::StreamOutputAttribute ( ) [default]

#### 10.106.2.2 LLGL::StreamOutputAttribute::StreamOutputAttribute ( const StreamOutputAttribute & ) [default]

### 10.106.3 Member Function Documentation

#### 10.106.3.1 StreamOutputAttribute& LLGL::StreamOutputAttribute::operator= ( const StreamOutputAttribute & ) [default]

### 10.106.4 Member Data Documentation

#### 10.106.4.1 std::uint8_t 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

#### 10.106.4.2 std::string LLGL::StreamOutputAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

#### 10.106.4.3 std::uint8_t LLGL::StreamOutputAttribute::outputSlot = 0

Stream-output buffer output slot.

**Remarks**

This is used when multiple stream-output buffers are used simultaneously.

#### 10.106.4.4 std::uint32_t LLGL::StreamOutputAttribute::semanticIndex = 0

Semantic index.

**Note**

Only supported with: Direct3D 11, Direct3D 12.

#### 10.106.4.5 std::uint8_t LLGL::StreamOutputAttribute::startComponent = 0

Start vector component index, which is to be written. Must be 0, 1, 2, or 3. By default 0.

#### 10.106.4.6 std::uint32_t LLGL::StreamOutputAttribute::stream = 0

Zero-based stream number. By default 0.

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

- StreamOutputAttribute.h

## 10.107 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.*

### 10.107.1 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.

### 10.107.2 Member Function Documentation

#### 10.107.2.1 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. |

#### 10.107.2.2 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.

### 10.107.3 Member Data Documentation

#### 10.107.3.1 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

## 10.108 LLGL::Surface Class Reference

The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms).

```
#include <Surface.h>
```

Inheritance diagram for LLGL::Surface:

### Public Member Functions

- virtual void GetNativeHandle (void *nativeHandle) const =0

  *Returns the native surface handle.*
- virtual Extent2D GetContentSize () const =0

  *Returns the size of the surface context (or rather the drawing area).*
- virtual bool AdaptForVideoMode (VideoModeDescriptor &videoModeDesc)=0

  *Adapts the surface to fits the needs for the specified video mode descriptor.*
- virtual void ResetPixelFormat ()=0

  *Resets the internal pixel format of the surface.*
- virtual bool ProcessEvents ()=0

  *Processes all events for this surface, i.e. input-, movement-, resize-, and other events.*

### Additional Inherited Members

## 10.108.1 Detailed Description

The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms).

#### Remarks

Surface provides the minimal required interface for a graphics rendering context, such as the access to the native handle, information about the content size (i.e. the client area size), and the ability to adapt for a new video mode or an updated pixel format. (which is required for multi-sampled framebuffers on a WGL context for instance).

#### See also

Window
Canvas

## 10.108.2 Member Function Documentation

### 10.108.2.1 virtual bool LLGL::Surface::AdaptForVideoMode ( VideoModeDescriptor & *videoModeDesc* ) [pure virtual]

Adapts the surface to fits the needs for the specified video mode descriptor.

#### Parameters

| | | |
|---|---|---|
| in,out | *videoModeDesc* | Specifies the input and output video mode descriptor. |

#### Returns

If the video mode descriptor has been accepted with no modifications and this surface has been updated then the return value is true. Otherwise the video mode descriptor has been modified to the value this surface supports and the return value is false.

Implemented in LLGL::Window, and LLGL::Canvas.

#### 10.108.2.2 virtual Extent2D LLGL::Surface::GetContentSize ( ) const [pure virtual]

Returns the size of the surface context (or rather the drawing area).

**Remarks**

For the Window interface this is equivalent of calling `Window::GetSize(true)` for instance.

**See also**

Window::GetSize

#### 10.108.2.3 virtual void LLGL::Surface::GetNativeHandle ( void * *nativeHandle* ) const [pure virtual]

Returns the native surface handle.

**Remarks**

This must be casted to a platform specific structure:

```
// Example for a custom Win32 window class
#include <LLGL/Platform/NativeHandle.h>
//...
void MyWindowClass::GetNativeHandle(void* nativeHandle) {
    auto handle = reinterpret_cast<LLGL::NativeHandle*>(nativeHandle);
    //handle->window = 'some HWND window handle';
}
```

#### 10.108.2.4 virtual bool LLGL::Surface::ProcessEvents ( ) [pure virtual]

Processes all events for this surface, i.e. input-, movement-, resize-, and other events.

**Remarks**

This function is only implemented by the Window and Canvas interfaces.

**See also**

Window::ProcessEvents
Canvas::ProcessEvents

Implemented in LLGL::Window, and LLGL::Canvas.

#### 10.108.2.5 virtual void LLGL::Surface::ResetPixelFormat ( ) [pure virtual]

Resets the internal pixel format of the surface.

**Remarks**

This function is mainly used by the OpenGL renderer on Win32 when a multi-sampled framebuffer is created.

**Note**

This may invalidate the native handle previously returned by `GetNativeHandle`.

**See also**

GetNativeHandle

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

- Surface.h

## 10.109 LLGL::Texture Class Reference

Texture interface.

```
#include <Texture.h>
```

Inheritance diagram for LLGL::Texture:

### Public Member Functions

- ResourceType GetResourceType () const override final

  *Returns ResourceType::Texture.*
- TextureType GetType () const

  *Returns the type of this texture.*
- virtual TextureDescriptor GetDesc () const =0

  *Queries a descriptor of this texture.*
- virtual Extent3D GetMipExtent (std::uint32_t mipLevel) const =0

  *Returns the texture extent for the specified MIP-level. This also includes the number of array layers.*

### Protected Member Functions

- Texture (const TextureType type)

### 10.109.1 Detailed Description

Texture interface.

**See also**

RenderSystem::CreateTexture
TextureDescriptor
TextureLocation
TextureRegion

## 10.109.2 Constructor & Destructor Documentation

### 10.109.2.1 LLGL::Texture::Texture ( const TextureType *type* ) [protected]

## 10.109.3 Member Function Documentation

### 10.109.3.1 virtual TextureDescriptor LLGL::Texture::GetDesc ( ) const [pure virtual]

Queries a descriptor of this texture.

**Remarks**

This function only queries the following attributes:

- `type`
- `format`
- `extent`
- `arrayLayers`
- `mipLevels`

All other attributes (such as `bindFlags`, `cpuAccessFlags`, `miscFlags` etc.) cannot be queried by this function. Those attributes are either set to zero (for flags) or the default value specified in TextureDescriptor is used.

**See also**

TextureDescriptor
Buffer::GetDesc

### 10.109.3.2 virtual Extent3D LLGL::Texture::GetMipExtent ( std::uint32_t *mipLevel* ) const [pure virtual]

Returns the texture extent for the specified MIP-level. This also includes the number of array layers.

For a 1D array texture, the number of array layers is stored in the height extent. For a 2D and cube array texture, the number of array layers is stored in the depth extent.

**Parameters**

| | | |
|---|---|---|
| in | *mipLevel* | Specifies the MIP-map level to query from. The first and largest MIP-map is level zero. If this level is greater than or equal to the maxmimum number of MIP-maps for this texture, the return value is undefined (i.e. depends on the render system). |

**Remarks**

For cube textures and cube array textures (i.e. texture type TextureType::TextureCube and TextureType::TextureCubeArray), the depth extent will be a multiple of 6. This is in contrast to the other handling of cube array layers, because this function determines the actual buffer extent of the hardware texture.

**See also**

TextureDescriptor::mipLevels
CommandBuffer::GenerateMips

#### 10.109.3.3 ResourceType LLGL::Texture::GetResourceType ( ) const [final], [override], [virtual]

Returns ResourceType::Texture.

Implements LLGL::Resource.

#### 10.109.3.4 TextureType LLGL::Texture::GetType ( ) const [inline]

Returns the type of this texture.

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

- Texture.h

## 10.110 LLGL::TextureDescriptor Struct Reference

Texture descriptor structure.

```
#include <TextureFlags.h>
```

### Public Attributes

- TextureType type = TextureType::Texture2D

  *Hardware texture type. By default TextureType::Texture2D.*
- long bindFlags = (BindFlags::Sampled | BindFlags::ColorAttachment)

  *These flags describe to which resource slots and render target attachments the texture can be bound. By default BindFlags::Sampled and BindFlags::ColorAttachment.*
- long cpuAccessFlags = 0

  *CPU read/write access flags. By default 0.*
- long miscFlags = (MiscFlags::FixedSamples | MiscFlags::GenerateMips)

  *Miscellaneous texture flags. By default MiscFlags::FixedSamples and MiscFlags::GenerateMips.*
- Format format = Format::RGBA8UNorm

  *Hardware texture format. By default Format::RGBA8UNorm.*
- Extent3D extent = { 1, 1, 1 }

  *Size of the texture (excluding the number of array layers). By default (1, 1, 1).*
- std::uint32_t arrayLayers = 1

  *Number of array layers. By default 1.*
- std::uint32_t mipLevels = 0

  *Number of MIP-map levels. By default 0.*
- std::uint32_t samples = 1

  *Number of samples per texel. By default 1.*
- ClearValue clearValue

  *Specifies a clear value to initialize the texture with, if no initial image data is provided.*

### 10.110.1 Detailed Description

Texture descriptor structure.

**Remarks**

Contains all information about type, format, and dimension to create a texture resource.

**See also**

RenderSystem::CreateTexture

### 10.110.2 Member Data Documentation

#### 10.110.2.1 std::uint32_t LLGL::TextureDescriptor::arrayLayers = 1

Number of array layers. By default 1.

**Remarks**

For array textures and cube textures (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCube, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray), this **must** be greater than or equal to 1.
For cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray), this **must** be a multiple of 6 and greater than zero (one array layer for each cube face).
For all other texture types, this **must** be 1.
The index offsets for each cube face are as follows:

- `X+` direction has index offset 0.
- `X-` direction has index offset 1.
- `Y+` direction has index offset 2.
- `Y-` direction has index offset 3.
- `Z+` direction has index offset 4.
- `Z-` direction has index offset 5.

**See also**

IsArrayTexture
IsCubeTexture
RenderingLimits::maxTextureArrayLayers

#### 10.110.2.2 long LLGL::TextureDescriptor::bindFlags = (BindFlags::Sampled | BindFlags::ColorAttachment)

These flags describe to which resource slots and render target attachments the texture can be bound. By default BindFlags::Sampled and BindFlags::ColorAttachment.

**Remarks**

When the texture will be bound as a color attachment to a render target for instance, the BindFlags::ColorAttachment flag is required.

**See also**

BindFlags

#### 10.110.2.3 ClearValue LLGL::TextureDescriptor::clearValue

Specifies a clear value to initialize the texture with, if no initial image data is provided.

**Remarks**

The initial texture data is only determined by this attribute if the `imageDesc` parameter of RenderSystem::CreateTexture is null and the MiscFlags::NoInitialData bit is **not** set in the `miscFlags` attribute. In either case, this value may be used by the renderer API as a hint which clear value the resource is optimized for (especially for Direct3D 12).

**See also**

RenderSystem::CreateTexture
TextureDescriptor::miscFlags

#### 10.110.2.4 long LLGL::TextureDescriptor::cpuAccessFlags = 0

CPU read/write access flags. By default 0.

**Remarks**

If this is 0 the texture cannot be mapped from GPU memory space into CPU memory space and vice versa.

**See also**

CPUAccessFlags
RenderSystem::MapTexture

**Todo** Not supported yet.

#### 10.110.2.5 Extent3D LLGL::TextureDescriptor::extent = { 1, 1, 1 }

Size of the texture (excluding the number of array layers). By default (1, 1, 1).

**Remarks**

The `height` component is only used for 2D, 3D, and Cube textures (i.e. TextureType::Texture2D, TextureType::Texture2DArray, TextureType::Texture3D, TextureType::TextureCube, TextureType::TextureCubeArray, TextureType::Texture2DMS, and TextureType::Texture2DMSArray).
The `depth` component is only used for 3D textures (i.e. TextureType::Texture3D).
The `width` and `height` components must be equal for cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray).

**See also**

IsCubeTexture

#### 10.110.2.6 Format LLGL::TextureDescriptor::format = Format::RGBA8UNorm

Hardware texture format. By default Format::RGBA8UNorm.

#### 10.110.2.7 std::uint32_t LLGL::TextureDescriptor::mipLevels = 0

Number of MIP-map levels. By default 0.

**Remarks**

If this is 0, the full MIP-chain will be generated. If this is 1, no MIP-mapping is used for this texture and it has only a single MIP-map level. This field is ignored for multi-sampled textures (i.e. TextureType::Texture2DMS, TextureType::Texture2DMSArray), since these texture types only have a single MIP-map level.

**See also**

NumMipLevels
CommandBuffer::GenerateMips

#### 10.110.2.8 long LLGL::TextureDescriptor::miscFlags = (MiscFlags::FixedSamples | MiscFlags::GenerateMips)

Miscellaneous texture flags. By default MiscFlags::FixedSamples and MiscFlags::GenerateMips.

**Remarks**

This can be used as a hint for the renderer how frequently the texture will be updated, or whether a multi-sampled texture has fixed sample locations.

**See also**

MiscFlags

#### 10.110.2.9 std::uint32_t LLGL::TextureDescriptor::samples = 1

Number of samples per texel. By default 1.

**Remarks**

This is only used for multi-sampled textures (i.e. TextureType::Texture2DMS and TextureType::Texture2DMSArray). The equivalent member for graphics pipeline states is MultiSamplingDescriptor::samples.

**See also**

IsMultiSampleTexture

#### 10.110.2.10 TextureType LLGL::TextureDescriptor::type = TextureType::Texture2D

Hardware texture type. By default TextureType::Texture2D.

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

- TextureFlags.h

## 10.111 LLGL::TextureLocation Struct Reference

Texture location structure: MIP-map level and offset.

```
#include <TextureFlags.h>
```

### Public Member Functions

- TextureLocation ()=default
- TextureLocation (const TextureLocation &)=default
- TextureLocation (const Offset3D &offset, std::uint32_t arrayLayer=0, std::uint32_t mipLevel=0)

  *Constructor to initialize all attributes.*

### Public Attributes

- Offset3D offset

  *Zero-based offset within the texture data.*
- std::uint32_t arrayLayer = 0

  *Zero-based array layer index.*
- std::uint32_t mipLevel = 0

  *MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.*

### 10.111.1 Detailed Description

Texture location structure: MIP-map level and offset.

#### Remarks

This is used to specifiy the source and destination location of a texture copy operation.

#### See also

CommandBuffer::CopyTexture
TextureRegion

## 10.111.2 Constructor & Destructor Documentation

### 10.111.2.1 LLGL::TextureLocation::TextureLocation ( ) [default]

### 10.111.2.2 LLGL::TextureLocation::TextureLocation ( const TextureLocation & ) [default]

### 10.111.2.3 LLGL::TextureLocation::TextureLocation ( const Offset3D & *offset*, std::uint32_t *arrayLayer* = 0, std::uint32_t *mipLevel* = 0 ) [inline]

Constructor to initialize all attributes.

## 10.111.3 Member Data Documentation

### 10.111.3.1 std::uint32_t LLGL::TextureLocation::arrayLayer = 0

Zero-based array layer index.

**Remarks**

Only used by array texture types (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray).
This field is ignored by all other texture types.

**See also**

TextureDescriptor::arrayLayers

### 10.111.3.2 std::uint32_t LLGL::TextureLocation::mipLevel = 0

MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.

**See also**

TextureDescriptor::mipLevels

### 10.111.3.3 Offset3D LLGL::TextureLocation::offset

Zero-based offset within the texture data.

**Remarks**

Any component of this field that is not meant for the respective texture type is ignored. All other components must be greater than or equal to zero.

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

- TextureFlags.h

## 10.112 LLGL::TextureRegion Struct Reference

Texture region structure: MIP-map level, offset, and extent.

```
#include <TextureFlags.h>
```

### Public Member Functions

- TextureRegion ()=default
- TextureRegion (const TextureRegion &)=default
- TextureRegion (const Offset3D &offset, const Extent3D &extent)

  *Constructor to initialize offset and extent only.*

### Public Attributes

- TextureSubresource subresource

  *Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.*
- Offset3D offset

  *Zero-based offset within the texture data.*
- Extent3D extent

  *Extent of the sub texture region.*

### 10.112.1 Detailed Description

Texture region structure: MIP-map level, offset, and extent.

#### Remarks

This is used to write (or partially write) the image data of a texture MIP-map level.

#### See also

RenderSystem::WriteTexture
TextureLocation

### 10.112.2 Constructor & Destructor Documentation

#### 10.112.2.1 LLGL::TextureRegion::TextureRegion ( ) [default]

#### 10.112.2.2 LLGL::TextureRegion::TextureRegion ( const TextureRegion & ) [default]

#### 10.112.2.3 LLGL::TextureRegion::TextureRegion ( const Offset3D & *offset*, const Extent3D & *extent* ) [inline]

Constructor to initialize offset and extent only.

### 10.112.3 Member Data Documentation

#### 10.112.3.1 Extent3D LLGL::TextureRegion::extent

Extent of the sub texture region.

**See also**

TextureDescriptor::extent

#### 10.112.3.2 Offset3D LLGL::TextureRegion::offset

Zero-based offset within the texture data.

**Remarks**

Any component of this field that is not meant for the respective texture type is ignored. All other components must be greater than or equal to zero.

#### 10.112.3.3 TextureSubresource LLGL::TextureRegion::subresource

Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.

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

- TextureFlags.h

## 10.113 LLGL::TextureSubresource Struct Reference

Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource.

```
#include <TextureFlags.h>
```

### Public Member Functions

- TextureSubresource ()=default
- TextureSubresource (const TextureSubresource &)=default
- TextureSubresource (std::uint32_t baseArrayLayer, std::uint32_t baseMipLevel)

  *Constructor to initialize base MIP-map level and base array layer only.*
- TextureSubresource (std::uint32_t baseArrayLayer, std::uint32_t numArrayLayers, std::uint32_t baseMip↩Level, std::uint32_t numMipLevels)

  *Constructor to initialize all attributes.*

### Public Attributes

- std::uint32_t baseArrayLayer = 0

  *Zero-based index of the first array layer. By default 0.*
- std::uint32_t numArrayLayers = 1

  *Number of array layers. By default 1.*
- std::uint32_t baseMipLevel = 0

  *MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.*
- std::uint32_t numMipLevels = 1

  *Number of MIP-map levels. By default 1.*

## 10.113.1 Detailed Description

Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource.

#### Remarks

The default values refer to the first array layer and the first MIP-map level.

#### See also

TextureRegion::subresource
TextureViewDescriptor::subresource

## 10.113.2 Constructor & Destructor Documentation

### 10.113.2.1 LLGL::TextureSubresource::TextureSubresource ( ) [default]

### 10.113.2.2 LLGL::TextureSubresource::TextureSubresource ( const TextureSubresource & ) [default]

### 10.113.2.3 LLGL::TextureSubresource::TextureSubresource ( std::uint32_t *baseArrayLayer*, std::uint32_t *baseMipLevel* ) [inline]

Constructor to initialize base MIP-map level and base array layer only.

### 10.113.2.4 LLGL::TextureSubresource::TextureSubresource ( std::uint32_t *baseArrayLayer*, std::uint32_t *numArrayLayers*, std::uint32_t *baseMipLevel*, std::uint32_t *numMipLevels* ) [inline]

Constructor to initialize all attributes.

## 10.113.3 Member Data Documentation

### 10.113.3.1 std::uint32_t LLGL::TextureSubresource::baseArrayLayer = 0

Zero-based index of the first array layer. By default 0.

**Remarks**

Only used by array texture types (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, Texture↩Type::TextureCubeArray, and TextureType::Texture2DMSArray).
This field is ignored by all other texture types.

**See also**

TextureDescriptor::arrayLayers

### 10.113.3.2 std::uint32_t LLGL::TextureSubresource::baseMipLevel = 0

MIP-map level for the sub-texture, where 0 is the base texture, and N $>$ 0 is the N-th MIP-map level. By default 0.

**See also**

TextureDescriptor::mipLevels

### 10.113.3.3 std::uint32_t LLGL::TextureSubresource::numArrayLayers = 1

Number of array layers. By default 1.

**Remarks**

**Must** be greater than zero.

**See also**

TextureDescriptor::arrayLayers

### 10.113.3.4 std::uint32_t LLGL::TextureSubresource::numMipLevels = 1

Number of MIP-map levels. By default 1.

**Remarks**

**Must** be greater than zero.

**See also**

TextureDescriptor::mipLevels

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

- TextureFlags.h

## 10.114 LLGL::TextureSwizzleRGBA Struct Reference

Texture component swizzle structure for red, green, blue, and alpha components.

```
#include <TextureFlags.h>
```

### Public Attributes

- TextureSwizzle r = TextureSwizzle::Red

  *Red component swizzle. By default TextureSwizzle::Red.*
- TextureSwizzle g = TextureSwizzle::Green

  *Green component swizzle. By default TextureSwizzle::Green.*
- TextureSwizzle b = TextureSwizzle::Blue

  *Blue component swizzle. By default TextureSwizzle::Blue.*
- TextureSwizzle a = TextureSwizzle::Alpha

  *Alpha component swizzle. By default TextureSwizzle::Alpha.*

### 10.114.1 Detailed Description

Texture component swizzle structure for red, green, blue, and alpha components.

**Remarks**

Can be used to change the order of texel components independently of a shader.

**See also**

TextureViewDescriptor::swizzle

### 10.114.2 Member Data Documentation

#### 10.114.2.1 TextureSwizzle LLGL::TextureSwizzleRGBA::a = TextureSwizzle::Alpha

Alpha component swizzle. By default TextureSwizzle::Alpha.

#### 10.114.2.2 TextureSwizzle LLGL::TextureSwizzleRGBA::b = TextureSwizzle::Blue

Blue component swizzle. By default TextureSwizzle::Blue.

#### 10.114.2.3 TextureSwizzle LLGL::TextureSwizzleRGBA::g = TextureSwizzle::Green

Green component swizzle. By default TextureSwizzle::Green.

#### 10.114.2.4 TextureSwizzle LLGL::TextureSwizzleRGBA::r = TextureSwizzle::Red

Red component swizzle. By default TextureSwizzle::Red.

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

- TextureFlags.h

## 10.115 LLGL::TextureViewDescriptor Struct Reference

Texture view descriptor structure.

```
#include <TextureFlags.h>
```

### Public Attributes

- TextureType type = TextureType::Texture2D

  *Hardware texture type. By default TextureType::Texture2D.*
- Format format = Format::RGBA8UNorm

  *Hardware texture format. By default Format::RGBA8UNorm.*
- TextureSubresource subresource

  *Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.*
- TextureSwizzleRGBA swizzle

  *Specifies the color component mapping. Each component is mapped to its identity by default.*

### 10.115.1 Detailed Description

Texture view descriptor structure.

**Remarks**

Contains all information about type, format, and dimension to create a texture view that shares the image data of another texture.

**See also**

ResourceViewDescriptor::textureView
RenderingFeatures::hasTextureViews

### 10.115.2 Member Data Documentation

#### 10.115.2.1 Format LLGL::TextureViewDescriptor::format = Format::RGBA8UNorm

Hardware texture format. By default Format::RGBA8UNorm.

**Remarks**

The format of the shared texture and the texture-view must be in the same format class:

| Class | Compatible texture formats |
|---|---|
| 128 Bits | Format::RGBA32UInt, Format::RGBA32SInt, Format::RGBA32Float |
| 96 Bits | Format::RGB32UInt, Format::RGB32SInt, Format::RGB32Float |
| 64 Bits | Format::RG32UInt, Format::RG32SInt, Format::RG32Float, Format::RGBA16UNorm, Format::RGBA16SNorm, Format::RGBA16UInt, Format::RGBA16SInt, Format::RGBA16Float |
| 48 Bits | Format::RGB16UNorm, Format::RGB16SNorm, Format::RGB16UInt, Format::RGB16SInt, Format::RGB16Float |
| 32 Bits | Format::RG16UNorm, Format::RG16SNorm, Format::RG16UInt, Format::RG16SInt, Format::RG16Float, Format::RGBA8UNorm, Format::RGBA8SNorm, Format::RGBA8UInt, Format::RGBA8SInt |
| 24 Bits | Format::RGB8UNorm, Format::RGB8SNorm, Format::RGB8UInt, Format::RGB8SInt |
| 16 Bits | Format::R16UNorm, Format::R16SNorm, Format::R16UInt, Format::R16SInt, Format::R16Float, Format::RG8UNorm, Format::RG8SNorm, Format::RG8UInt, Format::RG8SInt |
| 8 Bits | Format::R8UNorm, Format::R8SNorm, Format::R8UInt, Format::R8SInt |

#### 10.115.2.2 TextureSubresource LLGL::TextureViewDescriptor::subresource

Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.

#### 10.115.2.3 TextureSwizzleRGBA LLGL::TextureViewDescriptor::swizzle

Specifies the color component mapping. Each component is mapped to its identity by default.

**Remarks**

If texture swizzling is not supported, this must be equal to the default value.

**Note**

Only supported with: OpenGL, Vulkan, Metal, Direct3D 12.

**See also**

RenderingFeatures::hasTextureViewSwizzle
IsTextureSwizzleIdentity

#### 10.115.2.4 TextureType LLGL::TextureViewDescriptor::type = TextureType::Texture2D

Hardware texture type. By default TextureType::Texture2D.

| Shared texture type | Compatible texture view types |
|---|---|

#### Remarks

The types of a shared texture can be mapped to the following type of texture-views:

| Shared texture type | Compatible texture view types |
|---|---|
| TextureType::Texture1D | TextureType::Texture1D, TextureType::Texture1DArray |
| TextureType::Texture2D | TextureType::Texture2D, TextureType::Texture2DArray |
| TextureType::Texture3D | TextureType::Texture3D |
| TextureType::TextureCube | TextureType::Texture2D, TextureType::Texture2DArray, TextureType↩ ::TextureCube, TextureType::TextureCubeArray |
| TextureType::Texture1DArray | TextureType::Texture1D, TextureType::Texture1DArray |
| TextureType::Texture2DArray | TextureType::Texture2D, TextureType::Texture2DArray |
| TextureType::TextureCubeArray | TextureType::Texture2D, TextureType::Texture2DArray, TextureType↩ ::TextureCube, TextureType::TextureCubeArray |
| TextureType::Texture2DMS | TextureType::Texture2DMS, TextureType::Texture2DMSArray |
| TextureType::Texture2DMSArray | TextureType::Texture2DMS, TextureType::Texture2DMSArray |

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

- TextureFlags.h

## 10.116 LLGL::Timer Class Reference

Interface for a Timer class.

`#include <Timer.h>`

Inheritance diagram for LLGL::Timer:

### Public Member Functions

- virtual void Start ()=0

  *Starts the timer.*
- virtual std::uint64_t Stop ()=0

  *Stops the timer and returns the elapsed ticks since "Start" was called.*
- virtual std::uint64_t GetFrequency () const =0

  *Returns the frequency resolution of this timer, or rather 'ticks per second' (e.g. for microseconds this is 1000000).*
- virtual bool IsRunning () const =0

  *Returns true if the timer is currently running.*
- void MeasureTime ()

  *Measures the time (elapsed time, and frame count) for each frame.*
- double GetDeltaTime () const

  *Returns the elapsed time (in seconds) between the current and the previous frame.*

### Static Public Member Functions

- static std::unique_ptr< Timer > Create ()

  *Creates a platform specific timer object.*

### Additional Inherited Members

## 10.116.1 Detailed Description

Interface for a Timer class.

**Remarks**

This basic class is also designed as interface, since the native timer is platform specific.

## 10.116.2 Member Function Documentation

### 10.116.2.1 static std::unique_ptr<Timer> LLGL::Timer::Create ( ) [static]

Creates a platform specific timer object.

### 10.116.2.2 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

### 10.116.2.3 virtual std::uint64_t LLGL::Timer::GetFrequency ( ) const [pure virtual]

Returns the frequency resolution of this timer, or rather 'ticks per second' (e.g. for microseconds this is 1000000).

### 10.116.2.4 virtual bool LLGL::Timer::IsRunning ( ) const [pure virtual]

Returns true if the timer is currently running.

**Remarks**

This is true between a call to "Start" and a call to "Stop".

**See also**

Start
Stop

#### 10.116.2.5 void LLGL::Timer::MeasureTime ( )

Measures the time (elapsed time, and frame count) for each frame.

**See also**

GetDeltaTime

#### 10.116.2.6 virtual void LLGL::Timer::Start ( ) [pure virtual]

Starts the timer.

#### 10.116.2.7 virtual std::uint64_t LLGL::Timer::Stop ( ) [pure virtual]

Stops the timer and returns the elapsed ticks since "Start" was called.

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

- Timer.h

## 10.117 LLGL::UninitializeTag Struct Reference

Common uninitialize tag.

```
#include <Tags.h>
```

### 10.117.1 Detailed Description

Common uninitialize tag.

**Remarks**

This can be used to explicitly construct an uninitialized color or even an entire container of uninitialized colors:

```
// Explicitly uninitialized color.
LLGL::ColorRGBAf color { Gs::UninitializeTag{} };

// Explicitly uninitialized color elements in a container.
std::vector<LLGL::ColorRGBAf> color;
color.resize(1024, LLGL::ColorRGBAf { Gs::UninitializeTag{} });
```

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

- Tags.h

## 10.118 LLGL::VertexAttribute Struct Reference

Vertex input 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 Format format, std::uint32_t instanceDivisor=0)

  *Constructs a vertex attribute with a specified name (used for GLSL).*
- VertexAttribute (const std::string &semanticName, std::uint32_t semanticIndex, const Format format, std↩ ::uint32_t instanceDivisor=0)

  *Constructs a vertex attribute with a specified semantic (used for HLSL).*
- std::uint32_t GetSize () const

  *Returns the size (in bytes) which is required for this vertex attribute, or zero if the format is not a valid vertex format.*

### Public Attributes

- std::string name

  *Vertex attribute name (for GLSL) or semantic name (for HLSL).*
- Format format = Format::RGBA32Float

  *Vertex attribute format. By default Format::RGBA32Float.*
- std::uint32_t instanceDivisor = 0

  *Instance data divisor (or instance data step rate).*
- std::uint32_t offset = 0

  *Byte offset within each vertex and each buffer. By default 0.*
- std::uint32_t semanticIndex = 0

  *Semantic index (for HLSL) or vector index (for GLSL).*
- SystemValue systemValue = SystemValue::Undefined

  *Specifies the system value type for this vertex attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.*

### 10.118.1 Detailed Description

Vertex input attribute structure.

**See also**

VertexFormat::attributes
ShaderReflection::vertexAttributes

### 10.118.2 Constructor & Destructor Documentation

#### 10.118.2.1 LLGL::VertexAttribute::VertexAttribute ( ) [default]

#### 10.118.2.2 LLGL::VertexAttribute::VertexAttribute ( const VertexAttribute & ) [default]

#### 10.118.2.3 LLGL::VertexAttribute::VertexAttribute ( const std::string & *name*, const Format *format*, std::uint32_t *instanceDivisor* = 0 )

Constructs a vertex attribute with a specified name (used for GLSL).

**Parameters**

| | | |
|---|---|---|
| in | *name* | Specifies the attribute name (for GLSL). |
| in | *format* | Specifies the attribute format (interpreted as vector format rather than color format). For a 3D-vector type, for example, Format::RGB32Float can be used. |
| 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);
```

**See also**

Format

#### 10.118.2.4 LLGL::VertexAttribute::VertexAttribute ( const std::string & *semanticName*, std::uint32_t *semanticIndex*, const Format *format*, std::uint32_t *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 | *format* | Specifies the attribute format (interpreted as vector format rather than color format). For a 3D-vector type, for example, Format::RGB32Float can be used. |
| 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);
```

**See also**

Format

### 10.118.3 Member Function Documentation

#### 10.118.3.1 std::uint32_t LLGL::VertexAttribute::GetSize ( ) const

Returns the size (in bytes) which is required for this vertex attribute, or zero if the format is not a valid vertex format.

**See also**

FormatDescriptor::bitSize
Format

#### 10.118.3.2 VertexAttribute& LLGL::VertexAttribute::operator= ( const VertexAttribute & ) [default]

### 10.118.4 Member Data Documentation

#### 10.118.4.1 Format LLGL::VertexAttribute::format = Format::RGBA32Float

Vertex attribute format. By default Format::RGBA32Float.

**Remarks**

Not all hardware formats are allowed for vertex attributes. In particular, depth-stencil formats and compressed formats are not allowed. To specify a vertex attribute of a matrix type, multiple attributes with ascending semantic indices must be used. Here is an example of a 4x4 matrix:

```
myVertexFormat.AppendAttribute({ "myMatrix4x4", 0, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 1, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 2, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 3, LLGL::Format::RGBA32Float });
```

Here is an example of a 2x2 matrix:

```
myVertexFormat.AppendAttribute({ "myMatrix2x2", 0, LLGL::Format::RG32Float });
myVertexFormat.AppendAttribute({ "myMatrix2x2", 1, LLGL::Format::RG32Float });
```

#### 10.118.4.2 std::uint32_t LLGL::VertexAttribute::instanceDivisor = 0

Instance data divisor (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.

#### 10.118.4.3 std::string LLGL::VertexAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

#### 10.118.4.4 std::uint32_t LLGL::VertexAttribute::offset = 0

Byte offset within each vertex and each buffer. By default 0.

#### 10.118.4.5 std::uint32_t LLGL::VertexAttribute::semanticIndex = 0

Semantic index (for HLSL) or vector index (for GLSL).

**Remarks**

This is used when a matrix is distributed over multiple vector attributes.

#### 10.118.4.6 SystemValue LLGL::VertexAttribute::systemValue = SystemValue::Undefined

Specifies the system value type for this vertex attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.

**Remarks**

System value semantics must not be specified to create a shader program. Instead, they are used only for shader code reflection. Examples of system value semantics are:

- Vertex ID: `SV_VertexID` (HLSL), `gl_VertexID` (GLSL), `gl_VertexIndex` (SPIR-V), `[[vertex_id]]` (Metal).
- Instance ID: `SV_InstanceID` (HLSL), `gl_InstanceID` (GLSL), `gl_InstanceIndex` (SPIR-V), `[[instance_id]]` (Metal).

**See also**

ShaderProgram::Reflect

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

- VertexAttribute.h

## 10.119 LLGL::BufferDescriptor::VertexBuffer Struct Reference

Vertex buffer specific descriptor structure.

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

### Public Attributes

- VertexFormat format
  *Specifies the vertex format layout.*

### 10.119.1 Detailed Description

Vertex buffer specific descriptor structure.

### 10.119.2 Member Data Documentation

#### 10.119.2.1 VertexFormat LLGL::BufferDescriptor::VertexBuffer::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

## 10.120 LLGL::VertexFormat Struct Reference

Vertex format structure.

```
#include <VertexFormat.h>
```

### Public Member Functions

- void AppendAttribute (const VertexAttribute &attrib, std::uint32_t offset=Constants::ignoreOffset)

  *Appends the specified vertex attribute to this vertex format.*

### Public Attributes

- std::vector< VertexAttribute > attributes

  *Specifies the list of vertex attributes.*
- std::uint32_t stride = 0

  *Specifies the vertex data stride (or format size) which describes the byte offset between consecutive vertices.*
- std::uint32_t inputSlot = 0

  *Vertex buffer input slot. By default 0.*

### 10.120.1 Detailed Description

Vertex format structure.

**Remarks**

A vertex format is required to describe how the vertex attributes are layed out inside a vertex buffer.

**See also**

BufferDescriptor::VertexBuffer::format
ShaderProgramDescriptor::vertexFormats

**Todo** Remove this struct and move both `stride` and `inputSlot` into `VertexAttribute`.

### 10.120.2 Member Function Documentation

#### 10.120.2.1 void LLGL::VertexFormat::AppendAttribute ( const VertexAttribute & *attrib*, std::uint32_t *offset* = `Constants::ignoreOffset` )

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 equal to `Constants::ignoreOffset`, 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 `Constants::ignoreOffset`. |

**Remarks**

This function modifies the `offset` member of specified attribute before adding it to the `attributes` list, and sets the `stride` member to the sum of the size of all attributes.

**See also**

VertexAttribute::offset
Constants::ignoreOffset

### 10.120.3 Member Data Documentation

#### 10.120.3.1 std::vector<VertexAttribute> LLGL::VertexFormat::attributes

Specifies the list of vertex attributes.

**See also**

AppendAttribute

#### 10.120.3.2 std::uint32_t LLGL::VertexFormat::inputSlot = 0

Vertex buffer input slot. By default 0.

**Remarks**

This is used when multiple vertex buffers are used simultaneously.

**Note**

Only supported with: Direct3D 11, Direct3D 12, Vulkan.
For OpenGL, the input slots are automatically generated in ascending order and beginning with zero.

#### 10.120.3.3 std::uint32_t 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 everytime `AppendAttribute` is called, but it can also be modified manually. It is commonly the size of all vertex attributes.

**See also**

AppendAttribute

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

- VertexFormat.h

## 10.121 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 (e.g. "NVIDIA Corporation", "Advanced Micro Devices, Inc." etc.).*
- std::uint64_t videoMemory = 0

  *Video memory size (in bytes).*
- std::vector< VideoOutputDescriptor > outputs

  *List of all adapter output descriptors.*

### 10.121.1 Detailed Description

Video adapter descriptor structure.

**Remarks**

A video adapter determines the output capabilities of a GPU.

**Todo** Currently unused in the interface.

### 10.121.2 Member Data Documentation

#### 10.121.2.1 std::wstring LLGL::VideoAdapterDescriptor::name

Hardware adapter name (name of the GPU).

#### 10.121.2.2 std::vector<VideoOutputDescriptor> LLGL::VideoAdapterDescriptor::outputs

List of all adapter output descriptors.

#### 10.121.2.3 std::string LLGL::VideoAdapterDescriptor::vendor

Vendor name (e.g. "NVIDIA Corporation", "Advanced Micro Devices, Inc." etc.).

#### 10.121.2.4 std::uint64_t LLGL::VideoAdapterDescriptor::videoMemory = 0

Video memory size (in bytes).

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

- VideoAdapter.h

## 10.122 LLGL::VideoModeDescriptor Struct Reference

Video mode descriptor structure.

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

### Public Attributes

- Extent2D resolution

  *Screen resolution (in pixels).*
- int colorBits = 32

  *Number of bits for each pixel in the color buffer. Should be 24 or 32. By default 32.*
- int depthBits = 24

  *Number of bits for each pixel in the depth buffer. Should be 24, 32, or zero to disable depth buffer. By default 24.*
- int stencilBits = 8

  *Number of bits for each pixel in the stencil buffer. Should be 8, or zero to disable stencil buffer. By default 8.*
- bool fullscreen = false

  *Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.*
- std::uint32_t swapChainSize = 2

  *Number of swap-chain buffers. By default 2 (for double-buffering).*

### 10.122.1 Detailed Description

Video mode descriptor structure.

#### Remarks

This is mainly used to set the video mode of a RenderContext object. The counterpart for a physical display mode is the DisplayModeDescriptor structure.

#### See also

RenderContext::SetVideoMode
DisplayModeDescriptor

### 10.122.2 Member Data Documentation

#### 10.122.2.1 int LLGL::VideoModeDescriptor::colorBits = 32

Number of bits for each pixel in the color buffer. Should be 24 or 32. By default 32.

**Remarks**

This is only a hint to the renderer and there is no guarantee which hardware format is finally used for the color buffer. To determine the actual color format of a render context, use the RenderContext::GetColorFormat function.

**See also**

RenderContext::GetColorFormat

#### 10.122.2.2 int LLGL::VideoModeDescriptor::depthBits = 24

Number of bits for each pixel in the depth buffer. Should be 24, 32, or zero to disable depth buffer. By default 24.

**Remarks**

This is only a hint to the renderer and there is no guarantee which hardware format is finally used for the depth buffer. To determine the actual depth-stencil format of a render context, use the RenderContext::GetDepth↩StencilFormat function.

**See also**

RenderContext::GetDepthStencilFormat

#### 10.122.2.3 bool LLGL::VideoModeDescriptor::fullscreen = false

Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

#### 10.122.2.4 Extent2D LLGL::VideoModeDescriptor::resolution

Screen resolution (in pixels).

**Remarks**

If the resolution contains a member with a value of 0, the video mode is invalid.

**See also**

RenderTarget::GetResolution

#### 10.122.2.5 int LLGL::VideoModeDescriptor::stencilBits = 8

Number of bits for each pixel in the stencil buffer. Should be 8, or zero to disable stencil buffer. By default 8.

**Remarks**

This is only a hint to the renderer and there is no guarantee which hardware format is finally used for the stencil buffer. To determine the actual depth-stencil format of a render context, use the RenderContext::GetDepthStencilFormat function.

**See also**

RenderContext::GetDepthStencilFormat

#### 10.122.2.6 std::uint32_t LLGL::VideoModeDescriptor::swapChainSize = 2

Number of swap-chain buffers. By default 2 (for double-buffering).

**Remarks**

This is only a hint to the renderer and there is no guarantee how many buffers are finally used for the swap chain. Especially OpenGL does not support custom swap chain sizes. If this value is 0, the video mode is invalid.

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

- RenderContextFlags.h

## 10.123 LLGL::VideoOutputDescriptor Struct Reference

Video output structure.

```
#include <VideoAdapter.h>
```

### Public Attributes

- std::vector< DisplayModeDescriptor > displayModes

  *List of all display mode descriptors for this video output.*

### 10.123.1 Detailed Description

Video output structure.

**See also**

VideoAdapterDescriptor::outputs

**Todo** Currently unused in the interface.

### 10.123.2 Member Data Documentation

#### 10.123.2.1 std::vector<DisplayModeDescriptor> LLGL::VideoOutputDescriptor::displayModes

List of all display mode descriptors for this video output.

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

- VideoAdapter.h

## 10.124 LLGL::Viewport Struct Reference

Viewport dimensions.

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

### Public Member Functions

- Viewport ()=default
- Viewport (const Viewport &)=default
- Viewport (float x, float y, float width, float height)

  *Viewport constructor with default depth range of [0, 1].*
- Viewport (float x, float y, float width, float height, float minDepth, float maxDepth)

  *Viewport constructor with parameters for all attributes.*
- Viewport (const Extent2D &extent)

  *Viewport constructor with extent and default depth range of [0, 1].*
- Viewport (const Extent2D &extent, float minDepth, float maxDepth)

  *Viewport constructor with extent and explicit depth range.*
- Viewport (const Offset2D &offset, const Extent2D &extent)

  *Viewport constructor with offset, extent, and default depth range of [0, 1].*
- Viewport (const Offset2D &offset, const Extent2D &extent, float minDepth, float maxDepth)

  *Viewport constructor with offset, extent, and explicit depth range.*

### Public Attributes

- float x = 0.0f

  *Y coordinate of the left-top origin. By default 0.0.*
- float y = 0.0f
- float width = 0.0f

  *Width of the right-bottom size. By default 0.0.*
- float height = 0.0f

  *Height of the right-bottom size. By default 0.0.*
- float minDepth = 0.0f

  *Minimum of the depth range. Must be in the range [0, 1]. By default 0.0.*
- float maxDepth = 1.0f

  *Maximum of the depth range. Must be in the range [0, 1]. By default 1.0.*

### 10.124.1 Detailed Description

Viewport dimensions.

**Remarks**

A viewport is in screen coordinates where the origin is in the left-top corner.

**See also**

CommandBuffer::SetViewport
CommandBuffer::SetViewports
GraphicsPipelineDescriptor::viewports

### 10.124.2 Constructor & Destructor Documentation

#### 10.124.2.1 LLGL::Viewport::Viewport ( ) [default]

#### 10.124.2.2 LLGL::Viewport::Viewport ( const Viewport & ) [default]

#### 10.124.2.3 LLGL::Viewport::Viewport ( float *x*, float *y*, float *width*, float *height* ) [inline]

Viewport constructor with default depth range of [0, 1].

#### 10.124.2.4 LLGL::Viewport::Viewport ( float *x*, float *y*, float *width*, float *height*, float *minDepth*, float *maxDepth* ) [inline]

Viewport constructor with parameters for all attributes.

#### 10.124.2.5 LLGL::Viewport::Viewport ( const Extent2D & *extent* ) [inline]

Viewport constructor with extent and default depth range of [0, 1].

#### 10.124.2.6 LLGL::Viewport::Viewport ( const Extent2D & *extent*, float *minDepth*, float *maxDepth* ) [inline]

Viewport constructor with extent and explicit depth range.

#### 10.124.2.7 LLGL::Viewport::Viewport ( const Offset2D & *offset*, const Extent2D & *extent* ) [inline]

Viewport constructor with offset, extent, and default depth range of [0, 1].

#### 10.124.2.8 LLGL::Viewport::Viewport ( const Offset2D & *offset*, const Extent2D & *extent*, float *minDepth*, float *maxDepth* ) [inline]

Viewport constructor with offset, extent, and explicit depth range.

X coordinate of the left-top origin. By default 0.0.

### 10.124.3 Member Data Documentation

#### 10.124.3.1 float LLGL::Viewport::height = 0.0f

Height of the right-bottom size. By default 0.0.

**Remarks**

Setting a viewport of negative height results in undefined behavior.

#### 10.124.3.2 float LLGL::Viewport::maxDepth = 1.0f

Maximum of the depth range. Must be in the range [0, 1]. By default 1.0.

**Remarks**

Reverse mappings such as minDepth=1 and maxDepth=0 are also valid.

#### 10.124.3.3 float LLGL::Viewport::minDepth = 0.0f

Minimum of the depth range. Must be in the range [0, 1]. By default 0.0.

**Remarks**

Reverse mappings such as minDepth=1 and maxDepth=0 are also valid.

#### 10.124.3.4 float LLGL::Viewport::width = 0.0f

Width of the right-bottom size. By default 0.0.

**Remarks**

Setting a viewport of negative width results in undefined behavior.

#### 10.124.3.5 float LLGL::Viewport::x = 0.0f

Y coordinate of the left-top origin. By default 0.0.

#### 10.124.3.6 float LLGL::Viewport::y = 0.0f

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

- GraphicsPipelineFlags.h

## 10.125 LLGL::VsyncDescriptor Struct Reference

Vertical-synchronization (Vsync) descriptor structure.

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

### Public Attributes

- bool enabled = false

  *Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.*
- std::uint32_t refreshRate = 60

  *Refresh rate (in Hz). By default 60.*
- std::uint32_t interval = 1

  *Synchronisation interval. Can be 1, 2, 3, or 4. By default 1.*

### 10.125.1 Detailed Description

Vertical-synchronization (Vsync) descriptor structure.

**Todo** Maybe remove this entire structure and only use a "vsyncInterval" parameter.

### 10.125.2 Member Data Documentation

#### 10.125.2.1 bool LLGL::VsyncDescriptor::enabled = false

Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.

#### 10.125.2.2 std::uint32_t LLGL::VsyncDescriptor::interval = 1

Synchronisation interval. Can be 1, 2, 3, or 4. By default 1.

**Remarks**

If Vsync is disabled, this value is implicitly zero.

#### 10.125.2.3 std::uint32_t LLGL::VsyncDescriptor::refreshRate = 60

Refresh rate (in Hz). By default 60.

**Note**

Only supported with: Direct3D 11, Direct3D 12, Metal.

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

- RenderContextFlags.h

## 10.126 LLGL::Window Class Reference

Window interface for desktop platforms.

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

Inheritance diagram for LLGL::Window:

### Classes

- class EventListener

  *Interface for all window event listeners.*

### Public Member Functions

- virtual void SetPosition (const Offset2D &position)=0

  *Sets the window position relative to its parent.*
- virtual Offset2D GetPosition () const =0

  *Returns the window position relative to its parent (which can also be the display).*
- virtual void SetSize (const Extent2D &size, bool useClientArea=true)=0

  *Sets the either the overall window size or the client area size. By default the client area size is set.*
- virtual Extent2D GetSize (bool useClientArea=true) const =0

  *Returns either the overall window size or the client area size. By default the client area size is returned.*
- virtual void SetTitle (const std::wstring &title)=0

  *Sets the window title as UTF16 string. If the OS does not support UTF16 window title, it will be converted to UTF8.*
- virtual std::wstring GetTitle () const =0

  *Returns the window title as UTF16 string.*
- virtual void Show (bool show=true)=0

  *Shows or hides the window.*
- virtual bool IsShown () const =0

  *Returns true if this window is visible.*
- virtual void SetDesc (const WindowDescriptor &desc)=0

  *Sets the window attributes according to the specified window descriptor.*
- virtual WindowDescriptor GetDesc () const =0

  *Queries a window descriptor, which describes the attributes of this window.*
- virtual void SetBehavior (const WindowBehavior &behavior)

  *Sets the new window behavior.*
- const WindowBehavior & GetBehavior () const

  *Returns the window heavior.*

- virtual bool HasFocus () const

  *Returns true if this window has the keyboard focus.*
- virtual bool HasQuit () const

  *Returns true if this window is in the 'Quit' state.*
- bool AdaptForVideoMode (VideoModeDescriptor &videoModeDesc) override

  *Adapts the window for the specified video mode.*
- bool ProcessEvents () override final

  *Processes the events for this window (i.e. mouse movement, key presses etc.).*
- void AddEventListener (const std::shared_ptr< EventListener > &eventListener)

  *Adds the specified event listener to this window.*
- void RemoveEventListener (const EventListener *eventListener)

  *Removes the specified event listener from this window.*
- void PostQuit ()

  *Posts a 'Quit' event to all event listeners if the window is not yet in the 'Quit' state.*
- void PostKeyDown (Key keyCode)

  *Posts a 'KeyDown' event to all event listeners.*
- void PostKeyUp (Key keyCode)
- void PostDoubleClick (Key keyCode)
- void PostChar (wchar_t chr)
- void PostWheelMotion (int motion)
- void PostLocalMotion (const Offset2D &position)
- void PostGlobalMotion (const Offset2D &motion)
- void PostResize (const Extent2D &clientAreaSize)
- void PostGetFocus ()

  *Posts a 'GetFocus' event to all event listeners.*
- void PostLostFocus ()

  *Posts a 'LostFocus' event to all event listeners.*
- void PostTimer (std::uint32_t timerID)

  *Posts a timer event with the specified timer ID number.*

### Static Public Member Functions

- static std::unique_ptr< Window > Create (const WindowDescriptor &desc)

  *Creates a platform specific instance of the Window interface.*

### Protected Member Functions

- virtual void OnProcessEvents ()=0

  *Called inside the "ProcessEvents" function after all event listeners received the same event.*

## 10.126.1 Detailed Description

Window interface for desktop platforms.

#### Remarks

This is the main interface for the windowing system in LLGL. To implement a custom window (and use GLFW for instance) you have to derive from this class and implement all pure virtual functions. The couterpart is the Canvas interface for mobile platforms.

#### See also

Canvas

### 10.126.2 Member Function Documentation

#### 10.126.2.1 bool LLGL::Window::AdaptForVideoMode ( VideoModeDescriptor & *videoModeDesc* ) [override], [virtual]

Adapts the window for the specified video mode.

**Remarks**

This is a default implementation of the base class function and makes use of "GetDesc" and "SetDesc".

**See also**

GetDesc
SetDesc

Implements LLGL::Surface.

#### 10.126.2.2 void LLGL::Window::AddEventListener ( const std::shared_ptr< EventListener > & *eventListener* )

Adds the specified event listener to this window.

#### 10.126.2.3 static std::unique_ptr<Window> LLGL::Window::Create ( const WindowDescriptor & *desc* ) [static]

Creates a platform specific instance of the Window interface.

**Returns**

Unique pointer to a new instance of the platform specific Window interface or null if the platform does not support windows (such as Android and iOS).

**Remarks**

For mobile platforms the interface Canvas can be used.

**See also**

Canvas

#### 10.126.2.4 const WindowBehavior& LLGL::Window::GetBehavior ( ) const [inline]

Returns the window heavior.

#### 10.126.2.5 virtual WindowDescriptor LLGL::Window::GetDesc ( ) const [pure virtual]

Queries a window descriptor, which describes the attributes of this window.

#### 10.126.2.6 virtual Offset2D LLGL::Window::GetPosition ( ) const [pure virtual]

Returns the window position relative to its parent (which can also be the display).

#### 10.126.2.7 virtual Extent2D LLGL::Window::GetSize ( bool *useClientArea* = true ) const [pure virtual]

Returns either the overall window size or the client area size. By default the client area size is returned.

#### 10.126.2.8 virtual std::wstring LLGL::Window::GetTitle ( ) const [pure virtual]

Returns the window title as UTF16 string.

#### 10.126.2.9 virtual bool LLGL::Window::HasFocus ( ) const [virtual]

Returns true if this window has the keyboard focus.

#### 10.126.2.10 virtual bool LLGL::Window::HasQuit ( ) const [virtual]

Returns true if this window is in the 'Quit' state.

**See also**

PostQuit
ProcessEvents

#### 10.126.2.11 virtual bool LLGL::Window::IsShown ( ) const [pure virtual]

Returns true if this window is visible.

#### 10.126.2.12 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

#### 10.126.2.13 void LLGL::Window::PostChar ( wchar_t *chr* )

**See also**

PostKeyDown

#### 10.126.2.14 void LLGL::Window::PostDoubleClick ( Key *keyCode* )

**See also**

PostKeyDown

#### 10.126.2.15 void LLGL::Window::PostGetFocus ( )

Posts a 'GetFocus' event to all event listeners.

#### 10.126.2.16 void LLGL::Window::PostGlobalMotion ( const Offset2D & *motion* )

**See also**

PostKeyDown

#### 10.126.2.17 void LLGL::Window::PostKeyDown ( Key *keyCode* )

Posts a 'KeyDown' event to all event listeners.

**Remarks**

This will be called automatically by the "ProcessEvents" function.

**See also**

EventListener::OnKeyDown
ProcessEvents

#### 10.126.2.18 void LLGL::Window::PostKeyUp ( Key *keyCode* )

**See also**

PostKeyDown

#### 10.126.2.19 void LLGL::Window::PostLocalMotion ( const Offset2D & *position* )

**See also**

PostKeyDown

#### 10.126.2.20 void LLGL::Window::PostLostFocus ( )

Posts a 'LostFocus' event to all event listeners.

#### 10.126.2.21 void LLGL::Window::PostQuit ( )

Posts a 'Quit' event to all event listeners if the window is not yet in the 'Quit' state.

**Remarks**

If one or more event listeners set the `veto` parameter to true in the `OnQuit` callback, the window will not quit. Otherwise, the `ProcessEvents` function will return false from then on.

**See also**

EventListener::OnQuit
ProcessEvents
HasQuit

#### 10.126.2.22 void LLGL::Window::PostResize ( const Extent2D & *clientAreaSize* )

**See also**

PostKeyDown

#### 10.126.2.23 void LLGL::Window::PostTimer ( std::uint32_t *timerID* )

Posts a timer event with the specified timer ID number.

**Remarks**

This can be used to refresh the screen while the underlying window is currently being moved or resized by the user.

**Note**

Only supported on: MS. Windows.

#### 10.126.2.24 void LLGL::Window::PostWheelMotion ( int *motion* )

**See also**

PostKeyDown

#### 10.126.2.25 bool LLGL::Window::ProcessEvents ( ) `[final],[override],[virtual]`

Processes the events for this window (i.e. mouse movement, key presses etc.).

**Returns**

True, as long as the window can process events. Once the `PostQuit` function has set this window to the 'Quit' state, this function returns false. This happens when the user clicks on the close button.

**See also**

PostQuit

Implements LLGL::Surface.

#### 10.126.2.26 void LLGL::Window::RemoveEventListener ( const EventListener * *eventListener* )

Removes the specified event listener from this window.

#### 10.126.2.27 virtual void LLGL::Window::SetBehavior ( const WindowBehavior & *behavior* ) [virtual]

Sets the new window behavior.

#### 10.126.2.28 virtual void LLGL::Window::SetDesc ( const WindowDescriptor & *desc* ) [pure virtual]

Sets the window attributes according to the specified window descriptor.

**Remarks**

This is used by the RenderContext interface when the video mode is about to change.

**See also**

RenderContext::SetVideoMode

#### 10.126.2.29 virtual void LLGL::Window::SetPosition ( const Offset2D & *position* ) [pure virtual]

Sets the window position relative to its parent.

#### 10.126.2.30 virtual void LLGL::Window::SetSize ( const Extent2D & *size*, bool *useClientArea* = true ) [pure virtual]

Sets the either the overall window size or the client area size. By default the client area size is set.

#### 10.126.2.31 virtual void LLGL::Window::SetTitle ( const std::wstring & *title* ) [pure virtual]

Sets the window title as UTF16 string. If the OS does not support UTF16 window title, it will be converted to UTF8.

#### 10.126.2.32 virtual void LLGL::Window::Show ( bool *show* = true ) [pure virtual]

Shows or hides the window.

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

- Window.h

## 10.127 LLGL::WindowBehavior Struct Reference

Window behavior structure.

```
#include <WindowFlags.h>
```

### Public Attributes

- bool disableClearOnResize = false

  *Specifies whether to clear the content of the window when it is resized. By default false.*
- std::uint32_t moveAndResizeTimerID = invalidWindowTimerID

  *Specifies an ID for a timer which will be activated when the window is moved or sized. By default invalidWindow↩ TimerID.*

### 10.127.1 Detailed Description

Window behavior structure.

**See also**

Window::SetBehavior

### 10.127.2 Member Data Documentation

#### 10.127.2.1 bool LLGL::WindowBehavior::disableClearOnResize = false

Specifies whether to clear the content of the window when it is resized. By default false.

**Remarks**

This is used by Win32 to erase (WM_ERASEBKGND message) or keep the background on a window resize. If this is false, some kind of flickering during a window resize can be avoided.

**Note**

Only supported on: Win32.

#### 10.127.2.2 std::uint32_t LLGL::WindowBehavior::moveAndResizeTimerID = invalidWindowTimerID

Specifies an ID for a timer which will be activated when the window is moved or sized. By default invalidWindow↩ TimerID.

**Remarks**

This is used by Win32 to set a timer during a window is moved or resized to make continous scene updates. Do not reset it during the 'OnTimer' event, otherwise a timer might be not be released correctly!

**Note**

Only supported on: Win32.

**See also**

Window::EventListener::OnTimer
invalidWindowTimerID

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

- WindowFlags.h

## 10.128 LLGL::WindowDescriptor Struct Reference

Window descriptor structure.

```
#include <WindowFlags.h>
```

### Public Attributes

- std::wstring title

  *Window title as unicode string.*
- Offset2D position

  *Window position (relative to the client area).*
- Extent2D size

  *Window size (this should be the client area size).*
- bool visible = false

  *Specifies whether the window is visible at creation time. By default false.*
- bool borderless = false

  *Specifies whether the window is borderless. This is required for a fullscreen render context. By default false.*
- bool resizable = false

  *Specifies whether the window can be resized. By default false.*
- bool acceptDropFiles = false

  *Specifies whether the window allows that files can be draged-and-droped onto the window. By default false.*
- bool preventForPowerSafe = false

  *Specifies whether this window prevents the host system for power-safe mode. By default false.*
- bool centered = false

  *Specifies whether the window is centered within the desktop screen. By default false.*
- const void * windowContext = nullptr

  *Window context handle.*

### 10.128.1 Detailed Description

Window descriptor structure.

### 10.128.2 Member Data Documentation

#### 10.128.2.1 bool LLGL::WindowDescriptor::acceptDropFiles = false

Specifies whether the window allows that files can be draged-and-droped onto the window. By default false.

**Note**

Only supported on: MS/Windows.

#### 10.128.2.2 bool LLGL::WindowDescriptor::borderless = false

Specifies whether the window is borderless. This is required for a fullscreen render context. By default false.

#### 10.128.2.3 bool LLGL::WindowDescriptor::centered = false

Specifies whether the window is centered within the desktop screen. By default false.

#### 10.128.2.4 Offset2D LLGL::WindowDescriptor::position

Window position (relative to the client area).

#### 10.128.2.5 bool LLGL::WindowDescriptor::preventForPowerSafe = false

Specifies whether this window prevents the host system for power-safe mode. By default false.

**Note**

Only supported on: MS/Windows.

#### 10.128.2.6 bool LLGL::WindowDescriptor::resizable = false

Specifies whether the window can be resized. By default false.

**Remarks**

For every window representing the surface for a RenderContext which has been resized, the video mode of that RenderContext must be updated with the resolution of the surface's content size. This can be done by setting the video mode with the new resolution before the respective render context is set as render target, or it can be handled by a window event listener on the 'OnResize' callback:

```
// Alternative 1
class MyEventListener : public LLGL::Window::EventListener {
    void OnResize(Window& sender, const Extent2D& clientAreaSize) override {
        auto myVideoMode = myRenderContext->GetVideoMode();
        myVideoMode.resolution = clientAreaSize;
        myRenderContext->SetVideoMode(myVideoMode);
    }
};
myWindow->AddEventListener(std::make_shared<MyEventListener>());

// Alternative 2
auto myVideoMode = myRenderContext->GetVideoMode();
myVideoMode.resolution = myWindow->GetContentSize();
myRenderContext->SetVideoMode(myVideoMode);
myCmdBuffer->SetRenderTarget(*myRenderContext);
```

**Note**

Not updating the render context on a resized window is undefined behavior.

**See also**

RenderContext::SetVideoMode
Surface::GetContentSize
Window::EventListener::OnResize

#### 10.128.2.7 Extent2D LLGL::WindowDescriptor::size

Window size (this should be the client area size).

#### 10.128.2.8 std::wstring LLGL::WindowDescriptor::title

Window title as unicode string.

#### 10.128.2.9 bool LLGL::WindowDescriptor::visible = false

Specifies whether the window is visible at creation time. By default false.

#### 10.128.2.10 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:

- WindowFlags.h

# Chapter 11

# File Documentation

## 11.1 Buffer.h File Reference

```
#include "Resource.h"
#include "BufferFlags.h"
```

### Classes

- class LLGL::Buffer

  *Hardware buffer interface.*

### Namespaces

- LLGL

## 11.2 BufferArray.h File Reference

```
#include "RenderSystemChild.h"
#include "BufferFlags.h"
```

### Classes

- class LLGL::BufferArray

  *Hardware buffer container interface.*

### Namespaces

- LLGL

## 11.3 BufferFlags.h File Reference

```
#include "Export.h"
#include "ResourceFlags.h"
#include "VertexFormat.h"
#include "RenderSystemFlags.h"
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::BufferDescriptor

  *Hardware buffer descriptor structure.*
- struct LLGL::BufferDescriptor::VertexBuffer

  *Vertex buffer specific descriptor structure.*
- struct LLGL::BufferDescriptor::IndexBuffer

  *Index buffer specific descriptor structure.*
- struct LLGL::BufferDescriptor::StorageBuffer

  *Storage buffer specific descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::StorageBufferType { LLGL::StorageBufferType::Undefined, 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.*

### Functions

- LLGL_EXPORT bool LLGL::IsRWBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a read/write (RW) buffer.*
- LLGL_EXPORT bool LLGL::IsTypedBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a simply typed buffer.*
- LLGL_EXPORT bool LLGL::IsStructuredBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a structured buffer.*
- LLGL_EXPORT bool LLGL::IsByteAddressBuffer (const StorageBufferType type)

  *Returns true if the storage buffer type denotes a byte addresse buffer.*

## 11.4 Canvas.h File Reference

```
#include "CanvasFlags.h"
#include "Surface.h"
#include "Types.h"
#include "Key.h"
#include <memory>
```

### Classes

- class LLGL::Canvas

  *Canvas interface for mobile platforms.*
- class LLGL::Canvas::EventListener

  *Interface for all canvas event listeners.*

### Namespaces

- LLGL

## 11.5 CanvasFlags.h File Reference

```
#include <string>
```

### Classes

- struct LLGL::CanvasDescriptor

  *Canvas descriptor structure.*

### Namespaces

- LLGL

## 11.6 Color.h File Reference

```
#include "Tags.h"
#include <algorithm>
#include <type_traits>
#include <cstdint>
#include <stdexcept>
```

## 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<>
  std::uint8_t LLGL::MaxColorValue< std::uint8_t > ()

  *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 Dst , typename Src >
  Dst LLGL::CastColorValue (const Src &value)

  *Casts the specified color value and transforms it from the source data type range to the destination data type range.*
- template<>
  bool LLGL::CastColorValue< bool, bool > (const bool &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  float LLGL::CastColorValue< float, float > (const float &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  double LLGL::CastColorValue< double, double > (const double &value)

  *Specialized template which merely passes the input value as output.*
- template<>
  std::uint8_t LLGL::CastColorValue< std::uint8_t, std::uint8_t > (const std::uint8_t &value)

  *Specialized template which merely passes the input value as output.*
- 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>
  Color< T, N > LLGL::operator/ (const T &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)

  *Returns true if all components of both colors 'lhs' and 'rhs' are equal.*
- template<typename T , std::size_t N>
  bool LLGL::operator!= (const Color< T, N > &lhs, const Color< T, N > &rhs)

  *Returns true if any component of both colors 'lhs' and 'rhs' are unequal.*

## 11.7 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::ColorRGBb = ColorRGBT< bool >
- using LLGL::ColorRGBf = ColorRGBT< float >
- using LLGL::ColorRGBd = ColorRGBT< double >
- using LLGL::ColorRGBub = ColorRGBT< std::uint8_t >

## 11.8 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::ColorRGBAb = ColorRGBAT< bool >
- using LLGL::ColorRGBAf = ColorRGBAT< float >
- using LLGL::ColorRGBAd = ColorRGBAT< double >
- using LLGL::ColorRGBAub = ColorRGBAT< std::uint8_t >

## 11.9 CommandBuffer.h File Reference

```
#include "RenderSystemChild.h"
#include "CommandBufferFlags.h"
#include "RenderSystemFlags.h"
#include "ColorRGBA.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "ResourceHeap.h"
#include "PipelineLayoutFlags.h"
#include "RenderPass.h"
#include "RenderTarget.h"
#include "ShaderProgram.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "QueryHeap.h"
#include <cstdint>
```

### Classes

- class LLGL::CommandBuffer

  *Command buffer interface used for storing and encoding GPU commands.*

### Namespaces

- LLGL

## 11.10 CommandBufferFlags.h File Reference

```
#include "ColorRGBA.h"
```

### Classes

- struct LLGL::CommandBufferFlags

  *Command buffer creation flags.*
- struct LLGL::ClearFlags

  *Command buffer clear flags.*
- struct LLGL::ClearValue

  *Clear value structure for color, depth, and stencil clear operations.*
- struct LLGL::AttachmentClear

  *Attachment clear command structure.*
- struct LLGL::OpenGLDependentStateDescriptor

  *Graphics API dependent state descriptor for the OpenGL renderer.*
- struct LLGL::CommandBufferDescriptor

  *Command buffer descriptor structure.*

### 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.*

## 11.11 CommandQueue.h File Reference

```
#include "RenderSystemChild.h"
#include "ForwardDecls.h"
#include <cstdint>
#include <cstddef>
```

### Classes

- class LLGL::CommandQueue

  *Command queue interface.*

### Namespaces

- LLGL

## 11.12 ComputePipeline.h File Reference

```
#include "RenderSystemChild.h"
```

### Classes

- class LLGL::ComputePipeline

  *Compute pipeline interface.*

### Namespaces

- LLGL

## 11.13 ComputePipelineFlags.h File Reference

### Classes

- struct LLGL::ComputePipelineDescriptor

  *Compute pipeline descriptor structure.*

### Namespaces

- LLGL

## 11.14 Constants.h File Reference

```
#include <cstdint>
```

### Namespaces

- LLGL
- LLGL::Constants

  *Namespace with all constants used as default arguments.*

## 11.15 Display.h File Reference

```
#include "Interface.h"
#include "DisplayFlags.h"
#include <vector>
#include <memory>
#include <string>
```

### Classes

- class LLGL::Display

  *Display interface to query the attributes of all connected displays/monitors.*

### Namespaces

- LLGL

## 11.16 DisplayFlags.h File Reference

```
#include "Export.h"
#include "Types.h"
```

### Classes

- struct LLGL::DisplayModeDescriptor

  *Display mode descriptor structure.*

### Namespaces

- LLGL

### Functions

- LLGL_EXPORT bool LLGL::operator== (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

  *Compares the two specified display mode descriptors on equality.*
- LLGL_EXPORT bool LLGL::operator!= (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

  *Compares the two specified display mode descriptors on inequality.*
- LLGL_EXPORT bool LLGL::CompareSWO (const DisplayModeDescriptor &lhs, const DisplayMode↩ Descriptor &rhs)

  *Compares the two display modes in a strict-weak-order (SWO) fashion.*
- LLGL_EXPORT Extent2D LLGL::GetExtentRatio (const Extent2D &extent)

  *Returns the ratio of the specified extent as another extent, i.e. all attributes are divided by their greatest common divisor.*

## 11.17 Export.h File Reference

### Macros

- #define LLGL_EXPORT

### 11.17.1 Macro Definition Documentation

#### 11.17.1.1 #define LLGL_EXPORT

## 11.18 Fence.h File Reference

```
#include "RenderSystemChild.h"
```

### Classes

- class LLGL::Fence

  *Fence interface for CPU/GPU synchronization.*

### Namespaces

- LLGL

## 11.19 Format.h File Reference

```
#include "Export.h"
#include <cstdint>
```

### Classes

- struct LLGL::FormatDescriptor

  *Vertex and texture format descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::Format {
  LLGL::Format::Undefined, LLGL::Format::A8UNorm, LLGL::Format::R8UNorm, LLGL::Format::R8SNorm, LLGL::Format::R8UInt, LLGL::Format::R8SInt, LLGL::Format::R16UNorm, LLGL::Format::R16SNorm, LLGL::Format::R16UInt, LLGL::Format::R16SInt, LLGL::Format::R16Float, LLGL::Format::R32UInt, LLGL::Format::R32SInt, LLGL::Format::R32Float, LLGL::Format::R64Float, LLGL::Format::RG8UNorm, LLGL::Format::RG8SNorm, LLGL::Format::RG8UInt, LLGL::Format::RG8SInt, LLGL::Format::RG16UNorm, LLGL::Format::RG16SNorm, LLGL::Format::RG16UInt, LLGL::Format::RG16SInt, LLGL::Format::RG16↩Float,
  LLGL::Format::RG32UInt, LLGL::Format::RG32SInt, LLGL::Format::RG32Float, LLGL::Format::RG64Float, LLGL::Format::RGB8UNorm, LLGL::Format::RGB8UNorm_sRGB, LLGL::Format::RGB8SNorm, LLGL::↩Format::RGB8UInt,
  LLGL::Format::RGB8SInt, LLGL::Format::RGB16UNorm, LLGL::Format::RGB16SNorm, LLGL::Format::R↩GB16UInt,
  LLGL::Format::RGB16SInt, LLGL::Format::RGB16Float, LLGL::Format::RGB32UInt, LLGL::Format::RG↩B32SInt,
  LLGL::Format::RGB32Float, LLGL::Format::RGB64Float, LLGL::Format::RGBA8UNorm, LLGL::Format::R↩GBA8UNorm_sRGB,
  LLGL::Format::RGBA8SNorm, LLGL::Format::RGBA8UInt, LLGL::Format::RGBA8SInt, LLGL::Format::R↩GBA16UNorm,
  LLGL::Format::RGBA16SNorm, LLGL::Format::RGBA16UInt, LLGL::Format::RGBA16SInt, LLGL::Format↩::RGBA16Float,
  LLGL::Format::RGBA32UInt, LLGL::Format::RGBA32SInt, LLGL::Format::RGBA32Float, LLGL::Format::↩RGBA64Float,
  LLGL::Format::BGRA8UNorm, LLGL::Format::BGRA8UNorm_sRGB, LLGL::Format::BGRA8SNorm, LLG↩L::Format::BGRA8UInt,
  LLGL::Format::BGRA8SInt, LLGL::Format::RGB10A2UNorm, LLGL::Format::RGB10A2UInt, LLGL::↩Format::RG11B10Float,
  LLGL::Format::RGB9E5Float, LLGL::Format::D16UNorm, LLGL::Format::D24UNormS8UInt, LLGL::↩Format::D32Float,
  LLGL::Format::D32FloatS8X24UInt, LLGL::Format::BC1RGB, LLGL::Format::BC1RGBA, LLGL::Format::↩BC2RGBA,
  LLGL::Format::BC3RGBA }

*Texture and vertex attribute format enumeration used for GPU side operations.*

- enum LLGL::ImageFormat {
  LLGL::ImageFormat::Alpha, LLGL::ImageFormat::R, LLGL::ImageFormat::RG, LLGL::ImageFormat::RGB, LLGL::ImageFormat::BGR, LLGL::ImageFormat::RGBA, LLGL::ImageFormat::BGRA, LLGL::ImageFormat↩ ::ARGB,
  LLGL::ImageFormat::ABGR, LLGL::ImageFormat::Depth, LLGL::ImageFormat::DepthStencil, LLGL::↩ ImageFormat::CompressedRGB,
  LLGL::ImageFormat::CompressedRGBA }

  *Image format enumeration that applies to each pixel of an image.*
- enum LLGL::DataType {
  LLGL::DataType::Undefined, LLGL::DataType::Int8, LLGL::DataType::UInt8, LLGL::DataType::Int16, LLGL::DataType::UInt16, LLGL::DataType::Int32, LLGL::DataType::UInt32, LLGL::DataType::Float16, LLGL::DataType::Float32, LLGL::DataType::Float64 }

  *Data types enumeration used for CPU side operations.*

### Functions

- LLGL_EXPORT const FormatDescriptor & LLGL::GetFormatDesc (const Format format)

  *Returns the descriptor for the specified hardware format or a zero-initialized descriptor in case of an invalid argument.*
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const Format format)

  *Returns true if the specified hardware format is a compressed format, i.e. either Format::BC1RGB, Format::BC1R↩ GBA, Format::BC2RGBA, or Format::BC3RGBA.*
- LLGL_EXPORT bool LLGL::IsDepthStencilFormat (const Format format)

  *Returns true if the specified hardware format is a depth or depth-stencil format, i.e. Format::D16UNorm, Format::↩ D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsDepthFormat (const Format format)

  *Returns true if the specified hardware format is a depth format, i.e. Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, or Format::D32FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsStencilFormat (const Format format)

  *Returns true if the specified hardware format is a stencil format, i.e. Format::D24UNormS8UInt or Format::D32↩ FloatS8X24UInt.*
- LLGL_EXPORT bool LLGL::IsNormalizedFormat (const Format format)

  *Returns true if the specified hardware format is a normalized format (like Format::RGBA8UNorm, Format::R8SNorm etc.).*
- LLGL_EXPORT bool LLGL::IsIntegralFormat (const Format format)

  *Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::R8SInt etc.).*
- LLGL_EXPORT bool LLGL::IsFloatFormat (const Format format)

  *Returns true if the specified hardware format is a floating-point format (like Format::RGBA32Float, Format::R32Float etc.).*
- LLGL_EXPORT std::uint32_t LLGL::DataTypeSize (const DataType dataType)

  *Returns the size (in bytes) of the specified data type.*
- LLGL_EXPORT bool LLGL::IsIntDataType (const DataType dataType)

  *Determines if the argument refers to a signed integer data type.*
- LLGL_EXPORT bool LLGL::IsUIntDataType (const DataType dataType)

  *Determines if the argument refers to an unsigned integer data type.*
- LLGL_EXPORT bool LLGL::IsFloatDataType (const DataType dataType)

  *Determines if the argument refers to a floating-pointer data type.*

## 11.20 ForwardDecls.h File Reference

### Namespaces

- LLGL

## 11.21 FragmentAttribute.h File Reference

```
#include "Format.h"
#include "SystemValue.h"
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::FragmentAttribute

  *Fragment output attribute structure.*

### Namespaces

- LLGL

## 11.22 GraphicsPipeline.h File Reference

```
#include "RenderSystemChild.h"
```

### Classes

- class LLGL::GraphicsPipeline

  *Graphics pipeline interface.*

### Namespaces

- LLGL

## 11.23 GraphicsPipelineFlags.h File Reference

```
#include "Export.h"
#include "ColorRGBA.h"
#include "Types.h"
#include "ForwardDecls.h"
#include <vector>
#include <cstdint>
```

## Classes

- struct LLGL::Viewport

  *Viewport dimensions.*
- struct LLGL::Scissor

  *Scissor dimensions.*
- 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::DepthBiasDescriptor

  *Depth bias descriptor structure to control fragment depth values.*
- 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::Primitive↩ Topology::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::NeverPass, LLGL::CompareOp::Less, LLGL::CompareOp::Equal, LLGL::CompareOp::LessEqual, LLGL::CompareOp::Greater, LLGL::CompareOp::NotEqual, LLGL::CompareOp::GreaterEqual, LLGL::CompareOp::AlwaysPass }

  *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::DstColor, LLGL::BlendOp::InvDstColor, LLGL::BlendOp::DstAlpha, LLGL::BlendOp::InvDstAlpha, 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.*

- enum LLGL::LogicOp { LLGL::LogicOp::Disabled, LLGL::LogicOp::Clear, LLGL::LogicOp::Set, LLGL::LogicOp::Copy, LLGL::LogicOp::CopyInverted, LLGL::LogicOp::NoOp, LLGL::LogicOp::Invert, LLGL::LogicOp::AND, LLGL::LogicOp::ANDReverse, LLGL::LogicOp::ANDInverted, LLGL::LogicOp::NAND, LLGL::LogicOp::OR, LLGL::LogicOp::ORReverse, LLGL::LogicOp::ORInverted, LLGL::LogicOp::NOR, LLGL::LogicOp::XOR, LLGL::LogicOp::Equiv }

  *Logical pixel operation enumeration.*

## Functions

- LLGL_EXPORT bool LLGL::IsPrimitiveTopologyPatches (const PrimitiveTopology primitiveTopology)

  *Returns true if the specified primitive topology is a patch list.*

- LLGL_EXPORT std::uint32_t LLGL::GetPrimitiveTopologyPatchSize (const PrimitiveTopology primitiveTopology)

  *Returns the number of patch control points of the specified primitive topology (in range [1, 32]), or 0 if the topology is not a patch list.*

## 11.24 Image.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "ImageFlags.h"
#include "SamplerFlags.h"
```

### Classes

- class LLGL::Image

  *Utility class to manage the storage and attributes of an image.*

### Namespaces

- LLGL

## 11.25 ImageFlags.h File Reference

```
#include "Export.h"
#include "Format.h"
#include "RenderSystemFlags.h"
#include "TextureFlags.h"
#include "ColorRGBA.h"
#include <memory>
#include <cstdint>
```

### Classes

- struct LLGL::SrcImageDescriptor

  *Descriptor structure for an image that is used as source for reading the image data.*
- struct LLGL::DstImageDescriptor

  *Descriptor structure for an image that is used as destination for writing the image data.*

### Namespaces

- LLGL

### Typedefs

- using LLGL::ByteBuffer = std::unique_ptr< char[ ]>

  *Common byte buffer type.*

### Functions

- LLGL_EXPORT std::uint32_t LLGL::ImageFormatSize (const ImageFormat imageFormat)

  *Returns the size (in number of components) of the specified image format.*
- LLGL_EXPORT std::uint32_t LLGL::ImageDataSize (const ImageFormat imageFormat, const DataType dataType, std::uint32_t numPixels)

  *Returns the required data size (in bytes) of an image with the specified format, data type, and number of pixels.*
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const ImageFormat imageFormat)

  *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 imageFormat)

  *Returns true if the specified color format is a depth-stencil format, i.e. either ImageFormat::Depth or ImageFormat↩ ::DepthStencil.*
- LLGL_EXPORT bool LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const Dst↩ ImageDescriptor &dstImageDesc, std::size_t threadCount=0)

  *Converts the image format and data type of the source image (only uncompressed color formats).*
- LLGL_EXPORT ByteBuffer LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, Image↩ Format dstFormat, DataType dstDataType, std::size_t threadCount=0)

  *Converst the image format and data type of the source image (only uncompressed color formats) and returns the new generated image buffer.*
- LLGL_EXPORT ByteBuffer LLGL::GenerateImageBuffer (ImageFormat format, DataType dataType, std↩ ::size_t imageSize, const ColorRGBAd &fillColor)

  *Generates an image buffer with the specified fill data for each pixel.*
- LLGL_EXPORT ByteBuffer LLGL::GenerateEmptyByteBuffer (std::size_t bufferSize, bool initialize=true)

  *Generates a new byte buffer with zeros in each byte.*

## 11.26 IndirectArguments.h File Reference

```
#include <cstdint>
```

### Classes

- struct LLGL::DrawIndirectArguments

  *Format structure for the arguments of an indirect draw command.*
- struct LLGL::DrawIndexedIndirectArguments

  *Format structure for the arguments of an indirect indexed draw command.*
- struct LLGL::DrawPatchIndirectArgument

  *Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer.*
- struct LLGL::DispatchIndirectArguments

  *Format structure for the arguments of an indirect compute command.*

### Namespaces

- LLGL

## 11.27 Input.h File Reference

```
#include <LLGL/Window.h>
#include <LLGL/Types.h>
#include <array>
#include <string>
```

### Classes

- class LLGL::Input

  *Default window event listener to receive user input.*

### Namespaces

- LLGL

## 11.28 Interface.h File Reference

```
#include "NonCopyable.h"
#include "InterfaceID.h"
```

### Classes

- class LLGL::Interface

  *Base class for all interfaces in LLGL.*

### Namespaces

- LLGL

### Macros

- #define LLGL_DECLARE_INTERFACE(ID)

  *Declares the base functions for all LLGL interfaces.*
- #define LLGL_IMPLEMENT_INTERFACE(SELF, BASE)

  *Implements the base functions for all LLGL interfaces.*

### 11.28.1 Macro Definition Documentation

#### 11.28.1.1 #define LLGL_DECLARE_INTERFACE( *ID* )

**Value:**

```
public:                                         \
        inline static int GetInterfaceID()          \
        {                                           \
            return (ID);                            \
        }                                           \
        bool IsInstanceOf(int id) const override
```

Declares the base functions for all LLGL interfaces.

#### 11.28.1.2 #define LLGL_IMPLEMENT_INTERFACE( *SELF, BASE* )

**Value:**

```
bool SELF::IsInstanceOf(int id) const                                   \
    {                                                                       \
        return (id == SELF::GetInterfaceID() || BASE::IsInstanceOf(id));    \
    }
```

Implements the base functions for all LLGL interfaces.

## 11.29 InterfaceID.h File Reference

### Classes

- struct LLGL::InterfaceID

  *Enumeration of all interfaces in LLGL.*

### Namespaces

- LLGL

## 11.30 IOSNativeHandle.h File Reference

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

### Classes

- struct LLGL::NativeHandle

  *iOS native handle structure.*
- struct LLGL::NativeContextHandle

  *iOS native context handle structure.*

### Namespaces

- LLGL

## 11.31 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,
  LLGL::Key::Any }

  *Input key codes.*

## 11.32 LinuxNativeHandle.h File Reference

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

### Classes

- struct LLGL::NativeHandle

  *iOS native handle structure.*
- struct LLGL::NativeContextHandle

  *iOS native context handle structure.*

### Namespaces

- LLGL

## 11.33 LLGL.h File Reference

```
#include "Version.h"
#include "Window.h"
#include "Canvas.h"
#include "Display.h"
#include "Input.h"
#include "Timer.h"
#include "TypeInfo.h"
#include "ColorRGB.h"
#include "ColorRGBA.h"
#include "RenderSystem.h"
#include "Log.h"
#include "IndirectArguments.h"
#include "ImageFlags.h"
```

## 11.34 Log.h File Reference

```
#include "Export.h"
#include <functional>
#include <string>
#include <iostream>
```

### Namespaces

- LLGL
- LLGL::Log

### Typedefs

- using LLGL::Log::ReportCallback = std::function< void(ReportType type, const std::string &message, const std::string &contextInfo, void *userData)>

  *Report callback function signature.*

### Enumerations

- enum LLGL::Log::ReportType { LLGL::Log::ReportType::Error, LLGL::Log::ReportType::Warning, LLGL::Log::ReportType::Information, LLGL::Log::ReportType::Performance }

  *Report type enumeration.*

### Functions

- LLGL_EXPORT void LLGL::Log::PostReport (ReportType type, const std::string &message, const std::string &contextInfo="")
- LLGL_EXPORT void LLGL::Log::SetReportCallback (const ReportCallback &callback, void *userData=nullptr)

  *Sets the new report callback. No report callback is specified by default, in which case the reports are ignored.*
- LLGL_EXPORT void LLGL::Log::SetReportCallbackStd (std::ostream &stream=std::cerr)

  *Sets the new report callback to the standard output streams.*
- LLGL_EXPORT void LLGL::Log::SetReportLimit (std::size_t maxCount)

  *Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.*

## 11.35 MacOSNativeHandle.h File Reference

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

### Classes

- struct LLGL::NativeHandle

  *iOS native handle structure.*
- struct LLGL::NativeContextHandle

  *iOS native context handle structure.*

### Namespaces

- LLGL

## 11.36 NativeHandle.h File Reference

```
#include "Platform.h"
```

## 11.37 NonCopyable.h File Reference

```
#include "Export.h"
```

### Classes

- class LLGL::NonCopyable

  *Base class for all classes with virtual function tables in LLGL.*

### Namespaces

- LLGL

## 11.38 PipelineLayout.h File Reference

```
#include "RenderSystemChild.h"
```

### Classes

- class LLGL::PipelineLayout

  *Pipeline layout interface.*

### Namespaces

- LLGL

## 11.39 PipelineLayoutFlags.h File Reference

```
#include "Export.h"
#include "ResourceFlags.h"
#include "BufferFlags.h"
#include "ShaderFlags.h"
#include <vector>
```

### Classes

- struct LLGL::BindingDescriptor

  *Layout structure for a single binding point of the pipeline layout descriptor.*
- struct LLGL::PipelineLayoutDescriptor

  *Pipeline layout descritpor structure.*

### Namespaces

- LLGL

## 11.40 Platform.h File Reference

### Macros

- #define LLGL_API
- #define LLGL_API_CDECL
- #define LLGL_API_STDCALL
- #define LLGL_API_THISCALL

### 11.40.1 Macro Definition Documentation

#### 11.40.1.1 #define LLGL_API

#### 11.40.1.2 #define LLGL_API_CDECL

#### 11.40.1.3 #define LLGL_API_STDCALL

#### 11.40.1.4 #define LLGL_API_THISCALL

## 11.41 QueryHeap.h File Reference

```
#include "RenderSystemChild.h"
#include "QueryHeapFlags.h"
```

### Classes

- class LLGL::QueryHeap

  *Query heap interface that holds a certain number of queries that are all of the same type.*

### Namespaces

- LLGL

## 11.42 QueryHeapFlags.h File Reference

```
#include "Constants.h"
#include <cstdint>
```

### Classes

- struct LLGL::QueryPipelineStatistics

  *Query data structure for pipeline statistics.*
- struct LLGL::QueryHeapDescriptor

  *Query heap descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::QueryType { LLGL::QueryType::SamplesPassed, LLGL::QueryType::AnySamplesPassed, LLGL::QueryType::AnySamplesPassedConservative, LLGL::QueryType::TimeElapsed, LLGL::QueryType::StreamOutPrimitivesWritten, LLGL::QueryType::StreamOutOverflow, LLGL::QueryType::PipelineStatistics }

  *Query type enumeration.*

## 11.43 RenderContext.h File Reference

```
#include "RenderTarget.h"
#include "RenderContextFlags.h"
#include "RenderSystemFlags.h"
#include "Surface.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "ShaderProgram.h"
#include "Texture.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "Sampler.h"
#include "QueryHeap.h"
#include <string>
#include <memory>
#include <map>
```

### Classes

- class LLGL::RenderContext

  *Render context interface.*

### Namespaces

- LLGL

## 11.44 RenderContextFlags.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "GraphicsPipelineFlags.h"
#include <cstdint>
```

### Classes

- struct LLGL::VsyncDescriptor

  *Vertical-synchronization (Vsync) descriptor structure.*
- struct LLGL::VideoModeDescriptor

  *Video mode descriptor structure.*
- struct LLGL::RenderContextDescriptor

  *Render context descriptor structure.*

### Namespaces

- LLGL

### Functions

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

  *Compares the two specified V-sync descriptors on equality.*
- LLGL_EXPORT bool LLGL::operator!= (const VsyncDescriptor &lhs, const VsyncDescriptor &rhs)

  *Compares the two specified V-sync descriptors on inequality.*
- LLGL_EXPORT bool LLGL::operator== (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)

  *Compares the two specified video mode descriptors on equality.*
- LLGL_EXPORT bool LLGL::operator!= (const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs)

  *Compares the two specified video mode descriptors on inequality.*

## 11.45 RendererConfiguration.h File Reference

```
#include <string>
#include <vector>
#include <cstdint>
```

### Classes

- struct LLGL::ApplicationDescriptor

  *Application descriptor structure.*
- struct LLGL::RendererConfigurationVulkan

  *Structure for a Vulkan renderer specific configuration.*
- struct LLGL::RendererConfigurationOpenGL

  *OpenGL profile descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::OpenGLContextProfile { LLGL::OpenGLContextProfile::CompatibilityProfile, LLGL::OpenGL↩ ContextProfile::CoreProfile, LLGL::OpenGLContextProfile::ESProfile }

  *OpenGL context profile enumeration.*

## 11.46 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, LLGL::ErrorType::UndefinedBehavior }

  *Rendering debugger error types enumeration.*
- enum LLGL::WarningType { LLGL::WarningType::ImproperArgument, LLGL::WarningType::ImproperState, LLGL::WarningType::PointlessOperation, LLGL::WarningType::VaryingBehavior }

  *Rendering debugger warning types enumeration.*

## 11.47 RenderingProfiler.h File Reference

```
#include "Export.h"
#include "RenderContextFlags.h"
#include "GraphicsPipelineFlags.h"
#include <cstdint>
#include <algorithm>
```

### Classes

- struct LLGL::FrameProfile

  *Profile of a rendered frame.*
- class LLGL::RenderingProfiler

  *Rendering profiler model class.*

### Namespaces

- LLGL

## 11.48 RenderPass.h File Reference

```
#include "RenderSystemChild.h"
```

### Classes

- class LLGL::RenderPass

  *Render pass interface.*

### Namespaces

- LLGL

## 11.49 RenderPassFlags.h File Reference

```
#include "Format.h"
#include <vector>
```

### Classes

- struct LLGL::AttachmentFormatDescriptor

  *Render target attachment descriptor structure.*
- struct LLGL::RenderPassDescriptor

  *Render pass descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::AttachmentLoadOp { LLGL::AttachmentLoadOp::Undefined, LLGL::AttachmentLoadOp::Load, LLGL::AttachmentLoadOp::Clear }

  *Enumeration for render pass attachment load operations.*
- enum LLGL::AttachmentStoreOp { LLGL::AttachmentStoreOp::Undefined, LLGL::AttachmentStoreOp::Store }

  *Enumeration for render pass attachment store operations.*

## 11.50 RenderSystem.h File Reference

```
#include "Interface.h"
#include "RenderContext.h"
#include "CommandQueue.h"
#include "CommandBuffer.h"
#include "RenderSystemFlags.h"
#include "RenderingProfiler.h"
#include "RenderingDebugger.h"
#include "Buffer.h"
#include "BufferFlags.h"
#include "BufferArray.h"
#include "Texture.h"
#include "TextureFlags.h"
#include "Sampler.h"
#include "SamplerFlags.h"
#include "ResourceHeap.h"
#include "ResourceHeapFlags.h"
#include "RenderPass.h"
#include "RenderPassFlags.h"
#include "RenderTarget.h"
#include "RenderTargetFlags.h"
#include "Shader.h"
#include "ShaderFlags.h"
#include "ShaderProgram.h"
#include "ShaderProgramFlags.h"
#include "PipelineLayout.h"
#include "PipelineLayoutFlags.h"
#include "GraphicsPipeline.h"
#include "GraphicsPipelineFlags.h"
#include "ComputePipeline.h"
#include "ComputePipelineFlags.h"
#include "QueryHeap.h"
#include "Fence.h"
#include <string>
#include <memory>
#include <vector>
#include <cstdint>
```

### Classes

- class LLGL::RenderSystem

  *Render system interface.*

### Namespaces

- LLGL

## 11.51 RenderSystemChild.h File Reference

```
#include "Interface.h"
```

### Classes

- class LLGL::RenderSystemChild

  *Base class for all interfaces whoes instances are owned by the RenderSystem.*

### Namespaces

- LLGL

## 11.52 RenderSystemFlags.h File Reference

```
#include "Export.h"
#include "CommandBufferFlags.h"
#include "TextureFlags.h"
#include "Constants.h"
#include "RendererConfiguration.h"
#include <cstddef>
#include <cstdint>
#include <string>
#include <vector>
#include <functional>
```

### Classes

- struct LLGL::RenderSystemConfiguration

  *Render system configuration structure.*
- struct LLGL::RendererID

  *Renderer identification number enumeration.*
- struct LLGL::RendererInfo

  *Renderer basic information structure.*
- struct LLGL::RenderSystemDescriptor

  *Render system descriptor structure.*
- struct LLGL::RenderingFeatures

  *Contains the attributes for all supported rendering features.*
- struct LLGL::RenderingLimits

  *Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.*
- struct LLGL::RenderingCapabilities

  *Structure with all attributes describing the rendering capabilities of the render system.*

## Namespaces

- LLGL

## Typedefs

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

  *Debug callback function interface.*
- using LLGL::ValidateRenderingCapsFunc = std::function< bool(const std::string &info, const std::string &attrib)>

  *Callback interface for the ValidateRenderingCaps function.*

## Enumerations

- enum LLGL::ShadingLanguage {
  LLGL::ShadingLanguage::GLSL = (0x10000), LLGL::ShadingLanguage::GLSL_110 = (0x10000 | 110), LLGL::ShadingLanguage::GLSL_120 = (0x10000 | 120), LLGL::ShadingLanguage::GLSL_130 = (0x10000 | 130),
  LLGL::ShadingLanguage::GLSL_140 = (0x10000 | 140), LLGL::ShadingLanguage::GLSL_150 = (0x10000 | 150), LLGL::ShadingLanguage::GLSL_330 = (0x10000 | 330), LLGL::ShadingLanguage::GLSL_400 = (0x10000 | 400),
  LLGL::ShadingLanguage::GLSL_410 = (0x10000 | 410), LLGL::ShadingLanguage::GLSL_420 = (0x10000 | 420), LLGL::ShadingLanguage::GLSL_430 = (0x10000 | 430), LLGL::ShadingLanguage::GLSL_440 = (0x10000 | 440),
  LLGL::ShadingLanguage::GLSL_450 = (0x10000 | 450), LLGL::ShadingLanguage::GLSL_460 = (0x10000 | 460), LLGL::ShadingLanguage::ESSL = (0x20000), LLGL::ShadingLanguage::ESSL_100 = (0x20000 | 100), LLGL::ShadingLanguage::ESSL_300 = (0x20000 | 300), LLGL::ShadingLanguage::ESSL_310 = (0x20000 | 310), LLGL::ShadingLanguage::ESSL_320 = (0x20000 | 320), LLGL::ShadingLanguage::HLSL = (0x30000), LLGL::ShadingLanguage::HLSL_2_0 = (0x30000 | 200), LLGL::ShadingLanguage::HLSL_2_0a = (0x30000 | 201), LLGL::ShadingLanguage::HLSL_2_0b = (0x30000 | 202), LLGL::ShadingLanguage::HLSL_3_0 = (0x30000 | 300),
  LLGL::ShadingLanguage::HLSL_4_0 = (0x30000 | 400), LLGL::ShadingLanguage::HLSL_4_1 = (0x30000 | 410), LLGL::ShadingLanguage::HLSL_5_0 = (0x30000 | 500), LLGL::ShadingLanguage::HLSL_5_1 = (0x30000 | 510),
  LLGL::ShadingLanguage::Metal = (0x40000), LLGL::ShadingLanguage::Metal_1_0 = (0x40000 | 100), LLGL::ShadingLanguage::Metal_1_1 = (0x40000 | 110), LLGL::ShadingLanguage::Metal_1_2 = (0x40000 | 120),
  LLGL::ShadingLanguage::Metal_2_0 = (0x40000 | 200), LLGL::ShadingLanguage::Metal_2_1 = (0x40000 | 210), LLGL::ShadingLanguage::SPIRV = (0x50000), LLGL::ShadingLanguage::SPIRV_100 = (0x50000 | 100),
  LLGL::ShadingLanguage::VersionBitmask = 0x0000ffff }

  *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.*
- enum LLGL::CPUAccess { LLGL::CPUAccess::ReadOnly, LLGL::CPUAccess::WriteOnly, LLGL::CPUAccess::WriteDiscard, LLGL::CPUAccess::ReadWrite }

  *Classifications of CPU access to mapped resources.*

## Functions

- LLGL_EXPORT bool LLGL::ValidateRenderingCaps (const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={})

  *Validates the presence of the specified required rendering capabilities.*

## 11.53 RenderTarget.h File Reference

```
#include "RenderSystemChild.h"
#include "RenderTargetFlags.h"
#include "Types.h"
```

### Classes

- class LLGL::RenderTarget

  *Render target interface.*

### Namespaces

- LLGL

## 11.54 RenderTargetFlags.h File Reference

```
#include "TextureFlags.h"
#include "ForwardDecls.h"
#include "GraphicsPipelineFlags.h"
#include <vector>
#include <cstdint>
```

### Classes

- struct LLGL::AttachmentDescriptor

  *Render target attachment descriptor structure.*
- struct LLGL::RenderTargetDescriptor

  *Render target descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::AttachmentType { LLGL::AttachmentType::Color, LLGL::AttachmentType::Depth, LLGL::↩ AttachmentType::DepthStencil, LLGL::AttachmentType::Stencil }

  *Render target attachment type enumeration.*

## 11.55 Resource.h File Reference

```
#include "RenderSystemChild.h"
#include "ResourceFlags.h"
```

### Classes

- class LLGL::Resource

  *Base class for all hardware resource interfaces.*

### Namespaces

- LLGL

## 11.56 ResourceFlags.h File Reference

### Classes

- struct LLGL::BindFlags

  *Flags for Buffer and Texture resources that describe for which purposes they will be used.*
- struct LLGL::CPUAccessFlags

  *CPU read/write access flag enumeration.*
- struct LLGL::MiscFlags

  *Miscellaneous resource flags.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::ResourceType { LLGL::ResourceType::Undefined, LLGL::ResourceType::Buffer, LLGL::↩ ResourceType::Texture, LLGL::ResourceType::Sampler }

  *Hardware resource type enumeration.*

## 11.57 ResourceHeap.h File Reference

```
#include "RenderSystemChild.h"
#include "ResourceHeapFlags.h"
```

### Classes

- class LLGL::ResourceHeap

  *Resource heap interface.*

### Namespaces

- LLGL

## 11.58 ResourceHeapFlags.h File Reference

```
#include "Export.h"
#include "Texture.h"
#include "TextureFlags.h"
#include "Buffer.h"
#include "BufferFlags.h"
#include <vector>
```

### Classes

- struct LLGL::ResourceViewDescriptor

  *Resource view descriptor structure.*
- struct LLGL::ResourceHeapDescriptor

  *Resource heap descriptor structure.*

### Namespaces

- LLGL

## 11.59 Sampler.h File Reference

```
#include "Resource.h"
#include "SamplerFlags.h"
```

### Classes

- class LLGL::Sampler

  *Sampler interface.*

### Namespaces

- LLGL

## 11.60 SamplerFlags.h File Reference

```
#include "Export.h"
#include "GraphicsPipelineFlags.h"
#include "ColorRGBA.h"
#include <cstddef>
#include <cstdint>
```

### Classes

- struct LLGL::SamplerDescriptor

  *Texture sampler descriptor structure.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::SamplerAddressMode { LLGL::SamplerAddressMode::Repeat, LLGL::SamplerAddressMode::Mirror, LLGL::SamplerAddressMode↩ ::Clamp, LLGL::SamplerAddressMode::Border, LLGL::SamplerAddressMode::MirrorOnce }

  *Technique for resolving texture coordinates that are outside of the range [0, 1].*
- enum LLGL::SamplerFilter { LLGL::SamplerFilter::Nearest, LLGL::SamplerFilter::Linear }

  *Sampling filter enumeration.*

## 11.61 Shader.h File Reference

```
#include "RenderSystemChild.h"
#include "ShaderFlags.h"
```

### Classes

- class LLGL::Shader

  *Shader interface.*

### Namespaces

- LLGL

## 11.62 ShaderFlags.h File Reference

```
#include "Export.h"
#include "StreamOutputFormat.h"
#include <cstddef>
```

### Classes

- struct LLGL::ShaderCompileFlags

  *Shader compilation flags enumeration.*
- struct LLGL::StageFlags

  *Shader stage flags enumeration.*
- struct LLGL::ShaderMacro

  *Shader macro structure with name and optional body definition.*
- struct LLGL::ShaderDescriptor

  *Shader source and binary code descriptor structure.*
- struct LLGL::ShaderDescriptor::StreamOutput

  *Additional descriptor for stream outputs.*

### Namespaces

- LLGL

### Enumerations

- enum LLGL::ShaderType { LLGL::ShaderType::Undefined, LLGL::ShaderType::Vertex, LLGL::ShaderType::TessControl, LLGL::ShaderType::TessEvaluation, LLGL::ShaderType::Geometry, LLGL::ShaderType::Fragment, LLGL::ShaderType::Compute }

  *Shader type enumeration.*
- enum LLGL::ShaderSourceType { LLGL::ShaderSourceType::CodeString, LLGL::ShaderSourceType::CodeFile, LLGL::ShaderSourceType::BinaryBuffer, LLGL::ShaderSourceType::BinaryFile }

  *Shader source type enumeration.*

### Functions

- LLGL_EXPORT bool LLGL::IsShaderSourceCode (const ShaderSourceType type)

  *Returns true if the specified shader source type is either ShaderSourceType::CodeString or ShaderSourceType::CodeFile.*
- LLGL_EXPORT bool LLGL::IsShaderSourceBinary (const ShaderSourceType type)

  *Returns true if the specified shader source type is either ShaderSourceType::BinaryBuffer or ShaderSourceType::BinaryFile.*

## 11.63 ShaderProgram.h File Reference

```
#include "RenderSystemChild.h"
#include "ShaderProgramFlags.h"
```

### Classes

- class LLGL::ShaderProgram

  *Shader program interface.*

### Namespaces

- LLGL

## 11.64 ShaderProgramFlags.h File Reference

```
#include "ForwardDecls.h"
#include "VertexFormat.h"
#include "StreamOutputFormat.h"
#include "ResourceFlags.h"
#include "BufferFlags.h"
#include "PipelineLayoutFlags.h"
#include <vector>
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::ShaderProgramDescriptor

  *Descriptor structure for shader programs.*
- struct LLGL::ShaderResource

  *Shader reflection resource structure.*
- struct LLGL::ShaderUniform

  *Shader reflection uniform structure.*
- struct LLGL::ShaderReflection

  *Shader reflection structure.*

### Namespaces

- LLGL

### Typedefs

- using LLGL::UniformLocation = std::int32_t

  *Shader uniform location type, as zero-based index in 32-bit signed integer format.*

### Enumerations

- enum LLGL::UniformType {
  LLGL::UniformType::Undefined, LLGL::UniformType::Float1, LLGL::UniformType::Float2, LLGL::Uniform↩Type::Float3,
  LLGL::UniformType::Float4, LLGL::UniformType::Double1, LLGL::UniformType::Double2, LLGL::Uniform↩Type::Double3,
  LLGL::UniformType::Double4, LLGL::UniformType::Int1, LLGL::UniformType::Int2, LLGL::UniformType::Int3,
  LLGL::UniformType::Int4, LLGL::UniformType::UInt1, LLGL::UniformType::UInt2, LLGL::UniformType::UInt3,
  LLGL::UniformType::UInt4, LLGL::UniformType::Bool1, LLGL::UniformType::Bool2, LLGL::UniformType::↩Bool3,
  LLGL::UniformType::Bool4, LLGL::UniformType::Float2x2, LLGL::UniformType::Float2x3, LLGL::Uniform↩Type::Float2x4,
  LLGL::UniformType::Float3x2, LLGL::UniformType::Float3x3, LLGL::UniformType::Float3x4, LLGL::↩UniformType::Float4x2,
  LLGL::UniformType::Float4x3, LLGL::UniformType::Float4x4, LLGL::UniformType::Double2x2, LLGL::↩UniformType::Double2x3,
  LLGL::UniformType::Double2x4, LLGL::UniformType::Double3x2, LLGL::UniformType::Double3x3, LLGL::↩UniformType::Double3x4,
  LLGL::UniformType::Double4x2, LLGL::UniformType::Double4x3, LLGL::UniformType::Double4x4, LLGL::↩UniformType::Sampler,
  LLGL::UniformType::Image, LLGL::UniformType::AtomicCounter }

  *Shader uniform type enumeration.*

## 11.65 StreamOutputAttribute.h File Reference

```
#include "Export.h"
#include "Format.h"
#include <string>
#include <cstdint>
```

### 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)

## 11.66 StreamOutputFormat.h File Reference

```
#include "Export.h"
#include "StreamOutputAttribute.h"
#include <vector>
#include <cstdint>
```

### Classes

- struct LLGL::StreamOutputFormat

  *Stream-output format descriptor structure.*

### Namespaces

- LLGL

## 11.67 Strings.h File Reference

```
#include "Export.h"
#include "ShaderFlags.h"
#include "TextureFlags.h"
#include "RenderingDebugger.h"
#include "RenderSystemFlags.h"
```

### Namespaces

- LLGL

### Functions

- LLGL_EXPORT const char * LLGL::ToString (const ShaderType t)

  *Returns a string representation for the spcified ShaderType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const ErrorType t)

  *Returns a string representation for the specified ErrorType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const WarningType t)

  *Returns a string representation for the specified WarningType value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const ShadingLanguage t)

  *Returns a string representation for the specified ShadingLanguage value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const Format t)

  *Returns a string representation for the specified Format value, or null if the input type is invalid.*
- LLGL_EXPORT const char * LLGL::ToString (const TextureType t)

  *Returns a string representation for the specified TextureType value, or null if the input type is invalid.*

## 11.68 Surface.h File Reference

```
#include "Interface.h"
#include "Types.h"
#include "RenderContextFlags.h"
```

### Classes

- class LLGL::Surface

  *The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms).*

### Namespaces

- LLGL

## 11.69 SystemValue.h File Reference

### Namespaces

- LLGL

### Enumerations

- enum LLGL::SystemValue {
  LLGL::SystemValue::Undefined, LLGL::SystemValue::ClipDistance, LLGL::SystemValue::Color, LLGL::↩
  SystemValue::CullDistance,
  LLGL::SystemValue::Depth, LLGL::SystemValue::DepthGreater, LLGL::SystemValue::DepthLess, LLGL::↩
  SystemValue::FrontFacing,
  LLGL::SystemValue::InstanceID, LLGL::SystemValue::Position, LLGL::SystemValue::PrimitiveID, LLGL::↩
  SystemValue::RenderTargetIndex,
  LLGL::SystemValue::SampleMask, LLGL::SystemValue::SampleID, LLGL::SystemValue::Stencil, LLGL::↩
  SystemValue::VertexID,
  LLGL::SystemValue::ViewportIndex }

  *Enumeration of system values for vertex input or fragment output attributes.*

## 11.70 Tags.h File Reference

### Classes

- struct LLGL::UninitializeTag

  *Common uninitialize tag.*

### Namespaces

- LLGL

## 11.71 Texture.h File Reference

```
#include "Resource.h"
#include "Types.h"
#include "TextureFlags.h"
#include <cstdint>
```

### Classes

- class LLGL::Texture

  *Texture interface.*

### Namespaces

- LLGL

## 11.72 TextureFlags.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "Format.h"
#include "ResourceFlags.h"
#include "CommandBufferFlags.h"
#include <cstddef>
#include <cstdint>
```

### Classes

- struct LLGL::TextureSwizzleRGBA

  *Texture component swizzle structure for red, green, blue, and alpha components.*
- struct LLGL::TextureSubresource

  *Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource.*
- struct LLGL::TextureLocation

  *Texture location structure: MIP-map level and offset.*
- struct LLGL::TextureRegion

  *Texture region structure: MIP-map level, offset, and extent.*
- struct LLGL::TextureDescriptor

  *Texture descriptor structure.*
- struct LLGL::TextureViewDescriptor

  *Texture view descriptor structure.*

### 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::Texture↩
  CubeArray, LLGL::TextureType::Texture2DMS,
  LLGL::TextureType::Texture2DMSArray }

  *Texture type enumeration.*
- enum LLGL::TextureSwizzle {
  LLGL::TextureSwizzle::Zero, LLGL::TextureSwizzle::One, LLGL::TextureSwizzle::Red, LLGL::Texture↩
  Swizzle::Green,
  LLGL::TextureSwizzle::Blue, LLGL::TextureSwizzle::Alpha }

  *Texture component swizzle enumeration.*

### Functions

- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1)

  *Returns the number of MIP-map levels for a texture with the specified size.*
- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (const TextureDescriptor &textureDesc)

  *Returns the number of MIP-map levels for the specified texture descriptor.*
- LLGL_EXPORT std::uint32_t LLGL::TextureBufferSize (const Format format, std::uint32_t numTexels)

  *Returns the required buffer size (in bytes) of a texture with the specified hardware format and number of texels.*
- LLGL_EXPORT std::uint32_t LLGL::TextureSize (const TextureDescriptor &textureDesc)

  *Returns the texture size (in texels) of the specified texture descriptor, or zero if the texture type is invalid.*
- LLGL_EXPORT bool LLGL::IsMipMappedTexture (const TextureDescriptor &textureDesc)

  *Returns true if the specified texture descriptor describes a texture with MIP-mapping enabled.*
- 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.*
- LLGL_EXPORT bool LLGL::IsCubeTexture (const TextureType type)

  *Returns true if the specified texture type is a cube texture.*
- LLGL_EXPORT bool LLGL::IsTextureSwizzleIdentity (const TextureSwizzleRGBA &swizzle)

  *Returns true if the specified texture swizzling is equal to the identity mapping.*

## 11.73 Timer.h File Reference

```
#include "Interface.h"
#include <memory>
#include <cstdint>
```

### Classes

- class LLGL::Timer

  *Interface for a Timer class.*

### Namespaces

- LLGL

## 11.74 TypeInfo.h File Reference

```
#include "Interface.h"
```

### Namespaces

- LLGL

### Functions

- template<typename T >
  const T * LLGL::CastTo (const Interface *obj)

  *Returns a constant pointer to this instance of type `const T*` if it implements the specified interface.*
- template<typename T >
  T * LLGL::CastTo (Interface *obj)

  *Returns a pointer to this instance of type `T*` if it implements the specified interface.*
- template<typename T >
  const T & LLGL::CastTo (const Interface &obj)

  *Returns a constant reference to this instance of type `const T&` if it implements the specified interface.*
- template<typename T >
  T & LLGL::CastTo (Interface &obj)

  *Returns a reference to this instance of type `T&` if it implements the specified interface.*

## 11.75 Types.h File Reference

```
#include "Export.h"
#include <cstdint>
```

### Classes

- struct LLGL::Extent2D

  *2-Dimensional extent structure.*
- struct LLGL::Extent3D

  *3-Dimensional extent structure.*
- struct LLGL::Offset2D

  *2-Dimensional offset structure.*
- struct LLGL::Offset3D

  *3-Dimensional offset structure.*

### Namespaces

- LLGL

### Functions

- LLGL_EXPORT Extent2D LLGL::operator+ (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent2D LLGL::operator- (const Extent2D &lhs, const Extent2D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D LLGL::operator+ (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the sum of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Extent3D LLGL::operator- (const Extent3D &lhs, const Extent3D &rhs)

  *Returns the subtractionn of left hand side extent 'lhs' and the right hand side extent 'rhs'.*
- LLGL_EXPORT Offset2D LLGL::operator+ (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset2D LLGL::operator- (const Offset2D &lhs, const Offset2D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D LLGL::operator+ (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the sum of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- LLGL_EXPORT Offset3D LLGL::operator- (const Offset3D &lhs, const Offset3D &rhs)

  *Returns the subtractionn of left hand side offset 'lhs' and the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool LLGL::operator!= (const Offset2D &lhs, const Offset2D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is equal to the right hand side offset 'rhs'.*
- bool LLGL::operator!= (const Offset3D &lhs, const Offset3D &rhs)

  *Returns true if the left hand side offset 'lhs' is unequal to the right hand side offset 'rhs'.*
- bool LLGL::operator== (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool LLGL::operator!= (const Extent2D &lhs, const Extent2D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*
- bool LLGL::operator== (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is equal to the right hand side extent 'rhs'.*
- bool LLGL::operator!= (const Extent3D &lhs, const Extent3D &rhs)

  *Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.*

## 11.76 Utility.h File Reference

```
#include "Export.h"
#include "TextureFlags.h"
#include "BufferFlags.h"
#include "RenderTargetFlags.h"
#include "RenderPassFlags.h"
#include "ResourceHeapFlags.h"
#include "ShaderFlags.h"
#include "ShaderProgramFlags.h"
#include "PipelineLayoutFlags.h"
#include <initializer_list>
```

## Namespaces

- LLGL

## Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (Format format, std::uint32_t width, long bind↩ Flags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture1D type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2D type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture3D type.*
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (Format format, std::uint32_t width, std::uint32↩ _t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCube type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (Format format, std::uint32_t width, std↩ ::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (Format format, std::uint32_t width, std↩ ::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::↩ Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (Format format, std::uint32_t width, std↩ ::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::↩ Sampled))

  *Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (Format format, std::uint32_t width, std↩ ::uint32_t height, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.*
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (Format format, std::uint32_t width, std↩ ::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::Color↩ Attachment|BindFlags::Sampled))

  *Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.*
- LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a vertex buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t size, const Format format, long cpu↩ AccessFlags=0)

  *Returns a BufferDescriptor structure for an index buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t size, long cpuAccessFlags=0)

  *Returns a BufferDescriptor structure for a constant buffer.*
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite)

  *Returns a BufferDescriptor structure for a storage buffer.*
- LLGL_EXPORT ShaderDescriptor LLGL::ShaderDescFromFile (const ShaderType type, const char ∗filename, const char ∗entryPoint=nullptr, const char ∗profile=nullptr, long flags=0)

  *Returns a ShaderDescriptor structure.*
- LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc (const std::initializer_list< Shader ∗ > &shaders, const std::initializer_list< VertexFormat > &vertexFormats={})

*Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*

- LLGL_EXPORT ShaderProgramDescriptor LLGL::ShaderProgramDesc (const std::vector< Shader * > &shaders, const std::vector< VertexFormat > &vertexFormats={})

  *Returns a ShaderProgramDescriptor structure and assigns the input shaders into the respective structure members.*
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const ShaderReflection &reflection)

  *Converts the specified shader reflection descriptor into a pipeline layout descriptor.*
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const char *layoutSignature)

  *Generates a pipeline layout descriptor by parsing the specified string.*
- LLGL_EXPORT RenderPassDescriptor LLGL::RenderPassDesc (const RenderTargetDescriptor &renderTargetDesc)

  *Converts the specified render target descriptor into a render pass descriptor with default settings.*

## 11.77 Version.h File Reference

```
#include "Export.h"
#include <string>
#include <cstdint>
```

### Namespaces

- LLGL
- LLGL::Version

  *Namespace with functions to determine LLGL version.*

### Functions

- LLGL_EXPORT std::uint32_t LLGL::Version::GetMajor ()

  *Returns the major LLGL version (e.g. 1 stands for "1.00").*
- LLGL_EXPORT std::uint32_t LLGL::Version::GetMinor ()

  *Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.*
- LLGL_EXPORT std::uint32_t 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 std::uint32_t 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)").*

## 11.78 VertexAttribute.h File Reference

```
#include "Export.h"
#include "Format.h"
#include "SystemValue.h"
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::VertexAttribute

  *Vertex input attribute structure.*

### Namespaces

- LLGL

### Functions

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

  *Compares the two VertexAttribute types for equality (including their names and all other members).*
- LLGL_EXPORT bool LLGL::operator!= (const VertexAttribute &lhs, const VertexAttribute &rhs)

  *Compares the two VertexAttribute types for inequality (including their names and all other members).*

## 11.79 VertexFormat.h File Reference

```
#include "Export.h"
#include "Constants.h"
#include "VertexAttribute.h"
#include <vector>
#include <cstdint>
```

### Classes

- struct LLGL::VertexFormat

  *Vertex format structure.*

### Namespaces

- LLGL

## 11.80 VideoAdapter.h File Reference

```
#include "Export.h"
#include "DisplayFlags.h"
#include <vector>
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::VideoOutputDescriptor

  *Video output structure.*
- struct LLGL::VideoAdapterDescriptor

  *Video adapter descriptor structure.*

### Namespaces

- LLGL

## 11.81 Win32NativeHandle.h File Reference

```
#include <Windows.h>
```

### Classes

- struct LLGL::NativeHandle

  *iOS native handle structure.*
- struct LLGL::NativeContextHandle

  *iOS native context handle structure.*

### Namespaces

- LLGL

## 11.82 Window.h File Reference

```
#include "Surface.h"
#include "WindowFlags.h"
#include "Key.h"
#include <memory>
#include <vector>
```

### Classes

- class LLGL::Window

  *Window interface for desktop platforms.*
- class LLGL::Window::EventListener

  *Interface for all window event listeners.*

### Namespaces

- LLGL

## 11.83 WindowFlags.h File Reference

```
#include "Types.h"
#include <string>
#include <cstdint>
```

### Classes

- struct LLGL::WindowDescriptor

  *Window descriptor structure.*
- struct LLGL::WindowBehavior

  *Window behavior structure.*

### Namespaces

- LLGL

# Index


~NonCopyable
LLGL::NonCopyable, 245
~RenderingDebugger
LLGL::RenderingDebugger, 274

A
LLGL, 79
a
LLGL::Color< T, 4u >, 146
LLGL::TextureSwizzleRGBA, 372
A8UNorm
LLGL, 74
ABGR
LLGL, 77
ANDInverted
LLGL, 82
ANDReverse
LLGL, 82
AND
LLGL, 82
ARGB
LLGL, 77
acceptDropFiles
LLGL::WindowDescriptor, 400
Accumulate
LLGL::FrameProfile, 208
LLGL::RenderingProfiler, 286
AdaptForVideoMode
LLGL::Canvas, 129
LLGL::Surface, 358
LLGL::Window, 394
Add
LLGL, 69
AddEventListener
LLGL::Canvas, 129
LLGL::Window, 394
addressModeU
LLGL::SamplerDescriptor, 323
addressModeV
LLGL::SamplerDescriptor, 323
addressModeW
LLGL::SamplerDescriptor, 323
All
LLGL::ClearFlags, 132
AllGraphicsStages
LLGL::StageFlags, 349
AllStages
LLGL::StageFlags, 349
AllTessStages
LLGL::StageFlags, 349
Alpha
LLGL, 77, 93
alphaArithmetic
LLGL::BlendTargetDescriptor, 122
alphaToCoverageEnabled
LLGL::BlendDescriptor, 119
AlwaysPass
LLGL, 70
antiAliasedLineEnabled
LLGL::RasterizerDescriptor, 255
Any
LLGL, 81
AnySamplesPassed
LLGL, 85
AnySamplesPassedConservative
LLGL, 85
AppendAttribute
LLGL::StreamOutputFormat, 356
LLGL::VertexFormat, 382
AppendAttributes
LLGL::StreamOutputFormat, 357
AppendStructuredBuffer
LLGL, 91
application
LLGL::RendererConfigurationVulkan, 267
applicationName
LLGL::ApplicationDescriptor, 107
applicationVersion
LLGL::ApplicationDescriptor, 107
Apps
LLGL, 79
arrayLayer
LLGL::AttachmentDescriptor, 111
LLGL::TextureLocation, 367
arrayLayers
LLGL::TextureDescriptor, 363
arraySize
LLGL::BindingDescriptor, 117
AssertCreateBuffer
LLGL::RenderSystem, 293
AssertCreateBufferArray
LLGL::RenderSystem, 293
AssertCreateRenderPass
LLGL::RenderSystem, 293
AssertCreateRenderTarget
LLGL::RenderSystem, 293
AssertCreateShader
LLGL::RenderSystem, 293
AssertCreateShaderProgram

LLGL::RenderSystem, 293
AssertImageDataSize
LLGL::RenderSystem, 294
AtomicCounter
LLGL, 95
AttachmentClear
LLGL::AttachmentClear, 109
attachmentClears
LLGL::FrameProfile, 208
AttachmentDescriptor
LLGL::AttachmentDescriptor, 111
AttachmentFormatDescriptor
LLGL::AttachmentFormatDescriptor, 113
AttachmentLoadOp
LLGL, 67
AttachmentStoreOp
LLGL, 67
AttachmentType
LLGL, 68
attachments
LLGL::RenderTargetDescriptor, 315
Attn
LLGL, 81
attributes
LLGL::StreamOutputFormat, 357
LLGL::VertexFormat, 383

B
LLGL, 79
b
LLGL::Color< T, 3u >, 141
LLGL::Color< T, 4u >, 146
LLGL::TextureSwizzleRGBA, 372
BC1RGBA
LLGL, 77
BC1RGB
LLGL, 77
BC2RGBA
LLGL, 77
BC3RGBA
LLGL, 77
BGRA8SInt
LLGL, 77
BGRA8SNorm
LLGL, 76
BGRA8UInt
LLGL, 76
BGRA8UNorm
LLGL, 76
BGRA8UNorm_sRGB
LLGL, 76
BGRA
LLGL, 77
BGR
LLGL, 77
Back
LLGL, 71, 78
back
LLGL::StencilDescriptor, 350
baseArrayLayer
LLGL::TextureSubresource, 371
baseMipLevel
LLGL::TextureSubresource, 371
Begin
LLGL::CommandBuffer, 150
BeginQuery
LLGL::CommandBuffer, 150
BeginRenderCondition
LLGL::CommandBuffer, 151
BeginRenderPass
LLGL::CommandBuffer, 151
BeginStreamOutput
LLGL::CommandBuffer, 152
BinaryBuffer
LLGL, 89
BinaryFile
LLGL, 89
bindFlags
LLGL::BindingDescriptor, 117
LLGL::BufferDescriptor, 126
LLGL::TextureDescriptor, 363
binding
LLGL::ShaderResource, 345
BindingDescriptor
LLGL::BindingDescriptor, 117
bindings
LLGL::PipelineLayoutDescriptor, 249
bitSize
LLGL::FormatDescriptor, 203
blend
LLGL::GraphicsPipelineDescriptor, 216
BlendArithmetic
LLGL, 68
blendEnabled
LLGL::BlendTargetDescriptor, 122
BlendFactor
LLGL, 69
blendFactor
LLGL::BlendDescriptor, 119
BlendOp
LLGL, 69
Blit
LLGL::Image, 221
Block
LLGL::RenderingDebugger::Message, 237
BlockAfter
LLGL::RenderingDebugger::Message, 237
blockHeight
LLGL::FormatDescriptor, 203
blockWidth
LLGL::FormatDescriptor, 204
Blue
LLGL, 93
Bool1
LLGL, 94
Bool2
LLGL, 94

Bool3
LLGL, 94
Bool4
LLGL, 94
Border
LLGL, 87
borderColor
LLGL::SamplerDescriptor, 324
borderless
LLGL::CanvasDescriptor, 131
LLGL::WindowDescriptor, 400
BrowserBack
LLGL, 80
BrowserFavorits
LLGL, 80
BrowserForward
LLGL, 80
BrowserHome
LLGL, 80
BrowserRefresh
LLGL, 80
BrowserSearch
LLGL, 80
BrowserStop
LLGL, 80
Buffer
LLGL::Buffer, 123
LLGL::InterfaceID, 235
LLGL, 86, 91
Buffer argument structures for indirect draw and compute commands., 37
Buffer utility functions to determine buffer types., 27
IsByteAddressBuffer, 27
IsRWBuffer, 27
IsStructuredBuffer, 27
IsTypedBuffer, 27
Buffer.h, 403
BufferArray
LLGL::BufferArray, 125
LLGL::InterfaceID, 235
BufferArray.h, 403
bufferCopies
LLGL::FrameProfile, 208
BufferFlags.h, 404
bufferMappings
LLGL::FrameProfile, 209
bufferReads
LLGL::FrameProfile, 209
bufferUpdates
LLGL::FrameProfile, 209
bufferWrites
LLGL::FrameProfile, 209
ByRegionNoWait
LLGL, 86
ByRegionNoWaitInverted
LLGL, 86
ByRegionWait
LLGL, 86
ByRegionWaitInverted
LLGL, 86
ByteAddressBuffer
LLGL, 91
ByteBuffer
LLGL, 66

C
LLGL, 79
CPUAccess
LLGL, 70
Cancel
LLGL, 78
Canvas
LLGL::Canvas::EventListener, 199
LLGL::InterfaceID, 236
Canvas.h, 405
Canvas_EventListener
LLGL::InterfaceID, 236
CanvasFlags.h, 405
Capital
LLGL, 78
Cast
LLGL::Color, 135
LLGL::Color< T, 3u >, 139
LLGL::Color< T, 4u >, 144
CastColorValue
LLGL, 95
CastColorValue< bool, bool >
LLGL, 95
CastColorValue< double, double >
LLGL, 96
CastColorValue< float, float >
LLGL, 96
CastColorValue< std::uint8_t, std::uint8_t >
LLGL, 96
CastTo
LLGL, 96, 97
centered
LLGL::WindowDescriptor, 400
Clamp
LLGL, 87
clamp
LLGL::DepthBiasDescriptor, 181
Clear
LLGL::CommandBuffer, 153
LLGL::FrameProfile, 208
LLGL, 67, 78, 82
ClearAttachments
LLGL::CommandBuffer, 153
ClearShaderReflection
LLGL::ShaderProgram, 337
clearValue
LLGL::AttachmentClear, 109
LLGL::TextureDescriptor, 363
ClipDistance
LLGL, 93
clippingInvocations
LLGL::QueryPipelineStatistics, 253

clippingPrimitives
LLGL::QueryPipelineStatistics, 253
ClippingRange
LLGL, 69
clippingRange
LLGL::RenderingCapabilities, 271
CodeFile
LLGL, 89
CodeString
LLGL, 89
Color
LLGL::ClearFlags, 132
LLGL::Color, 135
LLGL::Color< T, 3u >, 139
LLGL::Color< T, 4u >, 143, 144
LLGL, 68, 93
color
LLGL::ClearValue, 133
Color.h, 405
colorArithmetic
LLGL::BlendTargetDescriptor, 122
ColorAttachment
LLGL::BindFlags, 115
colorAttachment
LLGL::AttachmentClear, 109
colorAttachments
LLGL::RenderPassDescriptor, 289
colorBits
LLGL::VideoModeDescriptor, 386
ColorDepth
LLGL::ClearFlags, 132
colorMap
LLGL::NativeContextHandle, 242
colorMask
LLGL::BlendTargetDescriptor, 122
ColorRGB.h, 407
ColorRGBA.h, 407
ColorRGBAb
LLGL, 67
ColorRGBAd
LLGL, 67
ColorRGBAf
LLGL, 67
ColorRGBAT
LLGL, 67
ColorRGBAub
LLGL, 67
ColorRGBb
LLGL, 67
ColorRGBd
LLGL, 67
ColorRGBf
LLGL, 67
ColorRGBT
LLGL, 67
ColorRGBub
LLGL, 67
CombinedTextureSampler
LLGL::BindFlags, 116
Comma
LLGL, 81
CommandBuffer
LLGL::CommandBuffer, 150
LLGL::InterfaceID, 235
CommandBuffer.h, 408
commandBufferEncodings
LLGL::FrameProfile, 209
CommandBufferFlags.h, 408
commandBufferSubmittions
LLGL::FrameProfile, 209
CommandQueue
LLGL::CommandQueue, 176
LLGL::InterfaceID, 235
CommandQueue.h, 409
compareEnabled
LLGL::SamplerDescriptor, 324
CompareOp
LLGL, 70
compareOp
LLGL::DepthDescriptor, 182
LLGL::SamplerDescriptor, 324
LLGL::StencilFaceDescriptor, 351
CompareSWO
Global functions for Strict-Weak-Order (SWO) comparisons., 21
CompatibilityProfile
LLGL, 82
components
LLGL::Color, 137
LLGL::Color< T, 3u >, 141
LLGL::Color< T, 4u >, 146
LLGL::FormatDescriptor, 204
LLGL::StreamOutputAttribute, 355
compressed
LLGL::FormatDescriptor, 204
CompressedRGBA
LLGL, 77
CompressedRGB
LLGL, 77
Compute
LLGL, 89
ComputePipeline
LLGL::InterfaceID, 235
ComputePipeline.h, 409
computePipelineBindings
LLGL::FrameProfile, 210
ComputePipelineFlags.h, 410
computeResourceHeapBindings
LLGL::FrameProfile, 210
computeShader
LLGL::ShaderProgramDescriptor, 341
computeShaderInvocations
LLGL::QueryPipelineStatistics, 253
ComputeStage
LLGL::StageFlags, 349
conservativeRasterization

LLGL::RasterizerDescriptor, 255
ConstantBuffer
LLGL::BindFlags, 115
constantBufferBindings
LLGL::FrameProfile, 210
ConstantBufferDesc
Global utility functions, especially to fill descriptor structures., 45
constantBufferSize
LLGL::ShaderResource, 345
constantFactor
LLGL::DepthBiasDescriptor, 181
Constants.h, 410
ConsumeStructuredBuffer
LLGL, 91
contextProfile
LLGL::RendererConfigurationOpenGL, 265
Control
LLGL, 78
Convert
LLGL::Image, 222
ConvertImageBuffer
Image utility functions to classify and convert image data., 32, 33
Copy
LLGL, 82
CopyBuffer
LLGL::CommandBuffer, 154
CopyInverted
LLGL, 82
CopyTexture
LLGL::CommandBuffer, 154
CoreProfile
LLGL, 82
cpuAccessFlags
LLGL::BufferDescriptor, 126
LLGL::TextureDescriptor, 364
CrSel
LLGL, 81
Create
LLGL::Canvas, 129
LLGL::Timer, 376
LLGL::Window, 394
CreateBuffer
LLGL::RenderSystem, 294
CreateBufferArray
LLGL::RenderSystem, 294
CreateCommandBuffer
LLGL::RenderSystem, 295
CreateComputePipeline
LLGL::RenderSystem, 295
CreateFence
LLGL::RenderSystem, 295
CreateGraphicsPipeline
LLGL::RenderSystem, 295
CreatePipelineLayout
LLGL::RenderSystem, 296
CreateQueryHeap
LLGL::RenderSystem, 296
CreateRenderContext
LLGL::RenderSystem, 296
CreateRenderPass
LLGL::RenderSystem, 297
CreateRenderTarget
LLGL::RenderSystem, 297
CreateResourceHeap
LLGL::RenderSystem, 297
CreateSampler
LLGL::RenderSystem, 298
CreateShader
LLGL::RenderSystem, 298
CreateShaderProgram
LLGL::RenderSystem, 298
CreateTexture
LLGL::RenderSystem, 299
CullDistance
LLGL, 93
CullMode
LLGL, 71
cullMode
LLGL::RasterizerDescriptor, 255
customMultiSampling
LLGL::RenderTargetDescriptor, 315

D
LLGL, 79
D0
LLGL, 78
D1
LLGL, 78
D16UNorm
LLGL, 77
D2
LLGL, 78
D24UNormS8UInt
LLGL, 77
D3
LLGL, 78
D32Float
LLGL, 77
D32FloatS8X24UInt
LLGL, 77
D4
LLGL, 78
D5
LLGL, 78
D6
LLGL, 79
D7
LLGL, 79
D8
LLGL, 79
D9
LLGL, 79
data
LLGL::DstImageDescriptor, 194
LLGL::SrcImageDescriptor, 348

Data type utility functions., 31
DataTypeSize, 31
IsFloatDataType, 31
IsIntDataType, 31
IsUIntDataType, 31
dataSize
LLGL::DstImageDescriptor, 194
LLGL::SrcImageDescriptor, 348
DataType
LLGL, 71
dataType
LLGL::DstImageDescriptor, 194
LLGL::FormatDescriptor, 204
LLGL::SrcImageDescriptor, 348
DataTypeSize
Data type utility functions., 31
Debug
LLGL::ShaderCompileFlags, 329
DebugCallback
Global type aliases to callback interfaces., 25
debugCallback
LLGL::RenderSystemDescriptor, 309
DecClamp
LLGL, 91
DecWrap
LLGL, 91
DeferredSubmit
LLGL::CommandBufferFlags, 174
defines
LLGL::ShaderDescriptor, 331
definition
LLGL::ShaderMacro, 335
Delete
LLGL, 78
Depth
LLGL::ClearFlags, 132
LLGL, 68, 77, 93
depth
LLGL::ClearValue, 133
LLGL::Extent3D, 201
LLGL::FormatDescriptor, 204
LLGL::GraphicsPipelineDescriptor, 216
depthAttachment
LLGL::RenderPassDescriptor, 289
depthBias
LLGL::RasterizerDescriptor, 256
depthBits
LLGL::VideoModeDescriptor, 386
depthClampEnabled
LLGL::RasterizerDescriptor, 256
depthFailOp
LLGL::StencilFaceDescriptor, 351
DepthGreater
LLGL, 93
DepthLess
LLGL, 93
depthPassOp
LLGL::StencilFaceDescriptor, 351
DepthStencil
LLGL::ClearFlags, 132
LLGL, 68, 77
DepthStencilAttachment
LLGL::BindFlags, 116
deviceName
LLGL::RendererInfo, 270
Direct3D10
LLGL::RendererID, 269
Direct3D11
LLGL::RendererID, 269
Direct3D12
LLGL::RendererID, 269
Direct3D9
LLGL::RendererID, 269
disableClearOnResize
LLGL::WindowBehavior, 399
Disabled
LLGL, 71, 82
discardEnabled
LLGL::RasterizerDescriptor, 256
Dispatch
LLGL::CommandBuffer, 155
dispatchCommands
LLGL::FrameProfile, 210
DispatchIndirect
LLGL::CommandBuffer, 155
Display
LLGL::InterfaceID, 235
display
LLGL::NativeContextHandle, 242
LLGL::NativeHandle, 243
Display.h, 410
DisplayFlags.h, 410
displayModes
LLGL::VideoOutputDescriptor, 388
Double1
LLGL, 94
Double2
LLGL, 94
Double2x2
LLGL, 95
Double2x3
LLGL, 95
Double2x4
LLGL, 95
Double3
LLGL, 94
Double3x2
LLGL, 95
Double3x3
LLGL, 95
Double3x4
LLGL, 95
Double4
LLGL, 94
Double4x2
LLGL, 95

Double4x3
LLGL, 95
Double4x4
LLGL, 95
Down
LLGL, 78
Draw
LLGL::CommandBuffer, 155
drawCommands
LLGL::FrameProfile, 210
DrawIndexed
LLGL::CommandBuffer, 156
DrawIndexedIndirect
LLGL::CommandBuffer, 156
DrawIndexedInstanced
LLGL::CommandBuffer, 157
DrawIndirect
LLGL::CommandBuffer, 158
DrawInstanced
LLGL::CommandBuffer, 159
DstAlpha
LLGL, 69
dstAlpha
LLGL::BlendTargetDescriptor, 122
DstColor
LLGL, 69
dstColor
LLGL::BlendTargetDescriptor, 122
DstImageDescriptor
LLGL::DstImageDescriptor, 194
DynamicUsage
LLGL::MiscFlags, 239

E
LLGL, 79
ESProfile
LLGL, 82
ESSL_100
LLGL, 90
ESSL_300
LLGL, 90
ESSL_310
LLGL, 90
ESSL_320
LLGL, 90
ESSL
LLGL, 90
enabled
LLGL::MultiSamplingDescriptor, 241
LLGL::VsyncDescriptor, 391
enabledLayers
LLGL::RendererConfigurationVulkan, 267
End
LLGL::CommandBuffer, 160
LLGL, 78
EndQuery
LLGL::CommandBuffer, 160
EndRenderCondition
LLGL::CommandBuffer, 160
EndRenderPass
LLGL::CommandBuffer, 160
EndStreamOutput
LLGL::CommandBuffer, 160
engineName
LLGL::ApplicationDescriptor, 108
engineVersion
LLGL::ApplicationDescriptor, 108
entryPoint
LLGL::ShaderDescriptor, 331
Equal
LLGL, 70
Equiv
LLGL, 82
ErEOF
LLGL, 81
Error
LLGL::Log, 102
ErrorType
LLGL, 72
Escape
LLGL, 78
ExSel
LLGL, 81
Exe
LLGL, 78
Execute
LLGL::CommandBuffer, 161
Exponent
LLGL, 81
Export.h, 411
LLGL_EXPORT, 411
extent
LLGL::TextureDescriptor, 364
LLGL::TextureRegion, 369
Extent2D
LLGL::Extent2D, 200
Extent3D
LLGL::Extent3D, 201

F
LLGL, 79
F1
LLGL, 80
F10
LLGL, 80
F11
LLGL, 80
F12
LLGL, 80
F13
LLGL, 80
F14
LLGL, 80
F15
LLGL, 80
F16
LLGL, 80
F17

LLGL, 80
F18
LLGL, 80
F19
LLGL, 80
F2
LLGL, 80
F20
LLGL, 80
F21
LLGL, 80
F22
LLGL, 80
F23
LLGL, 80
F24
LLGL, 80
F3
LLGL, 80
F4
LLGL, 80
F5
LLGL, 80
F6
LLGL, 80
F7
LLGL, 80
F8
LLGL, 80
F9
LLGL, 80
features
LLGL::RenderingCapabilities, 271
Fence
LLGL::InterfaceID, 235
Fence.h, 411
fenceSubmissions
LLGL::FrameProfile, 211
Fill
LLGL::Image, 222
LLGL, 82
FinalizeDisplayModes
LLGL::Display, 185
FinalizeShaderReflection
LLGL::ShaderProgram, 337
FindModules
LLGL::RenderSystem, 299
FindUniformLocation
LLGL::ShaderProgram, 337
firstIndex
LLGL::DrawIndexedIndirectArguments, 189
firstInstance
LLGL::DrawIndexedIndirectArguments, 189
LLGL::DrawIndirectArguments, 191
LLGL::DrawPatchIndirectArgument, 192
firstPatch
LLGL::DrawPatchIndirectArgument, 192
firstVertex
LLGL::DrawIndirectArguments, 191
FixedSamples
LLGL::MiscFlags, 239
flags
LLGL::AttachmentClear, 110
LLGL::CommandBufferDescriptor, 173
LLGL::ShaderDescriptor, 332
Float1
LLGL, 94
Float16
LLGL, 72
Float2
LLGL, 94
Float2x2
LLGL, 95
Float2x3
LLGL, 95
Float2x4
LLGL, 95
Float3
LLGL, 94
Float32
LLGL, 72
Float3x2
LLGL, 95
Float3x3
LLGL, 95
Float3x4
LLGL, 95
Float4
LLGL, 94
Float4x2
LLGL, 95
Float4x3
LLGL, 95
Float4x4
LLGL, 95
Float64
LLGL, 72
Format
LLGL, 72
format
LLGL::AttachmentFormatDescriptor, 113
LLGL::BufferDescriptor::IndexBuffer, 228
LLGL::BufferDescriptor::StorageBuffer, 353
LLGL::BufferDescriptor::VertexBuffer, 381
LLGL::DstImageDescriptor, 194
LLGL::FormatDescriptor, 204
LLGL::FragmentAttribute, 205
LLGL::ShaderDescriptor::StreamOutput, 354
LLGL::SrcImageDescriptor, 348
LLGL::TextureDescriptor, 364
LLGL::TextureViewDescriptor, 373
LLGL::VertexAttribute, 380
Format.h, 412
ForwardDecls.h, 413
Fragment
LLGL, 89

FragmentAttribute.h, 414
fragmentShader
LLGL::ShaderProgramDescriptor, 341
fragmentShaderInvocations
LLGL::QueryPipelineStatistics, 253
FragmentStage
LLGL::StageFlags, 349
FrameProfile
LLGL::FrameProfile, 208
frameProfile
LLGL::RenderingProfiler, 287
Front
LLGL, 71
front
LLGL::StencilDescriptor, 350
frontCCW
LLGL::RasterizerDescriptor, 256
FrontFacing
LLGL, 93
fullscreen
LLGL::VideoModeDescriptor, 386

G
LLGL, 79
g
LLGL::Color< T, 3u >, 141
LLGL::Color< T, 4u >, 146
LLGL::TextureSwizzleRGBA, 372
GLSL_110
LLGL, 90
GLSL_120
LLGL, 90
GLSL_130
LLGL, 90
GLSL_140
LLGL, 90
GLSL_150
LLGL, 90
GLSL_330
LLGL, 90
GLSL_400
LLGL, 90
GLSL_410
LLGL, 90
GLSL_420
LLGL, 90
GLSL_430
LLGL, 90
GLSL_440
LLGL, 90
GLSL_450
LLGL, 90
GLSL_460
LLGL, 90
GLSL
LLGL, 90
GenerateEmptyByteBuffer
Image utility functions to classify and convert image data., 34
GenerateImageBuffer
Image utility functions to classify and convert image data., 34
GenerateMips
LLGL::CommandBuffer, 161
LLGL::MiscFlags, 239
Geometry
LLGL, 89
geometryShader
LLGL::ShaderProgramDescriptor, 341
geometryShaderInvocations
LLGL::QueryPipelineStatistics, 253
geometryShaderPrimitives
LLGL::QueryPipelineStatistics, 253
GeometryStage
LLGL::StageFlags, 349
GetAnyKeyCount
LLGL::Input, 230
GetBehavior
LLGL::Window, 394
GetBindFlags
LLGL::Buffer, 123
LLGL::BufferArray, 125
GetBytesPerPixel
LLGL::Image, 222
GetColorFormat
LLGL::RenderContext, 259
GetCommandQueue
LLGL::RenderSystem, 299
GetConfiguration
LLGL::RenderSystem, 299
GetContentSize
LLGL::Surface, 358
GetData
LLGL::Image, 223
GetDataSize
LLGL::Image, 223
GetDataType
LLGL::Image, 223
GetDeltaTime
LLGL::Timer, 376
GetDepthStencilFormat
LLGL::RenderContext, 259
GetDepthStride
LLGL::Image, 223
GetDesc
LLGL::Buffer, 123
LLGL::Texture, 361
LLGL::Window, 394
GetDeviceName
LLGL::Display, 185
GetDisplayMode
LLGL::Display, 185
GetDstDesc
LLGL::Image, 223
GetEnteredChars
LLGL::Input, 230
GetExtent

LLGL::Image, 223
GetExtentRatio
LLGL, 97
GetFormat
LLGL::Image, 223
GetFormatDesc
Hardware format utility functions., 28
GetFrequency
LLGL::Timer, 376
GetGroupName
LLGL::RenderingDebugger::Message, 237
GetID
LLGL::Version, 104
GetMajor
LLGL::Version, 104
GetMinor
LLGL::Version, 105
GetMipExtent
LLGL::Texture, 361
GetMouseMotion
LLGL::Input, 230
GetMousePosition
LLGL::Input, 230
GetName
LLGL::RenderSystem, 299
GetNativeHandle
LLGL::Surface, 359
GetNumColorAttachments
LLGL::RenderContext, 260
LLGL::RenderTarget, 312
GetNumPixels
LLGL::Image, 223
GetOccurrences
LLGL::RenderingDebugger::Message, 238
GetOffset
LLGL::Display, 185
GetPosition
LLGL::Window, 394
GetPrimitiveTopologyPatchSize
LLGL, 97
GetRenderPass
LLGL::RenderTarget, 312
GetRendererID
LLGL::RenderSystem, 300
GetRendererInfo
LLGL::RenderSystem, 300
GetRenderingCaps
LLGL::RenderSystem, 300
GetReport
LLGL::Shader, 328
LLGL::ShaderProgram, 337
GetResolution
LLGL::RenderContext, 260
LLGL::RenderTarget, 312
GetResourceType
LLGL::Buffer, 124
LLGL::Resource, 317
LLGL::Sampler, 322
LLGL::Texture, 362
GetRevision
LLGL::Version, 105
GetRowStride
LLGL::Image, 224
GetSize
LLGL::VertexAttribute, 379
LLGL::Window, 395
GetSource
LLGL::RenderingDebugger::Message, 238
GetSrcDesc
LLGL::Image, 224
GetStageFlags
LLGL::Shader, 328
GetStatus
LLGL::Version, 105
GetString
LLGL::Version, 105
GetSupportedDisplayModes
LLGL::Display, 185
GetSurface
LLGL::RenderContext, 260
GetText
LLGL::RenderingDebugger::Message, 238
GetTitle
LLGL::Canvas, 129
LLGL::Window, 395
GetType
LLGL::QueryHeap, 250
LLGL::Shader, 328
LLGL::Texture, 362
GetVideoMode
LLGL::RenderContext, 260
GetVsync
LLGL::RenderContext, 260
GetWheelMotion
LLGL::Input, 230
GetWorkGroupSize
LLGL::ShaderProgram, 338
Global functions for Strict-Weak-Order (SWO) comparisons., 21
CompareSWO, 21
Global operators for basic data structures., 22
operator!=, 22, 23
operator+, 23
operator-, 23, 24
operator==, 24
Global type aliases to callback interfaces., 25
DebugCallback, 25
ValidateRenderingCapsFunc, 25
Global type-to-string conversion functions., 38
ToString, 38, 39
Global utility functions, especially to fill descriptor structures., 44
ConstantBufferDesc, 45
IndexBufferDesc, 45
PipelineLayoutDesc, 45
RenderPassDesc, 47

ShaderDescFromFile, 47
ShaderProgramDesc, 47, 48
StorageBufferDesc, 48
Texture1DArrayDesc, 48
Texture1DDesc, 48
Texture2DArrayDesc, 49
Texture2DDesc, 49
Texture2DMSArrayDesc, 49
Texture2DMSDesc, 49
Texture3DDesc, 49
TextureCubeArrayDesc, 50
TextureCubeDesc, 50
VertexBufferDesc, 50
GraphicsPipeline
LLGL::InterfaceID, 235
GraphicsPipeline.h, 414
graphicsPipelineBindings
LLGL::FrameProfile, 211
GraphicsPipelineFlags.h, 414
graphicsResourceHeapBindings
LLGL::FrameProfile, 211
Greater
LLGL, 70
GreaterEqual
LLGL, 70
Green
LLGL, 93

H
LLGL, 79
HLSL_2_0
LLGL, 90
HLSL_2_0a
LLGL, 90
HLSL_2_0b
LLGL, 90
HLSL_3_0
LLGL, 90
HLSL_4_0
LLGL, 90
HLSL_4_1
LLGL, 90
HLSL_5_0
LLGL, 90
HLSL_5_1
LLGL, 90
HLSL
LLGL, 90
Hardware format utility functions., 28
GetFormatDesc, 28
IsCompressedFormat, 28
IsDepthFormat, 28
IsDepthStencilFormat, 29
IsFloatFormat, 29
IsIntegralFormat, 29
IsNormalizedFormat, 29
IsStencilFormat, 30
has3DTextures
LLGL::RenderingFeatures, 277
hasArrayTextures
LLGL::RenderingFeatures, 277
hasComputeShaders
LLGL::RenderingFeatures, 277
hasConservativeRasterization
LLGL::RenderingFeatures, 277
hasConstantBuffers
LLGL::RenderingFeatures, 277
hasCubeArrayTextures
LLGL::RenderingFeatures, 278
hasCubeTextures
LLGL::RenderingFeatures, 278
HasDepthAttachment
LLGL::RenderContext, 261
LLGL::RenderTarget, 313
hasDirectResourceBinding
LLGL::RenderingFeatures, 278
HasErrors
LLGL::Shader, 328
LLGL::ShaderProgram, 338
HasFocus
LLGL::Window, 395
hasGeometryShaders
LLGL::RenderingFeatures, 278
hasIndirectDrawing
LLGL::RenderingFeatures, 278
hasInstancing
LLGL::RenderingFeatures, 279
hasLogicOp
LLGL::RenderingFeatures, 279
hasMultiSampleTextures
LLGL::RenderingFeatures, 279
hasOffsetInstancing
LLGL::RenderingFeatures, 279
hasPipelineStatistics
LLGL::RenderingFeatures, 279
HasQuit
LLGL::Canvas, 130
LLGL::Window, 395
hasRenderCondition
LLGL::RenderingFeatures, 280
hasRenderTargets
LLGL::RenderingFeatures, 280
hasSamplers
LLGL::RenderingFeatures, 280
HasStencilAttachment
LLGL::RenderContext, 261
LLGL::RenderTarget, 313
hasStorageBuffers
LLGL::RenderingFeatures, 280
hasStreamOutputs
LLGL::RenderingFeatures, 280
hasTessellationShaders
LLGL::RenderingFeatures, 280
hasTextureViewSwizzle
LLGL::RenderingFeatures, 281
hasTextureViews
LLGL::RenderingFeatures, 281

hasUniforms
LLGL::RenderingFeatures, 281
hasViewportArrays
LLGL::RenderingFeatures, 281
height
LLGL::Extent2D, 200
LLGL::Extent3D, 201
LLGL::Scissor, 326
LLGL::Viewport, 390
Help
LLGL, 78
Home
LLGL, 78

I
LLGL, 79
IOSNativeHandle.h, 420
Image
LLGL::Image, 221
LLGL, 95
Image utility functions to classify and convert image data., 32
ConvertImageBuffer, 32, 33
GenerateEmptyByteBuffer, 34
GenerateImageBuffer, 34
ImageDataSize, 35
ImageFormatSize, 35
IsCompressedFormat, 35
IsDepthStencilFormat, 36
Image.h, 417
ImageDataSize
Image utility functions to classify and convert image data., 35
ImageFlags.h, 417
ImageFormat
LLGL, 77
ImageFormatSize
Image utility functions to classify and convert image data., 35
ImproperArgument
LLGL, 95
ImproperState
LLGL, 95
IncClamp
LLGL, 91
IncOccurrence
LLGL::RenderingDebugger::Message, 238
IncWrap
LLGL, 91
IncompleteAttachments
LLGL::ShaderProgram, 336
independentBlendEnabled
LLGL::BlendDescriptor, 120
IndexBuffer
LLGL::BindFlags, 115
indexBuffer
LLGL::BufferDescriptor, 127
indexBufferBindings
LLGL::FrameProfile, 211
IndexBufferDesc
Global utility functions, especially to fill descriptor structures., 45
IndirectArguments.h, 418
IndirectBuffer
LLGL::BindFlags, 115
Information
LLGL::Log, 103
Input
LLGL::Input, 230
LLGL::InterfaceID, 236
Input.h, 419
inputAssemblyPrimitives
LLGL::QueryPipelineStatistics, 254
inputAssemblyVertices
LLGL::QueryPipelineStatistics, 254
inputSlot
LLGL::VertexFormat, 383
Insert
LLGL, 78
instanceDivisor
LLGL::VertexAttribute, 380
InstanceID
LLGL, 93
InstantiateList
LLGL::Display, 185
InstantiatePrimary
LLGL::Display, 186
Int1
LLGL, 94
Int16
LLGL, 72
Int2
LLGL, 94
Int3
LLGL, 94
Int32
LLGL, 72
Int4
LLGL, 94
Int8
LLGL, 72
Interface
LLGL::InterfaceID, 235
Interface.h, 419
LLGL_DECLARE_INTERFACE, 420
LLGL_IMPLEMENT_INTERFACE, 420
InterfaceID.h, 420
interval
LLGL::VsyncDescriptor, 391
InvBlendFactor
LLGL, 69
InvDstAlpha
LLGL, 69
InvDstColor
LLGL, 69
InvSrc1Alpha
LLGL, 69

InvSrc1Color
LLGL, 69
InvSrcAlpha
LLGL, 69
InvSrcColor
LLGL, 69
InvalidArgument
LLGL, 72
InvalidByteCode
LLGL::ShaderProgram, 336
InvalidComposition
LLGL::ShaderProgram, 336
InvalidState
LLGL, 72
Invert
LLGL, 82, 91
invertFrontFace
LLGL::OpenGLDependentStateDescriptor, 248
IsArrayTexture
Texture utility functions to determine texture dimension and buffer sizes., 40
IsBlocked
LLGL::RenderingDebugger::Message, 238
IsByteAddressBuffer
Buffer utility functions to determine buffer types., 27
IsCompressedFormat
Hardware format utility functions., 28
Image utility functions to classify and convert image data., 35
IsCubeTexture
Texture utility functions to determine texture dimension and buffer sizes., 40
IsCursorShown
LLGL::Display, 186
IsDepthFormat
Hardware format utility functions., 28
IsDepthStencilFormat
Hardware format utility functions., 29
Image utility functions to classify and convert image data., 36
IsFloatDataType
Data type utility functions., 31
IsFloatFormat
Hardware format utility functions., 29
IsInstanceOf
LLGL::Interface, 234
IsIntDataType
Data type utility functions., 31
IsIntegralFormat
Hardware format utility functions., 29
IsMipMappedTexture
Texture utility functions to determine texture dimension and buffer sizes., 40
IsMultiSampleTexture
Texture utility functions to determine texture dimension and buffer sizes., 41
IsNormalizedFormat
Hardware format utility functions., 29
IsPrimary
LLGL::Display, 186
IsPrimitiveTopologyPatches
LLGL, 97
IsRWBuffer
Buffer utility functions to determine buffer types., 27
IsRegionInside
LLGL::Image, 224
IsRenderContext
LLGL::RenderTarget, 313
IsRunning
LLGL::Timer, 376
IsShaderSourceBinary
LLGL, 97
IsShaderSourceCode
LLGL, 98
IsShown
LLGL::Window, 395
IsStencilFormat
Hardware format utility functions., 30
IsStructuredBuffer
Buffer utility functions to determine buffer types., 27
IsTextureSwizzleIdentity
Texture utility functions to determine texture dimension and buffer sizes., 41
IsTypedBuffer
Buffer utility functions to determine buffer types., 27
IsUIntDataType
Data type utility functions., 31

J
LLGL, 79

K
LLGL, 79
Keep
LLGL, 91
Key
LLGL, 77
Key.h, 421
KeyDoubleClick
LLGL::Input, 230
KeyDown
LLGL::Input, 231
KeyDownRepeated
LLGL::Input, 231
KeyPressed
LLGL::Input, 231
KeyUp
LLGL::Input, 231
Keypad0
LLGL, 79
Keypad1
LLGL, 79
Keypad2
LLGL, 79
Keypad3
LLGL, 79
Keypad4

LLGL, 79
Keypad5
LLGL, 79
Keypad6
LLGL, 79
Keypad7
LLGL, 79
Keypad8
LLGL, 79
Keypad9
LLGL, 79
KeypadDecimal
LLGL, 80
KeypadDivide
LLGL, 80
KeypadMinus
LLGL, 80
KeypadMultiply
LLGL, 79
KeypadPlus
LLGL, 80
KeypadSeparator
LLGL, 80

L
LLGL, 79
LButton
LLGL, 78
LControl
LLGL, 80
LLGL.h, 422
LLGL::ApplicationDescriptor, 107
applicationName, 107
applicationVersion, 107
engineName, 108
engineVersion, 108
LLGL::AttachmentClear, 108
AttachmentClear, 109
clearValue, 109
colorAttachment, 109
flags, 110
operator=, 109
LLGL::AttachmentDescriptor, 110
arrayLayer, 111
AttachmentDescriptor, 111
mipLevel, 111
texture, 112
type, 112
LLGL::AttachmentFormatDescriptor, 112
AttachmentFormatDescriptor, 113
format, 113
loadOp, 113
storeOp, 114
LLGL::BindFlags, 114
ColorAttachment, 115
CombinedTextureSampler, 116
ConstantBuffer, 115
DepthStencilAttachment, 116
IndexBuffer, 115
IndirectBuffer, 115
Sampled, 115
Storage, 115
StreamOutputBuffer, 115
VertexBuffer, 115
LLGL::BindingDescriptor, 116
arraySize, 117
bindFlags, 117
BindingDescriptor, 117
name, 118
slot, 118
stageFlags, 118
type, 118
LLGL::BlendDescriptor, 119
alphaToCoverageEnabled, 119
blendFactor, 119
independentBlendEnabled, 120
logicOp, 120
targets, 120
LLGL::BlendTargetDescriptor, 121
alphaArithmetic, 122
blendEnabled, 122
colorArithmetic, 122
colorMask, 122
dstAlpha, 122
dstColor, 122
srcAlpha, 122
srcColor, 122
LLGL::Buffer, 123
Buffer, 123
GetBindFlags, 123
GetDesc, 123
GetResourceType, 124
LLGL::BufferArray, 124
BufferArray, 125
GetBindFlags, 125
LLGL::BufferDescriptor, 125
bindFlags, 126
cpuAccessFlags, 126
indexBuffer, 127
miscFlags, 127
size, 127
storageBuffer, 127
vertexBuffer, 127
LLGL::BufferDescriptor::IndexBuffer, 228
format, 228
LLGL::BufferDescriptor::StorageBuffer, 352
format, 353
storageType, 353
stride, 353
LLGL::BufferDescriptor::VertexBuffer, 381
format, 381
LLGL::CPUAccessFlags, 179
Read, 180
ReadWrite, 180
Write, 180
LLGL::Canvas, 128
AdaptForVideoMode, 129

AddEventListener, 129
Create, 129
GetTitle, 129
HasQuit, 130
OnProcessEvents, 130
PostQuit, 130
ProcessEvents, 130
RemoveEventListener, 130
SetTitle, 131
LLGL::Canvas::EventListener, 198
Canvas, 199
OnProcessEvents, 199
OnQuit, 199
LLGL::CanvasDescriptor, 131
borderless, 131
title, 131
LLGL::ClearFlags, 132
All, 132
Color, 132
ColorDepth, 132
Depth, 132
DepthStencil, 132
Stencil, 132
LLGL::ClearValue, 132
color, 133
depth, 133
stencil, 133
LLGL::Color
Cast, 135
Color, 135
components, 137
operator∗=, 136
operator+=, 136
operator-, 136
operator-=, 136
operator/=, 136
operator[], 136, 137
Ptr, 137
LLGL::Color< T, 3u >, 137
b, 141
Cast, 139
Color, 139
components, 141
g, 141
operator∗=, 140
operator+=, 140
operator-, 140
operator-=, 140
operator/=, 140
operator[], 140, 141
Ptr, 141
r, 142
ToRGBA, 141
LLGL::Color< T, 4u >, 142
a, 146
b, 146
Cast, 144
Color, 143, 144
components, 146
g, 146
operator∗=, 145
operator+=, 145
operator-, 145
operator-=, 145
operator/=, 145
operator[], 145, 146
Ptr, 146
r, 147
ToRGB, 146
LLGL::Color< T, N >, 133
LLGL::CommandBuffer, 147
Begin, 150
BeginQuery, 150
BeginRenderCondition, 151
BeginRenderPass, 151
BeginStreamOutput, 152
Clear, 153
ClearAttachments, 153
CommandBuffer, 150
CopyBuffer, 154
CopyTexture, 154
Dispatch, 155
DispatchIndirect, 155
Draw, 155
DrawIndexed, 156
DrawIndexedIndirect, 156
DrawIndexedInstanced, 157
DrawIndirect, 158
DrawInstanced, 159
End, 160
EndQuery, 160
EndRenderCondition, 160
EndRenderPass, 160
EndStreamOutput, 160
Execute, 161
GenerateMips, 161
PopDebugGroup, 162
PushDebugGroup, 162
ResetResourceSlots, 163
SetClearColor, 163
SetClearDepth, 164
SetClearStencil, 164
SetComputePipeline, 164
SetComputeResourceHeap, 165
SetGraphicsAPIDependentState, 165
SetGraphicsPipeline, 166
SetGraphicsResourceHeap, 166
SetIndexBuffer, 167
SetResource, 168
SetScissor, 168
SetScissors, 168
SetStreamOutputBuffer, 169
SetStreamOutputBufferArray, 169
SetUniform, 170
SetUniforms, 170
SetVertexBuffer, 171

SetVertexBufferArray, 171
SetViewport, 171
SetViewports, 171
UpdateBuffer, 172
LLGL::CommandBufferDescriptor, 173
flags, 173
numNativeBuffers, 173
LLGL::CommandBufferFlags, 174
DeferredSubmit, 174
MultiSubmit, 174
LLGL::CommandQueue, 175
CommandQueue, 176
QueryResult, 176
Submit, 176, 177
WaitFence, 177
WaitIdle, 177
LLGL::ComputePipeline, 178
LLGL::ComputePipelineDescriptor, 178
pipelineLayout, 179
shaderProgram, 179
LLGL::Constants, 101
LLGL::DepthBiasDescriptor, 180
clamp, 181
constantFactor, 181
slopeFactor, 181
LLGL::DepthDescriptor, 181
compareOp, 182
testEnabled, 182
writeEnabled, 182
LLGL::DispatchIndirectArguments, 182
numThreadGroups, 183
LLGL::Display, 183
FinalizeDisplayModes, 185
GetDeviceName, 185
GetDisplayMode, 185
GetOffset, 185
GetSupportedDisplayModes, 185
InstantiateList, 185
InstantiatePrimary, 186
IsCursorShown, 186
IsPrimary, 186
ResetDisplayMode, 186
SetDisplayMode, 186
ShowCursor, 187
LLGL::DisplayModeDescriptor, 187
refreshRate, 188
resolution, 188
LLGL::DrawIndexedIndirectArguments, 188
firstIndex, 189
firstInstance, 189
numIndices, 189
numInstances, 189
vertexOffset, 190
LLGL::DrawIndirectArguments, 190
firstInstance, 191
firstVertex, 191
numInstances, 191
numVertices, 191
LLGL::DrawPatchIndirectArgument, 191
firstInstance, 192
firstPatch, 192
numInstances, 192
numPatches, 192
LLGL::DstImageDescriptor, 193
data, 194
dataSize, 194
dataType, 194
DstImageDescriptor, 194
format, 194
LLGL::Extent2D, 199
Extent2D, 200
height, 200
width, 200
LLGL::Extent3D, 200
depth, 201
Extent3D, 201
height, 201
width, 201
LLGL::Fence, 202
LLGL::FormatDescriptor, 202
bitSize, 203
blockHeight, 203
blockWidth, 204
components, 204
compressed, 204
dataType, 204
depth, 204
format, 204
normalized, 204
packed, 204
sRGB, 204
stencil, 205
LLGL::FragmentAttribute, 205
format, 205
name, 205
semanticIndex, 206
systemValue, 206
LLGL::FrameProfile, 206
Accumulate, 208
attachmentClears, 208
bufferCopies, 208
bufferMappings, 209
bufferReads, 209
bufferUpdates, 209
bufferWrites, 209
Clear, 208
commandBufferEncodings, 209
commandBufferSubmittions, 209
computePipelineBindings, 210
computeResourceHeapBindings, 210
constantBufferBindings, 210
dispatchCommands, 210
drawCommands, 210
fenceSubmissions, 211
FrameProfile, 208
graphicsPipelineBindings, 211

graphicsResourceHeapBindings, 211
indexBufferBindings, 211
mipMapsGenerations, 211
querySections, 212
renderConditionSections, 212
renderPassSections, 212
sampledBufferBindings, 212
sampledTextureBindings, 212
samplerBindings, 213
storageBufferBindings, 213
storageTextureBindings, 213
streamOutputBufferBindings, 213
streamOutputSections, 213
textureCopies, 214
textureMappings, 214
textureReads, 214
textureWrites, 214
values, 214
vertexBufferBindings, 214
LLGL::GraphicsPipeline, 215
LLGL::GraphicsPipelineDescriptor, 216
blend, 216
depth, 216
pipelineLayout, 217
primitiveTopology, 217
rasterizer, 217
renderPass, 217
scissors, 217
shaderProgram, 217
stencil, 218
viewports, 218
LLGL::Image, 218
Blit, 221
Convert, 222
Fill, 222
GetBytesPerPixel, 222
GetData, 223
GetDataSize, 223
GetDataType, 223
GetDepthStride, 223
GetDstDesc, 223
GetExtent, 223
GetFormat, 223
GetNumPixels, 223
GetRowStride, 224
GetSrcDesc, 224
Image, 221
IsRegionInside, 224
MirrorXYPlane, 224
MirrorXZPlane, 224
MirrorYZPlane, 224
operator=, 224, 225
ReadPixels, 225
Release, 225
Reset, 225, 226
Resize, 226, 227
Swap, 227
WritePixels, 227
LLGL::Input, 228
GetAnyKeyCount, 230
GetEnteredChars, 230
GetMouseMotion, 230
GetMousePosition, 230
GetWheelMotion, 230
Input, 230
KeyDoubleClick, 230
KeyDown, 231
KeyDownRepeated, 231
KeyPressed, 231
KeyUp, 231
OnChar, 231
OnDoubleClick, 231
OnGlobalMotion, 231
OnKeyDown, 231
OnKeyUp, 232
OnLocalMotion, 232
OnLostFocus, 232
OnProcessEvents, 232
OnWheelMotion, 232
LLGL::Interface, 233
IsInstanceOf, 234
LLGL::InterfaceID, 234
Buffer, 235
BufferArray, 235
Canvas, 236
Canvas_EventListener, 236
CommandBuffer, 235
CommandQueue, 235
ComputePipeline, 235
Display, 235
Fence, 235
GraphicsPipeline, 235
Input, 236
Interface, 235
PipelineLayout, 235
QueryHeap, 235
RenderContext, 235
RenderPass, 235
RenderSystem, 235
RenderSystemChild, 235
RenderTarget, 235
Reserved, 236
Resource, 235
ResourceHeap, 235
Sampler, 235
Shader, 235
ShaderProgram, 235
Surface, 236
Texture, 235
Timer, 236
Window, 236
Window_EventListener, 236
LLGL::Log, 101
Error, 102
Information, 103
Performance, 103

PostReport, 103
ReportCallback, 102
ReportType, 102
SetReportCallback, 103
SetReportCallbackStd, 103
SetReportLimit, 104
Warning, 102
LLGL::MiscFlags, 239
DynamicUsage, 239
FixedSamples, 239
GenerateMips, 239
NoInitialData, 239
LLGL::MultiSamplingDescriptor, 240
enabled, 241
MultiSamplingDescriptor, 240
SampleCount, 241
sampleMask, 241
samples, 241
LLGL::NativeContextHandle, 241
colorMap, 242
display, 242
parentView, 242
parentWindow, 242
screen, 242
visual, 242
LLGL::NativeHandle, 243
display, 243
view, 243
visual, 243
window, 243
LLGL::NonCopyable, 244
~NonCopyable, 245
NonCopyable, 245
operator=, 245
LLGL::Offset2D, 245
Offset2D, 246
x, 246
y, 246
LLGL::Offset3D, 246
Offset3D, 247
x, 247
y, 247
z, 247
LLGL::OpenGLDependentStateDescriptor, 247
invertFrontFace, 248
originLowerLeft, 248
LLGL::PipelineLayout, 248
LLGL::PipelineLayoutDescriptor, 249
bindings, 249
LLGL::QueryHeap, 250
GetType, 250
QueryHeap, 250
LLGL::QueryHeapDescriptor, 251
numQueries, 251
renderCondition, 251
type, 251
LLGL::QueryPipelineStatistics, 252
clippingInvocations, 253
clippingPrimitives, 253
computeShaderInvocations, 253
fragmentShaderInvocations, 253
geometryShaderInvocations, 253
geometryShaderPrimitives, 253
inputAssemblyPrimitives, 254
inputAssemblyVertices, 254
tessControlShaderInvocations, 254
tessEvaluationShaderInvocations, 254
vertexShaderInvocations, 254
LLGL::RasterizerDescriptor, 254
antiAliasedLineEnabled, 255
conservativeRasterization, 255
cullMode, 255
depthBias, 256
depthClampEnabled, 256
discardEnabled, 256
frontCCW, 256
lineWidth, 256
multiSampling, 256
polygonMode, 257
scissorTestEnabled, 257
LLGL::RenderContext, 257
GetColorFormat, 259
GetDepthStencilFormat, 259
GetNumColorAttachments, 260
GetResolution, 260
GetSurface, 260
GetVideoMode, 260
GetVsync, 260
HasDepthAttachment, 261
HasStencilAttachment, 261
OnSetVideoMode, 261
OnSetVsync, 262
Present, 262
RenderContext, 259
SetDisplayModeByVideoMode, 262
SetOrCreateSurface, 262
SetVideoMode, 263
SetVsync, 263
ShareSurfaceAndConfig, 263
SwitchFullscreenMode, 264
LLGL::RenderContextDescriptor, 264
multiSampling, 264
videoMode, 264
vsync, 265
LLGL::RenderPass, 287
LLGL::RenderPassDescriptor, 288
colorAttachments, 289
depthAttachment, 289
stencilAttachment, 289
LLGL::RenderSystem, 290
AssertCreateBuffer, 293
AssertCreateBufferArray, 293
AssertCreateRenderPass, 293
AssertCreateRenderTarget, 293
AssertCreateShader, 293
AssertCreateShaderProgram, 293

AssertImageDataSize, 294
CreateBuffer, 294
CreateBufferArray, 294
CreateCommandBuffer, 295
CreateComputePipeline, 295
CreateFence, 295
CreateGraphicsPipeline, 295
CreatePipelineLayout, 296
CreateQueryHeap, 296
CreateRenderContext, 296
CreateRenderPass, 297
CreateRenderTarget, 297
CreateResourceHeap, 297
CreateSampler, 298
CreateShader, 298
CreateShaderProgram, 298
CreateTexture, 299
FindModules, 299
GetCommandQueue, 299
GetConfiguration, 299
GetName, 299
GetRendererID, 300
GetRendererInfo, 300
GetRenderingCaps, 300
Load, 300
MapBuffer, 301
ReadTexture, 301
Release, 302–304
RenderSystem, 293
SetConfiguration, 304
SetRendererInfo, 304
SetRenderingCaps, 304
Unload, 305
UnmapBuffer, 305
WriteBuffer, 305
WriteTexture, 305
LLGL::RenderSystemChild, 306
SetName, 307
LLGL::RenderSystemConfiguration, 307
threadCount, 308
LLGL::RenderSystemDescriptor, 308
debugCallback, 309
moduleName, 309
operator=, 309
RenderSystemDescriptor, 309
rendererConfig, 310
rendererConfigSize, 310
LLGL::RenderTarget, 311
GetNumColorAttachments, 312
GetRenderPass, 312
GetResolution, 312
HasDepthAttachment, 313
HasStencilAttachment, 313
IsRenderContext, 313
RenderContext, 314
ValidateMipResolution, 314
ValidateResolution, 314
LLGL::RenderTargetDescriptor, 314
attachments, 315
customMultiSampling, 315
multiSampling, 316
renderPass, 316
resolution, 316
LLGL::RendererConfigurationOpenGL, 265
contextProfile, 265
majorVersion, 265
minorVersion, 266
LLGL::RendererConfigurationVulkan, 266
application, 267
enabledLayers, 267
minDeviceMemoryAllocationSize, 267
reduceDeviceMemoryFragmentation, 267
LLGL::RendererID, 268
Direct3D10, 269
Direct3D11, 269
Direct3D12, 269
Direct3D9, 269
Metal, 269
OpenGLES1, 269
OpenGLES2, 269
OpenGLES3, 269
OpenGL, 269
Reserved, 269
Undefined, 269
Vulkan, 270
LLGL::RendererInfo, 270
deviceName, 270
rendererName, 270
shadingLanguageName, 270
vendorName, 271
LLGL::RenderingCapabilities, 271
clippingRange, 271
features, 271
limits, 272
screenOrigin, 272
shadingLanguages, 272
textureFormats, 272
LLGL::RenderingDebugger, 273
~RenderingDebugger, 274
OnError, 274
OnWarning, 274
PostError, 274
PostWarning, 274
SetDebugGroup, 275
SetSource, 275
LLGL::RenderingDebugger::Message, 236
Block, 237
BlockAfter, 237
GetGroupName, 237
GetOccurrences, 238
GetSource, 238
GetText, 238
IncOccurrence, 238
IsBlocked, 238
Message, 237
operator=, 238

RenderingDebugger, 238
ToReportString, 238
LLGL::RenderingFeatures, 275
has3DTextures, 277
hasArrayTextures, 277
hasComputeShaders, 277
hasConservativeRasterization, 277
hasConstantBuffers, 277
hasCubeArrayTextures, 278
hasCubeTextures, 278
hasDirectResourceBinding, 278
hasGeometryShaders, 278
hasIndirectDrawing, 278
hasInstancing, 279
hasLogicOp, 279
hasMultiSampleTextures, 279
hasOffsetInstancing, 279
hasPipelineStatistics, 279
hasRenderCondition, 280
hasRenderTargets, 280
hasSamplers, 280
hasStorageBuffers, 280
hasStreamOutputs, 280
hasTessellationShaders, 280
hasTextureViewSwizzle, 281
hasTextureViews, 281
hasUniforms, 281
hasViewportArrays, 281
LLGL::RenderingLimits, 282
lineWidthRange, 283
max1DTextureSize, 283
max2DTextureSize, 283
max3DTextureSize, 283
maxAnisotropy, 283
maxBufferSize, 284
maxColorAttachments, 284
maxComputeShaderWorkGroupSize, 284
maxComputeShaderWorkGroups, 284
maxConstantBufferSize, 284
maxCubeTextureSize, 285
maxPatchVertices, 285
maxTextureArrayLayers, 285
maxViewportSize, 285
maxViewports, 285
LLGL::RenderingProfiler, 286
Accumulate, 286
frameProfile, 287
NextProfile, 287
LLGL::Resource, 317
GetResourceType, 317
LLGL::ResourceHeap, 318
LLGL::ResourceHeapDescriptor, 318
pipelineLayout, 319
resourceViews, 319
LLGL::ResourceViewDescriptor, 320
resource, 321
ResourceViewDescriptor, 320
textureView, 321
LLGL::Sampler, 321
GetResourceType, 322
Sampler, 322
LLGL::SamplerDescriptor, 322
addressModeU, 323
addressModeV, 323
addressModeW, 323
borderColor, 324
compareEnabled, 324
compareOp, 324
magFilter, 324
maxAnisotropy, 324
maxLOD, 324
minFilter, 324
minLOD, 324
mipMapFilter, 324
mipMapLODBias, 325
mipMapping, 325
LLGL::Scissor, 325
height, 326
Scissor, 326
width, 326
x, 327
y, 327
LLGL::Shader, 327
GetReport, 328
GetStageFlags, 328
GetType, 328
HasErrors, 328
Shader, 328
LLGL::ShaderCompileFlags, 329
Debug, 329
O1, 329
O2, 329
O3, 329
WarnError, 329
LLGL::ShaderDescriptor, 330
defines, 331
entryPoint, 331
flags, 332
operator=, 331
profile, 332
ShaderDescriptor, 331
source, 332
sourceSize, 333
sourceType, 333
streamOutput, 333
type, 333
LLGL::ShaderDescriptor::StreamOutput, 353
format, 354
LLGL::ShaderMacro, 334
definition, 335
name, 335
operator=, 335
ShaderMacro, 334
LLGL::ShaderProgram, 335
ClearShaderReflection, 337
FinalizeShaderReflection, 337

FindUniformLocation, 337
GetReport, 337
GetWorkGroupSize, 338
HasErrors, 338
IncompleteAttachments, 336
InvalidByteCode, 336
InvalidComposition, 336
LinkError, 336
LinkErrorToString, 338
NoError, 336
Reflect, 338
SetWorkGroupSize, 339
TooManyAttachments, 336
ValidateShaderComposition, 340
LLGL::ShaderProgramDescriptor, 340
computeShader, 341
fragmentShader, 341
geometryShader, 341
tessControlShader, 342
tessEvaluationShader, 342
vertexFormats, 342
vertexShader, 342
LLGL::ShaderReflection, 343
resources, 344
streamOutputAttributes, 344
uniforms, 344
vertexAttributes, 344
LLGL::ShaderResource, 344
binding, 345
constantBufferSize, 345
storageBufferType, 345
LLGL::ShaderUniform, 346
location, 346
name, 346
size, 346
type, 346
LLGL::SrcImageDescriptor, 347
data, 348
dataSize, 348
dataType, 348
format, 348
SrcImageDescriptor, 347
LLGL::StageFlags, 348
AllGraphicsStages, 349
AllStages, 349
AllTessStages, 349
ComputeStage, 349
FragmentStage, 349
GeometryStage, 349
TessControlStage, 349
TessEvaluationStage, 349
VertexStage, 349
LLGL::StencilDescriptor, 349
back, 350
front, 350
testEnabled, 350
LLGL::StencilFaceDescriptor, 350
compareOp, 351
depthFailOp, 351
depthPassOp, 351
readMask, 351
reference, 351
stencilFailOp, 351
writeMask, 352
LLGL::StreamOutputAttribute, 354
components, 355
name, 355
operator=, 355
outputSlot, 355
semanticIndex, 355
startComponent, 356
stream, 356
StreamOutputAttribute, 355
LLGL::StreamOutputFormat, 356
AppendAttribute, 356
AppendAttributes, 357
attributes, 357
LLGL::Surface, 357
AdaptForVideoMode, 358
GetContentSize, 358
GetNativeHandle, 359
ProcessEvents, 359
ResetPixelFormat, 359
LLGL::Texture, 360
GetDesc, 361
GetMipExtent, 361
GetResourceType, 362
GetType, 362
Texture, 361
LLGL::TextureDescriptor, 362
arrayLayers, 363
bindFlags, 363
clearValue, 363
cpuAccessFlags, 364
extent, 364
format, 364
mipLevels, 365
miscFlags, 365
samples, 365
type, 365
LLGL::TextureLocation, 366
arrayLayer, 367
mipLevel, 367
offset, 367
TextureLocation, 367
LLGL::TextureRegion, 368
extent, 369
offset, 369
subresource, 369
TextureRegion, 368
LLGL::TextureSubresource, 369
baseArrayLayer, 371
baseMipLevel, 371
numArrayLayers, 371
numMipLevels, 371
TextureSubresource, 370

LLGL::TextureSwizzleRGBA, 372
a, 372
b, 372
g, 372
r, 372
LLGL::TextureViewDescriptor, 373
format, 373
subresource, 374
swizzle, 374
type, 374
LLGL::Timer, 375
Create, 376
GetDeltaTime, 376
GetFrequency, 376
IsRunning, 376
MeasureTime, 376
Start, 377
Stop, 377
LLGL::UninitializeTag, 377
LLGL::Version, 104
GetID, 104
GetMajor, 104
GetMinor, 105
GetRevision, 105
GetStatus, 105
GetString, 105
LLGL::VertexAttribute, 378
format, 380
GetSize, 379
instanceDivisor, 380
name, 380
offset, 380
operator=, 379
semanticIndex, 380
systemValue, 380
VertexAttribute, 378, 379
LLGL::VertexFormat, 382
AppendAttribute, 382
attributes, 383
inputSlot, 383
stride, 383
LLGL::VideoAdapterDescriptor, 384
name, 384
outputs, 384
vendor, 384
videoMemory, 384
LLGL::VideoModeDescriptor, 385
colorBits, 386
depthBits, 386
fullscreen, 386
resolution, 386
stencilBits, 386
swapChainSize, 387
LLGL::VideoOutputDescriptor, 387
displayModes, 388
LLGL::Viewport, 388
height, 390
maxDepth, 390
minDepth, 390
Viewport, 389
width, 390
x, 390
y, 390
LLGL::VsyncDescriptor, 391
enabled, 391
interval, 391
refreshRate, 391
LLGL::Window, 392
AdaptForVideoMode, 394
AddEventListener, 394
Create, 394
GetBehavior, 394
GetDesc, 394
GetPosition, 394
GetSize, 395
GetTitle, 395
HasFocus, 395
HasQuit, 395
IsShown, 395
OnProcessEvents, 395
PostChar, 395
PostDoubleClick, 395
PostGetFocus, 396
PostGlobalMotion, 396
PostKeyDown, 396
PostKeyUp, 396
PostLocalMotion, 396
PostLostFocus, 396
PostQuit, 396
PostResize, 397
PostTimer, 397
PostWheelMotion, 397
ProcessEvents, 397
RemoveEventListener, 397
SetBehavior, 398
SetDesc, 398
SetPosition, 398
SetSize, 398
SetTitle, 398
Show, 398
LLGL::Window::EventListener, 194
OnChar, 196
OnDoubleClick, 196
OnGetFocus, 196
OnGlobalMotion, 196
OnKeyDown, 196
OnKeyUp, 196
OnLocalMotion, 196
OnLostFocus, 196
OnProcessEvents, 197
OnQuit, 197
OnResize, 197
OnTimer, 197
OnWheelMotion, 197
Window, 198
LLGL::WindowBehavior, 399

disableClearOnResize, 399
moveAndResizeTimerID, 399
LLGL::WindowDescriptor, 400
acceptDropFiles, 400
borderless, 400
centered, 400
position, 401
preventForPowerSafe, 401
resizable, 401
size, 401
title, 402
visible, 402
windowContext, 402
LLGL_API_CDECL
Platform.h, 425
LLGL_API_STDCALL
Platform.h, 425
LLGL_API_THISCALL
Platform.h, 425
LLGL_API
Platform.h, 425
LLGL_DECLARE_INTERFACE
Interface.h, 420
LLGL_EXPORT
Export.h, 411
LLGL_IMPLEMENT_INTERFACE
Interface.h, 420
LLGL, 51
A, 79
A8UNorm, 74
ABGR, 77
ANDInverted, 82
ANDReverse, 82
AND, 82
ARGB, 77
Add, 69
Alpha, 77, 93
AlwaysPass, 70
Any, 81
AnySamplesPassed, 85
AnySamplesPassedConservative, 85
AppendStructuredBuffer, 91
Apps, 79
AtomicCounter, 95
AttachmentLoadOp, 67
AttachmentStoreOp, 67
AttachmentType, 68
Attn, 81
B, 79
BC1RGBA, 77
BC1RGB, 77
BC2RGBA, 77
BC3RGBA, 77
BGRA8SInt, 77
BGRA8SNorm, 76
BGRA8UInt, 76
BGRA8UNorm, 76
BGRA8UNorm_sRGB, 76
BGRA, 77
BGR, 77
Back, 71, 78
BinaryBuffer, 89
BinaryFile, 89
BlendArithmetic, 68
BlendFactor, 69
BlendOp, 69
Blue, 93
Bool1, 94
Bool2, 94
Bool3, 94
Bool4, 94
Border, 87
BrowserBack, 80
BrowserFavorits, 80
BrowserForward, 80
BrowserHome, 80
BrowserRefresh, 80
BrowserSearch, 80
BrowserStop, 80
Buffer, 86, 91
ByRegionNoWait, 86
ByRegionNoWaitInverted, 86
ByRegionWait, 86
ByRegionWaitInverted, 86
ByteAddressBuffer, 91
ByteBuffer, 66
C, 79
CPUAccess, 70
Cancel, 78
Capital, 78
CastColorValue, 95
CastColorValue< bool, bool >, 95
CastColorValue< double, double >, 96
CastColorValue< float, float >, 96
CastColorValue< std::uint8_t, std::uint8_t >, 96
CastTo, 96, 97
Clamp, 87
Clear, 67, 78, 82
ClipDistance, 93
ClippingRange, 69
CodeFile, 89
CodeString, 89
Color, 68, 93
ColorRGBAb, 67
ColorRGBAd, 67
ColorRGBAf, 67
ColorRGBAT, 67
ColorRGBAub, 67
ColorRGBb, 67
ColorRGBd, 67
ColorRGBf, 67
ColorRGBT, 67
ColorRGBub, 67
Comma, 81
CompareOp, 70
CompatibilityProfile, 82

CompressedRGBA, 77
CompressedRGB, 77
Compute, 89
ConsumeStructuredBuffer, 91
Control, 78
Copy, 82
CopyInverted, 82
CoreProfile, 82
CrSel, 81
CullDistance, 93
CullMode, 71
D, 79
D0, 78
D1, 78
D16UNorm, 77
D2, 78
D24UNormS8UInt, 77
D3, 78
D32Float, 77
D32FloatS8X24UInt, 77
D4, 78
D5, 78
D6, 79
D7, 79
D8, 79
D9, 79
DataType, 71
DecClamp, 91
DecWrap, 91
Delete, 78
Depth, 68, 77, 93
DepthGreater, 93
DepthLess, 93
DepthStencil, 68, 77
Disabled, 71, 82
Double1, 94
Double2, 94
Double2x2, 95
Double2x3, 95
Double2x4, 95
Double3, 94
Double3x2, 95
Double3x3, 95
Double3x4, 95
Double4, 94
Double4x2, 95
Double4x3, 95
Double4x4, 95
Down, 78
DstAlpha, 69
DstColor, 69
E, 79
ESProfile, 82
ESSL_100, 90
ESSL_300, 90
ESSL_310, 90
ESSL_320, 90
ESSL, 90
End, 78
Equal, 70
Equiv, 82
ErEOF, 81
ErrorType, 72
Escape, 78
ExSel, 81
Exe, 78
Exponent, 81
F, 79
F1, 80
F10, 80
F11, 80
F12, 80
F13, 80
F14, 80
F15, 80
F16, 80
F17, 80
F18, 80
F19, 80
F2, 80
F20, 80
F21, 80
F22, 80
F23, 80
F24, 80
F3, 80
F4, 80
F5, 80
F6, 80
F7, 80
F8, 80
F9, 80
Fill, 82
Float1, 94
Float16, 72
Float2, 94
Float2x2, 95
Float2x3, 95
Float2x4, 95
Float3, 94
Float32, 72
Float3x2, 95
Float3x3, 95
Float3x4, 95
Float4, 94
Float4x2, 95
Float4x3, 95
Float4x4, 95
Float64, 72
Format, 72
Fragment, 89
Front, 71
FrontFacing, 93
G, 79
GLSL_110, 90
GLSL_120, 90

GLSL_130, 90
GLSL_140, 90
GLSL_150, 90
GLSL_330, 90
GLSL_400, 90
GLSL_410, 90
GLSL_420, 90
GLSL_430, 90
GLSL_440, 90
GLSL_450, 90
GLSL_460, 90
GLSL, 90
Geometry, 89
GetExtentRatio, 97
GetPrimitiveTopologyPatchSize, 97
Greater, 70
GreaterEqual, 70
Green, 93
H, 79
HLSL_2_0, 90
HLSL_2_0a, 90
HLSL_2_0b, 90
HLSL_3_0, 90
HLSL_4_0, 90
HLSL_4_1, 90
HLSL_5_0, 90
HLSL_5_1, 90
HLSL, 90
Help, 78
Home, 78
I, 79
Image, 95
ImageFormat, 77
ImproperArgument, 95
ImproperState, 95
IncClamp, 91
IncWrap, 91
Insert, 78
InstanceID, 93
Int1, 94
Int16, 72
Int2, 94
Int3, 94
Int32, 72
Int4, 94
Int8, 72
InvBlendFactor, 69
InvDstAlpha, 69
InvDstColor, 69
InvSrc1Alpha, 69
InvSrc1Color, 69
InvSrcAlpha, 69
InvSrcColor, 69
InvalidArgument, 72
InvalidState, 72
Invert, 82, 91
IsPrimitiveTopologyPatches, 97
IsShaderSourceBinary, 97
IsShaderSourceCode, 98
J, 79
K, 79
Keep, 91
Key, 77
Keypad0, 79
Keypad1, 79
Keypad2, 79
Keypad3, 79
Keypad4, 79
Keypad5, 79
Keypad6, 79
Keypad7, 79
Keypad8, 79
Keypad9, 79
KeypadDecimal, 80
KeypadDivide, 80
KeypadMinus, 80
KeypadMultiply, 79
KeypadPlus, 80
KeypadSeparator, 80
L, 79
LButton, 78
LControl, 80
LMenu, 80
LShift, 80
LWin, 79
LaunchApp1, 81
LaunchApp2, 81
LaunchMail, 81
LaunchMediaSelect, 81
Left, 78
Less, 70
LessEqual, 70
LineList, 83
LineListAdjacency, 83
LineLoop, 83
LineStrip, 83
LineStripAdjacency, 83
Linear, 88
Lines, 84
Load, 67
LogicOp, 81
LowerLeft, 88
M, 79
MButton, 78
Max, 69
MaxColorValue, 98
MaxColorValue< bool >, 98
MaxColorValue< std::uint8_t >, 98
MediaNextTrack, 81
MediaPlayPause, 81
MediaPrevTrack, 81
MediaStop, 81
Menu, 78
Metal, 90
Metal_1_0, 90
Metal_1_1, 90

Metal_1_2, 90
Metal_2_0, 90
Metal_2_1, 90
Min, 69
Minus, 81
MinusOneToOne, 70
Mirror, 87
MirrorOnce, 87
N, 79
NAND, 82
NOR, 82
Nearest, 88
NeverPass, 70
NoName, 81
NoOp, 82
NoWait, 86
NoWaitInverted, 86
NotEqual, 70
NumLock, 80
O, 79
OEMClear, 81
ORInverted, 82
ORReverse, 82
One, 69, 93
OpenGLContextProfile, 82
operator!=, 98, 99
operator∗, 99
operator+, 99
operator-, 99
operator/, 99
operator==, 99, 100
OR, 82
P, 79
PA1, 81
PageDown, 78
PageUp, 78
Patches1, 83
Patches10, 84
Patches11, 84
Patches12, 84
Patches13, 84
Patches14, 84
Patches15, 84
Patches16, 84
Patches17, 84
Patches18, 84
Patches19, 84
Patches2, 83
Patches20, 84
Patches21, 84
Patches22, 84
Patches23, 84
Patches24, 84
Patches25, 84
Patches26, 84
Patches27, 84
Patches28, 84
Patches29, 84
Patches3, 83
Patches30, 84
Patches31, 84
Patches32, 84
Patches4, 83
Patches5, 83
Patches6, 83
Patches7, 83
Patches8, 84
Patches9, 84
Pause, 78
Period, 81
PipelineStatistics, 85
Play, 81
Plus, 81
PointList, 83
PointlessOperation, 95
Points, 82, 84
PolygonMode, 82
Position, 93
PrimitiveID, 93
PrimitiveTopology, 82
PrimitiveType, 84
Print, 78
Q, 79
QueryType, 85
R, 77, 79
R16Float, 74
R16SInt, 74
R16SNorm, 74
R16UInt, 74
R16UNorm, 74
R32Float, 74
R32SInt, 74
R32UInt, 74
R64Float, 74
R8SInt, 74
R8SNorm, 74
R8UInt, 74
R8UNorm, 74
RButton, 78
RControl, 80
RG11B10Float, 77
RG16Float, 75
RG16SInt, 75
RG16SNorm, 75
RG16UInt, 75
RG16UNorm, 75
RG32Float, 75
RG32SInt, 75
RG32UInt, 75
RG64Float, 75
RG8SInt, 75
RG8SNorm, 75
RG8UInt, 75
RG8UNorm, 75
RGB10A2UInt, 77
RGB10A2UNorm, 77

RGB16Float, 76
RGB16SInt, 75
RGB16SNorm, 75
RGB16UInt, 75
RGB16UNorm, 75
RGB32Float, 76
RGB32SInt, 76
RGB32UInt, 76
RGB64Float, 76
RGB8SInt, 75
RGB8SNorm, 75
RGB8UInt, 75
RGB8UNorm, 75
RGB8UNorm_sRGB, 75
RGB9E5Float, 77
RGBA16Float, 76
RGBA16SInt, 76
RGBA16SNorm, 76
RGBA16UInt, 76
RGBA16UNorm, 76
RGBA32Float, 76
RGBA32SInt, 76
RGBA32UInt, 76
RGBA64Float, 76
RGBA8SInt, 76
RGBA8SNorm, 76
RGBA8UInt, 76
RGBA8UNorm, 76
RGBA8UNorm_sRGB, 76
RGBA, 77
RGB, 77
RMenu, 80
RShift, 80
RWBuffer, 91
RWByteAddressBuffer, 91
RWStructuredBuffer, 91
RWin, 79
ReadOnly, 71
ReadWrite, 71
Red, 93
RenderConditionMode, 85
RenderTargetIndex, 93
Repeat, 87
Replace, 91
ResourceType, 86
Return, 78
RevSubtract, 69
RG, 77
Right, 78
S, 79
SPIRV_100, 90
SPIRV, 90
SampleID, 93
SampleMask, 93
Sampler, 86, 95
SamplerAddressMode, 86
SamplerFilter, 88
SamplesPassed, 85
ScreenOrigin, 88
ScrollLock, 80
Select, 78
Set, 82
ShaderSourceType, 89
ShaderType, 89
ShadingLanguage, 89
Shift, 78
Sleep, 79
Snapshot, 78
Space, 78
Src1Alpha, 69
Src1Color, 69
SrcAlpha, 69
SrcAlphaSaturate, 69
SrcColor, 69
Stencil, 68, 93
StencilOp, 90
StorageBufferType, 91
Store, 68
StreamOutOverflow, 85
StreamOutPrimitivesWritten, 85
StructuredBuffer, 91
Subtract, 69
SystemValue, 91
T, 79
Tab, 78
TessControl, 89
TessEvaluation, 89
Texture, 86
Texture1DArray, 94
Texture1D, 94
Texture2DArray, 94
Texture2DMSArray, 94
Texture2DMS, 94
Texture2D, 94
Texture3D, 94
TextureCube, 94
TextureCubeArray, 94
TextureSwizzle, 93
TextureType, 93
TimeElapsed, 85
TriangleFan, 83
TriangleList, 83
TriangleListAdjacency, 83
TriangleStrip, 83
TriangleStripAdjacency, 83
Triangles, 85
U, 79
UInt1, 94
UInt16, 72
UInt2, 94
UInt3, 94
UInt32, 72
UInt4, 94
UInt8, 72
Undefined, 67, 68, 72, 74, 86, 89, 91, 92, 94
UndefinedBehavior, 72

UniformLocation, 67
UniformType, 94
UnsupportedFeature, 72
Up, 78
UpperLeft, 89
V, 79
ValidateRenderingCaps, 100
VaryingBehavior, 95
VersionBitmask, 90
Vertex, 89
VertexID, 93
ViewportIndex, 93
VolumeDown, 81
VolumeMute, 80
VolumeUp, 81
W, 79
Wait, 86
WaitInverted, 86
WarningType, 95
Wireframe, 82
WriteDiscard, 71
WriteOnly, 71
X, 79
XButton1, 78
XButton2, 78
XOR, 82
Y, 79
Z, 79
Zero, 69, 91, 93
ZeroToOne, 70
Zoom, 81
LMenu
LLGL, 80
LShift
LLGL, 80
LWin
LLGL, 79
LaunchApp1
LLGL, 81
LaunchApp2
LLGL, 81
LaunchMail
LLGL, 81
LaunchMediaSelect
LLGL, 81
Left
LLGL, 78
Less
LLGL, 70
LessEqual
LLGL, 70
limits
LLGL::RenderingCapabilities, 272
LineList
LLGL, 83
LineListAdjacency
LLGL, 83
LineLoop
LLGL, 83
LineStrip
LLGL, 83
LineStripAdjacency
LLGL, 83
lineWidth
LLGL::RasterizerDescriptor, 256
lineWidthRange
LLGL::RenderingLimits, 283
Linear
LLGL, 88
Lines
LLGL, 84
LinkError
LLGL::ShaderProgram, 336
LinkErrorToString
LLGL::ShaderProgram, 338
LinuxNativeHandle.h, 422
Load
LLGL::RenderSystem, 300
LLGL, 67
loadOp
LLGL::AttachmentFormatDescriptor, 113
location
LLGL::ShaderUniform, 346
Log.h, 422
LogicOp
LLGL, 81
logicOp
LLGL::BlendDescriptor, 120
LowerLeft
LLGL, 88

M
LLGL, 79
MButton
LLGL, 78
MacOSNativeHandle.h, 423
magFilter
LLGL::SamplerDescriptor, 324
majorVersion
LLGL::RendererConfigurationOpenGL, 265
MapBuffer
LLGL::RenderSystem, 301
Max
LLGL, 69
max1DTextureSize
LLGL::RenderingLimits, 283
max2DTextureSize
LLGL::RenderingLimits, 283
max3DTextureSize
LLGL::RenderingLimits, 283
maxAnisotropy
LLGL::RenderingLimits, 283
LLGL::SamplerDescriptor, 324
maxBufferSize
LLGL::RenderingLimits, 284
maxColorAttachments
LLGL::RenderingLimits, 284

MaxColorValue
LLGL, 98
MaxColorValue< bool >
LLGL, 98
MaxColorValue< std::uint8_t >
LLGL, 98
maxComputeShaderWorkGroupSize
LLGL::RenderingLimits, 284
maxComputeShaderWorkGroups
LLGL::RenderingLimits, 284
maxConstantBufferSize
LLGL::RenderingLimits, 284
maxCubeTextureSize
LLGL::RenderingLimits, 285
maxDepth
LLGL::Viewport, 390
maxLOD
LLGL::SamplerDescriptor, 324
maxPatchVertices
LLGL::RenderingLimits, 285
maxTextureArrayLayers
LLGL::RenderingLimits, 285
maxViewportSize
LLGL::RenderingLimits, 285
maxViewports
LLGL::RenderingLimits, 285
MeasureTime
LLGL::Timer, 376
MediaNextTrack
LLGL, 81
MediaPlayPause
LLGL, 81
MediaPrevTrack
LLGL, 81
MediaStop
LLGL, 81
Menu
LLGL, 78
Message
LLGL::RenderingDebugger::Message, 237
Metal
LLGL::RendererID, 269
LLGL, 90
Metal_1_0
LLGL, 90
Metal_1_1
LLGL, 90
Metal_1_2
LLGL, 90
Metal_2_0
LLGL, 90
Metal_2_1
LLGL, 90
Min
LLGL, 69
minDepth
LLGL::Viewport, 390
minDeviceMemoryAllocationSize
LLGL::RendererConfigurationVulkan, 267
minFilter
LLGL::SamplerDescriptor, 324
minLOD
LLGL::SamplerDescriptor, 324
minorVersion
LLGL::RendererConfigurationOpenGL, 266
Minus
LLGL, 81
MinusOneToOne
LLGL, 70
mipLevel
LLGL::AttachmentDescriptor, 111
LLGL::TextureLocation, 367
mipLevels
LLGL::TextureDescriptor, 365
mipMapFilter
LLGL::SamplerDescriptor, 324
mipMapLODBias
LLGL::SamplerDescriptor, 325
mipMapping
LLGL::SamplerDescriptor, 325
mipMapsGenerations
LLGL::FrameProfile, 211
Mirror
LLGL, 87
MirrorOnce
LLGL, 87
MirrorXYPlane
LLGL::Image, 224
MirrorXZPlane
LLGL::Image, 224
MirrorYZPlane
LLGL::Image, 224
miscFlags
LLGL::BufferDescriptor, 127
LLGL::TextureDescriptor, 365
moduleName
LLGL::RenderSystemDescriptor, 309
moveAndResizeTimerID
LLGL::WindowBehavior, 399
multiSampling
LLGL::RasterizerDescriptor, 256
LLGL::RenderContextDescriptor, 264
LLGL::RenderTargetDescriptor, 316
MultiSamplingDescriptor
LLGL::MultiSamplingDescriptor, 240
MultiSubmit
LLGL::CommandBufferFlags, 174

N
LLGL, 79
NAND
LLGL, 82
NOR
LLGL, 82
name
LLGL::BindingDescriptor, 118
LLGL::FragmentAttribute, 205

LLGL::ShaderMacro, 335
LLGL::ShaderUniform, 346
LLGL::StreamOutputAttribute, 355
LLGL::VertexAttribute, 380
LLGL::VideoAdapterDescriptor, 384
NativeHandle.h, 423
Nearest
LLGL, 88
NeverPass
LLGL, 70
NextProfile
LLGL::RenderingProfiler, 287
NoError
LLGL::ShaderProgram, 336
NoInitialData
LLGL::MiscFlags, 239
NoName
LLGL, 81
NoOp
LLGL, 82
NoWait
LLGL, 86
NoWaitInverted
LLGL, 86
NonCopyable
LLGL::NonCopyable, 245
NonCopyable.h, 424
normalized
LLGL::FormatDescriptor, 204
NotEqual
LLGL, 70
numArrayLayers
LLGL::TextureSubresource, 371
numIndices
LLGL::DrawIndexedIndirectArguments, 189
numInstances
LLGL::DrawIndexedIndirectArguments, 189
LLGL::DrawIndirectArguments, 191
LLGL::DrawPatchIndirectArgument, 192
NumLock
LLGL, 80
NumMipLevels
Texture utility functions to determine texture dimension and buffer sizes., 41, 42
numMipLevels
LLGL::TextureSubresource, 371
numNativeBuffers
LLGL::CommandBufferDescriptor, 173
numPatches
LLGL::DrawPatchIndirectArgument, 192
numQueries
LLGL::QueryHeapDescriptor, 251
numThreadGroups
LLGL::DispatchIndirectArguments, 183
numVertices
LLGL::DrawIndirectArguments, 191

O
LLGL, 79
O1
LLGL::ShaderCompileFlags, 329
O2
LLGL::ShaderCompileFlags, 329
O3
LLGL::ShaderCompileFlags, 329
OEMClear
LLGL, 81
ORInverted
LLGL, 82
ORReverse
LLGL, 82
offset
LLGL::TextureLocation, 367
LLGL::TextureRegion, 369
LLGL::VertexAttribute, 380
Offset2D
LLGL::Offset2D, 246
Offset3D
LLGL::Offset3D, 247
OnChar
LLGL::Input, 231
LLGL::Window::EventListener, 196
OnDoubleClick
LLGL::Input, 231
LLGL::Window::EventListener, 196
OnError
LLGL::RenderingDebugger, 274
OnGetFocus
LLGL::Window::EventListener, 196
OnGlobalMotion
LLGL::Input, 231
LLGL::Window::EventListener, 196
OnKeyDown
LLGL::Input, 231
LLGL::Window::EventListener, 196
OnKeyUp
LLGL::Input, 232
LLGL::Window::EventListener, 196
OnLocalMotion
LLGL::Input, 232
LLGL::Window::EventListener, 196
OnLostFocus
LLGL::Input, 232
LLGL::Window::EventListener, 196
OnProcessEvents
LLGL::Canvas, 130
LLGL::Canvas::EventListener, 199
LLGL::Input, 232
LLGL::Window, 395
LLGL::Window::EventListener, 197
OnQuit
LLGL::Canvas::EventListener, 199
LLGL::Window::EventListener, 197
OnResize
LLGL::Window::EventListener, 197
OnSetVideoMode
LLGL::RenderContext, 261

OnSetVsync
LLGL::RenderContext, 262
OnTimer
LLGL::Window::EventListener, 197
OnWarning
LLGL::RenderingDebugger, 274
OnWheelMotion
LLGL::Input, 232
LLGL::Window::EventListener, 197
One
LLGL, 69, 93
OpenGLContextProfile
LLGL, 82
OpenGLES1
LLGL::RendererID, 269
OpenGLES2
LLGL::RendererID, 269
OpenGLES3
LLGL::RendererID, 269
OpenGL
LLGL::RendererID, 269
operator!=
Global operators for basic data structures., 22, 23
LLGL, 98, 99
operator∗
LLGL, 99
operator∗=
LLGL::Color, 136
LLGL::Color< T, 3u >, 140
LLGL::Color< T, 4u >, 145
operator+
Global operators for basic data structures., 23
LLGL, 99
operator+=
LLGL::Color, 136
LLGL::Color< T, 3u >, 140
LLGL::Color< T, 4u >, 145
operator-
Global operators for basic data structures., 23, 24
LLGL::Color, 136
LLGL::Color< T, 3u >, 140
LLGL::Color< T, 4u >, 145
LLGL, 99
operator-=
LLGL::Color, 136
LLGL::Color< T, 3u >, 140
LLGL::Color< T, 4u >, 145
operator/
LLGL, 99
operator/=
LLGL::Color, 136
LLGL::Color< T, 3u >, 140
LLGL::Color< T, 4u >, 145
operator=
LLGL::AttachmentClear, 109
LLGL::Image, 224, 225
LLGL::NonCopyable, 245
LLGL::RenderSystemDescriptor, 309
LLGL::RenderingDebugger::Message, 238
LLGL::ShaderDescriptor, 331
LLGL::ShaderMacro, 335
LLGL::StreamOutputAttribute, 355
LLGL::VertexAttribute, 379
operator==
Global operators for basic data structures., 24
LLGL, 99, 100
operator[]
LLGL::Color, 136, 137
LLGL::Color< T, 3u >, 140, 141
LLGL::Color< T, 4u >, 145, 146
OR
LLGL, 82
originLowerLeft
LLGL::OpenGLDependentStateDescriptor, 248
outputSlot
LLGL::StreamOutputAttribute, 355
outputs
LLGL::VideoAdapterDescriptor, 384

P
LLGL, 79
PA1
LLGL, 81
packed
LLGL::FormatDescriptor, 204
PageDown
LLGL, 78
PageUp
LLGL, 78
parentView
LLGL::NativeContextHandle, 242
parentWindow
LLGL::NativeContextHandle, 242
Patches1
LLGL, 83
Patches10
LLGL, 84
Patches11
LLGL, 84
Patches12
LLGL, 84
Patches13
LLGL, 84
Patches14
LLGL, 84
Patches15
LLGL, 84
Patches16
LLGL, 84
Patches17
LLGL, 84
Patches18
LLGL, 84
Patches19
LLGL, 84
Patches2
LLGL, 83

Patches20
LLGL, 84
Patches21
LLGL, 84
Patches22
LLGL, 84
Patches23
LLGL, 84
Patches24
LLGL, 84
Patches25
LLGL, 84
Patches26
LLGL, 84
Patches27
LLGL, 84
Patches28
LLGL, 84
Patches29
LLGL, 84
Patches3
LLGL, 83
Patches30
LLGL, 84
Patches31
LLGL, 84
Patches32
LLGL, 84
Patches4
LLGL, 83
Patches5
LLGL, 83
Patches6
LLGL, 83
Patches7
LLGL, 83
Patches8
LLGL, 84
Patches9
LLGL, 84
Pause
LLGL, 78
Performance
LLGL::Log, 103
Period
LLGL, 81
PipelineLayout
LLGL::InterfaceID, 235
pipelineLayout
LLGL::ComputePipelineDescriptor, 179
LLGL::GraphicsPipelineDescriptor, 217
LLGL::ResourceHeapDescriptor, 319
PipelineLayout.h, 424
PipelineLayoutDesc
Global utility functions, especially to fill descriptor structures., 45
PipelineLayoutFlags.h, 424
PipelineStatistics
LLGL, 85
Platform.h, 425
LLGL_API_CDECL, 425
LLGL_API_STDCALL, 425
LLGL_API_THISCALL, 425
LLGL_API, 425
Play
LLGL, 81
Plus
LLGL, 81
PointList
LLGL, 83
PointlessOperation
LLGL, 95
Points
LLGL, 82, 84
PolygonMode
LLGL, 82
polygonMode
LLGL::RasterizerDescriptor, 257
PopDebugGroup
LLGL::CommandBuffer, 162
Position
LLGL, 93
position
LLGL::WindowDescriptor, 401
PostChar
LLGL::Window, 395
PostDoubleClick
LLGL::Window, 395
PostError
LLGL::RenderingDebugger, 274
PostGetFocus
LLGL::Window, 396
PostGlobalMotion
LLGL::Window, 396
PostKeyDown
LLGL::Window, 396
PostKeyUp
LLGL::Window, 396
PostLocalMotion
LLGL::Window, 396
PostLostFocus
LLGL::Window, 396
PostQuit
LLGL::Canvas, 130
LLGL::Window, 396
PostReport
LLGL::Log, 103
PostResize
LLGL::Window, 397
PostTimer
LLGL::Window, 397
PostWarning
LLGL::RenderingDebugger, 274
PostWheelMotion
LLGL::Window, 397
Present

LLGL::RenderContext, 262
preventForPowerSafe
LLGL::WindowDescriptor, 401
PrimitiveID
LLGL, 93
PrimitiveTopology
LLGL, 82
primitiveTopology
LLGL::GraphicsPipelineDescriptor, 217
PrimitiveType
LLGL, 84
Print
LLGL, 78
ProcessEvents
LLGL::Canvas, 130
LLGL::Surface, 359
LLGL::Window, 397
profile
LLGL::ShaderDescriptor, 332
Ptr
LLGL::Color, 137
LLGL::Color< T, 3u >, 141
LLGL::Color< T, 4u >, 146
PushDebugGroup
LLGL::CommandBuffer, 162

Q
LLGL, 79
QueryHeap
LLGL::InterfaceID, 235
LLGL::QueryHeap, 250
QueryHeap.h, 425
QueryHeapFlags.h, 425
QueryResult
LLGL::CommandQueue, 176
querySections
LLGL::FrameProfile, 212
QueryType
LLGL, 85

R
LLGL, 77, 79
r
LLGL::Color< T, 3u >, 142
LLGL::Color< T, 4u >, 147
LLGL::TextureSwizzleRGBA, 372
R16Float
LLGL, 74
R16SInt
LLGL, 74
R16SNorm
LLGL, 74
R16UInt
LLGL, 74
R16UNorm
LLGL, 74
R32Float
LLGL, 74
R32SInt
LLGL, 74
R32UInt
LLGL, 74
R64Float
LLGL, 74
R8SInt
LLGL, 74
R8SNorm
LLGL, 74
R8UInt
LLGL, 74
R8UNorm
LLGL, 74
RButton
LLGL, 78
RControl
LLGL, 80
RG11B10Float
LLGL, 77
RG16Float
LLGL, 75
RG16SInt
LLGL, 75
RG16SNorm
LLGL, 75
RG16UInt
LLGL, 75
RG16UNorm
LLGL, 75
RG32Float
LLGL, 75
RG32SInt
LLGL, 75
RG32UInt
LLGL, 75
RG64Float
LLGL, 75
RG8SInt
LLGL, 75
RG8SNorm
LLGL, 75
RG8UInt
LLGL, 75
RG8UNorm
LLGL, 75
RGB10A2UInt
LLGL, 77
RGB10A2UNorm
LLGL, 77
RGB16Float
LLGL, 76
RGB16SInt
LLGL, 75
RGB16SNorm
LLGL, 75
RGB16UInt
LLGL, 75
RGB16UNorm

LLGL, 75
RGB32Float
LLGL, 76
RGB32SInt
LLGL, 76
RGB32UInt
LLGL, 76
RGB64Float
LLGL, 76
RGB8SInt
LLGL, 75
RGB8SNorm
LLGL, 75
RGB8UInt
LLGL, 75
RGB8UNorm
LLGL, 75
RGB8UNorm_sRGB
LLGL, 75
RGB9E5Float
LLGL, 77
RGBA16Float
LLGL, 76
RGBA16SInt
LLGL, 76
RGBA16SNorm
LLGL, 76
RGBA16UInt
LLGL, 76
RGBA16UNorm
LLGL, 76
RGBA32Float
LLGL, 76
RGBA32SInt
LLGL, 76
RGBA32UInt
LLGL, 76
RGBA64Float
LLGL, 76
RGBA8SInt
LLGL, 76
RGBA8SNorm
LLGL, 76
RGBA8UInt
LLGL, 76
RGBA8UNorm
LLGL, 76
RGBA8UNorm_sRGB
LLGL, 76
RGBA
LLGL, 77
RGB
LLGL, 77
RMenu
LLGL, 80
RShift
LLGL, 80
RWBuffer
LLGL, 91
RWByteAddressBuffer
LLGL, 91
RWStructuredBuffer
LLGL, 91
RWin
LLGL, 79
rasterizer
LLGL::GraphicsPipelineDescriptor, 217
Read
LLGL::CPUAccessFlags, 180
readMask
LLGL::StencilFaceDescriptor, 351
ReadOnly
LLGL, 71
ReadPixels
LLGL::Image, 225
ReadTexture
LLGL::RenderSystem, 301
ReadWrite
LLGL::CPUAccessFlags, 180
LLGL, 71
Red
LLGL, 93
reduceDeviceMemoryFragmentation
LLGL::RendererConfigurationVulkan, 267
reference
LLGL::StencilFaceDescriptor, 351
Reflect
LLGL::ShaderProgram, 338
refreshRate
LLGL::DisplayModeDescriptor, 188
LLGL::VsyncDescriptor, 391
Release
LLGL::Image, 225
LLGL::RenderSystem, 302–304
RemoveEventListener
LLGL::Canvas, 130
LLGL::Window, 397
renderCondition
LLGL::QueryHeapDescriptor, 251
RenderConditionMode
LLGL, 85
renderConditionSections
LLGL::FrameProfile, 212
RenderContext
LLGL::InterfaceID, 235
LLGL::RenderContext, 259
LLGL::RenderTarget, 314
RenderContext.h, 426
RenderContextFlags.h, 427
RenderPass
LLGL::InterfaceID, 235
renderPass
LLGL::GraphicsPipelineDescriptor, 217
LLGL::RenderTargetDescriptor, 316
RenderPass.h, 429
RenderPassDesc

Global utility functions, especially to fill descriptor structures., 47
RenderPassFlags.h, 429
renderPassSections
LLGL::FrameProfile, 212
RenderSystem
LLGL::InterfaceID, 235
LLGL::RenderSystem, 293
RenderSystem.h, 430
RenderSystemChild
LLGL::InterfaceID, 235
RenderSystemChild.h, 431
RenderSystemDescriptor
LLGL::RenderSystemDescriptor, 309
RenderSystemFlags.h, 431
RenderTarget
LLGL::InterfaceID, 235
RenderTarget.h, 433
RenderTargetFlags.h, 433
RenderTargetIndex
LLGL, 93
rendererConfig
LLGL::RenderSystemDescriptor, 310
rendererConfigSize
LLGL::RenderSystemDescriptor, 310
RendererConfiguration.h, 427
rendererName
LLGL::RendererInfo, 270
RenderingDebugger
LLGL::RenderingDebugger::Message, 238
RenderingDebugger.h, 428
RenderingProfiler.h, 428
Repeat
LLGL, 87
Replace
LLGL, 91
ReportCallback
LLGL::Log, 102
ReportType
LLGL::Log, 102
Reserved
LLGL::InterfaceID, 236
LLGL::RendererID, 269
Reset
LLGL::Image, 225, 226
ResetDisplayMode
LLGL::Display, 186
ResetPixelFormat
LLGL::Surface, 359
ResetResourceSlots
LLGL::CommandBuffer, 163
resizable
LLGL::WindowDescriptor, 401
Resize
LLGL::Image, 226, 227
resolution
LLGL::DisplayModeDescriptor, 188
LLGL::RenderTargetDescriptor, 316
LLGL::VideoModeDescriptor, 386
Resource
LLGL::InterfaceID, 235
resource
LLGL::ResourceViewDescriptor, 321
Resource.h, 433
ResourceFlags.h, 434
ResourceHeap
LLGL::InterfaceID, 235
ResourceHeap.h, 434
ResourceHeapFlags.h, 435
ResourceType
LLGL, 86
ResourceViewDescriptor
LLGL::ResourceViewDescriptor, 320
resourceViews
LLGL::ResourceHeapDescriptor, 319
resources
LLGL::ShaderReflection, 344
Return
LLGL, 78
RevSubtract
LLGL, 69
RG
LLGL, 77
Right
LLGL, 78

S
LLGL, 79
SPIRV_100
LLGL, 90
SPIRV
LLGL, 90
sRGB
LLGL::FormatDescriptor, 204
SampleCount
LLGL::MultiSamplingDescriptor, 241
SampleID
LLGL, 93
SampleMask
LLGL, 93
sampleMask
LLGL::MultiSamplingDescriptor, 241
Sampled
LLGL::BindFlags, 115
sampledBufferBindings
LLGL::FrameProfile, 212
sampledTextureBindings
LLGL::FrameProfile, 212
Sampler
LLGL::InterfaceID, 235
LLGL::Sampler, 322
LLGL, 86, 95
Sampler.h, 435
SamplerAddressMode
LLGL, 86
samplerBindings
LLGL::FrameProfile, 213

SamplerFilter
LLGL, 88
SamplerFlags.h, 435
samples
LLGL::MultiSamplingDescriptor, 241
LLGL::TextureDescriptor, 365
SamplesPassed
LLGL, 85
Scissor
LLGL::Scissor, 326
scissorTestEnabled
LLGL::RasterizerDescriptor, 257
scissors
LLGL::GraphicsPipelineDescriptor, 217
screen
LLGL::NativeContextHandle, 242
ScreenOrigin
LLGL, 88
screenOrigin
LLGL::RenderingCapabilities, 272
ScrollLock
LLGL, 80
Select
LLGL, 78
semanticIndex
LLGL::FragmentAttribute, 206
LLGL::StreamOutputAttribute, 355
LLGL::VertexAttribute, 380
Set
LLGL, 82
SetBehavior
LLGL::Window, 398
SetClearColor
LLGL::CommandBuffer, 163
SetClearDepth
LLGL::CommandBuffer, 164
SetClearStencil
LLGL::CommandBuffer, 164
SetComputePipeline
LLGL::CommandBuffer, 164
SetComputeResourceHeap
LLGL::CommandBuffer, 165
SetConfiguration
LLGL::RenderSystem, 304
SetDebugGroup
LLGL::RenderingDebugger, 275
SetDesc
LLGL::Window, 398
SetDisplayMode
LLGL::Display, 186
SetDisplayModeByVideoMode
LLGL::RenderContext, 262
SetGraphicsAPIDependentState
LLGL::CommandBuffer, 165
SetGraphicsPipeline
LLGL::CommandBuffer, 166
SetGraphicsResourceHeap
LLGL::CommandBuffer, 166
SetIndexBuffer
LLGL::CommandBuffer, 167
SetName
LLGL::RenderSystemChild, 307
SetOrCreateSurface
LLGL::RenderContext, 262
SetPosition
LLGL::Window, 398
SetRendererInfo
LLGL::RenderSystem, 304
SetRenderingCaps
LLGL::RenderSystem, 304
SetReportCallback
LLGL::Log, 103
SetReportCallbackStd
LLGL::Log, 103
SetReportLimit
LLGL::Log, 104
SetResource
LLGL::CommandBuffer, 168
SetScissor
LLGL::CommandBuffer, 168
SetScissors
LLGL::CommandBuffer, 168
SetSize
LLGL::Window, 398
SetSource
LLGL::RenderingDebugger, 275
SetStreamOutputBuffer
LLGL::CommandBuffer, 169
SetStreamOutputBufferArray
LLGL::CommandBuffer, 169
SetTitle
LLGL::Canvas, 131
LLGL::Window, 398
SetUniform
LLGL::CommandBuffer, 170
SetUniforms
LLGL::CommandBuffer, 170
SetVertexBuffer
LLGL::CommandBuffer, 171
SetVertexBufferArray
LLGL::CommandBuffer, 171
SetVideoMode
LLGL::RenderContext, 263
SetViewport
LLGL::CommandBuffer, 171
SetViewports
LLGL::CommandBuffer, 171
SetVsync
LLGL::RenderContext, 263
SetWorkGroupSize
LLGL::ShaderProgram, 339
Shader
LLGL::InterfaceID, 235
LLGL::Shader, 328
Shader.h, 436
ShaderDescFromFile

Global utility functions, especially to fill descriptor structures., 47
ShaderDescriptor
LLGL::ShaderDescriptor, 331
ShaderFlags.h, 436
ShaderMacro
LLGL::ShaderMacro, 334
ShaderProgram
LLGL::InterfaceID, 235
shaderProgram
LLGL::ComputePipelineDescriptor, 179
LLGL::GraphicsPipelineDescriptor, 217
ShaderProgram.h, 437
ShaderProgramDesc
Global utility functions, especially to fill descriptor structures., 47, 48
ShaderProgramFlags.h, 438
ShaderSourceType
LLGL, 89
ShaderType
LLGL, 89
ShadingLanguage
LLGL, 89
shadingLanguageName
LLGL::RendererInfo, 270
shadingLanguages
LLGL::RenderingCapabilities, 272
ShareSurfaceAndConfig
LLGL::RenderContext, 263
Shift
LLGL, 78
Show
LLGL::Window, 398
ShowCursor
LLGL::Display, 187
size
LLGL::BufferDescriptor, 127
LLGL::ShaderUniform, 346
LLGL::WindowDescriptor, 401
Sleep
LLGL, 79
slopeFactor
LLGL::DepthBiasDescriptor, 181
slot
LLGL::BindingDescriptor, 118
Snapshot
LLGL, 78
source
LLGL::ShaderDescriptor, 332
sourceSize
LLGL::ShaderDescriptor, 333
sourceType
LLGL::ShaderDescriptor, 333
Space
LLGL, 78
Src1Alpha
LLGL, 69
Src1Color
LLGL, 69
SrcAlpha
LLGL, 69
srcAlpha
LLGL::BlendTargetDescriptor, 122
SrcAlphaSaturate
LLGL, 69
SrcColor
LLGL, 69
srcColor
LLGL::BlendTargetDescriptor, 122
SrcImageDescriptor
LLGL::SrcImageDescriptor, 347
stageFlags
LLGL::BindingDescriptor, 118
Start
LLGL::Timer, 377
startComponent
LLGL::StreamOutputAttribute, 356
Stencil
LLGL::ClearFlags, 132
LLGL, 68, 93
stencil
LLGL::ClearValue, 133
LLGL::FormatDescriptor, 205
LLGL::GraphicsPipelineDescriptor, 218
stencilAttachment
LLGL::RenderPassDescriptor, 289
stencilBits
LLGL::VideoModeDescriptor, 386
stencilFailOp
LLGL::StencilFaceDescriptor, 351
StencilOp
LLGL, 90
Stop
LLGL::Timer, 377
Storage
LLGL::BindFlags, 115
storageBuffer
LLGL::BufferDescriptor, 127
storageBufferBindings
LLGL::FrameProfile, 213
StorageBufferDesc
Global utility functions, especially to fill descriptor structures., 48
StorageBufferType
LLGL, 91
storageBufferType
LLGL::ShaderResource, 345
storageTextureBindings
LLGL::FrameProfile, 213
storageType
LLGL::BufferDescriptor::StorageBuffer, 353
Store
LLGL, 68
storeOp
LLGL::AttachmentFormatDescriptor, 114
stream

LLGL::StreamOutputAttribute, 356
StreamOutOverflow
LLGL, 85
StreamOutPrimitivesWritten
LLGL, 85
streamOutput
LLGL::ShaderDescriptor, 333
StreamOutputAttribute
LLGL::StreamOutputAttribute, 355
StreamOutputAttribute.h, 439
streamOutputAttributes
LLGL::ShaderReflection, 344
StreamOutputBuffer
LLGL::BindFlags, 115
streamOutputBufferBindings
LLGL::FrameProfile, 213
StreamOutputFormat.h, 440
streamOutputSections
LLGL::FrameProfile, 213
stride
LLGL::BufferDescriptor::StorageBuffer, 353
LLGL::VertexFormat, 383
Strings.h, 440
StructuredBuffer
LLGL, 91
Submit
LLGL::CommandQueue, 176, 177
subresource
LLGL::TextureRegion, 369
LLGL::TextureViewDescriptor, 374
Subtract
LLGL, 69
Surface
LLGL::InterfaceID, 236
Surface.h, 441
Swap
LLGL::Image, 227
swapChainSize
LLGL::VideoModeDescriptor, 387
SwitchFullscreenMode
LLGL::RenderContext, 264
swizzle
LLGL::TextureViewDescriptor, 374
SystemValue
LLGL, 91
systemValue
LLGL::FragmentAttribute, 206
LLGL::VertexAttribute, 380
SystemValue.h, 441

T
LLGL, 79
Tab
LLGL, 78
Tags.h, 441
targets
LLGL::BlendDescriptor, 120
TessControl
LLGL, 89
tessControlShader
LLGL::ShaderProgramDescriptor, 342
tessControlShaderInvocations
LLGL::QueryPipelineStatistics, 254
TessControlStage
LLGL::StageFlags, 349
TessEvaluation
LLGL, 89
tessEvaluationShader
LLGL::ShaderProgramDescriptor, 342
tessEvaluationShaderInvocations
LLGL::QueryPipelineStatistics, 254
TessEvaluationStage
LLGL::StageFlags, 349
testEnabled
LLGL::DepthDescriptor, 182
LLGL::StencilDescriptor, 350
Texture
LLGL::InterfaceID, 235
LLGL::Texture, 361
LLGL, 86
texture
LLGL::AttachmentDescriptor, 112
Texture utility functions to determine texture dimension and buffer sizes., 40
IsArrayTexture, 40
IsCubeTexture, 40
IsMipMappedTexture, 40
IsMultiSampleTexture, 41
IsTextureSwizzleIdentity, 41
NumMipLevels, 41, 42
TextureBufferSize, 42
TextureSize, 42
Texture.h, 442
Texture1DArray
LLGL, 94
Texture1DArrayDesc
Global utility functions, especially to fill descriptor structures., 48
Texture1DDesc
Global utility functions, especially to fill descriptor structures., 48
Texture1D
LLGL, 94
Texture2DArray
LLGL, 94
Texture2DArrayDesc
Global utility functions, especially to fill descriptor structures., 49
Texture2DDesc
Global utility functions, especially to fill descriptor structures., 49
Texture2DMSArray
LLGL, 94
Texture2DMSArrayDesc
Global utility functions, especially to fill descriptor structures., 49
Texture2DMSDesc

Global utility functions, especially to fill descriptor structures., 49
Texture2DMS
LLGL, 94
Texture2D
LLGL, 94
Texture3DDesc
Global utility functions, especially to fill descriptor structures., 49
Texture3D
LLGL, 94
TextureBufferSize
Texture utility functions to determine texture dimension and buffer sizes., 42
textureCopies
LLGL::FrameProfile, 214
TextureCube
LLGL, 94
TextureCubeArray
LLGL, 94
TextureCubeArrayDesc
Global utility functions, especially to fill descriptor structures., 50
TextureCubeDesc
Global utility functions, especially to fill descriptor structures., 50
TextureFlags.h, 442
textureFormats
LLGL::RenderingCapabilities, 272
TextureLocation
LLGL::TextureLocation, 367
textureMappings
LLGL::FrameProfile, 214
textureReads
LLGL::FrameProfile, 214
TextureRegion
LLGL::TextureRegion, 368
TextureSize
Texture utility functions to determine texture dimension and buffer sizes., 42
TextureSubresource
LLGL::TextureSubresource, 370
TextureSwizzle
LLGL, 93
TextureType
LLGL, 93
textureView
LLGL::ResourceViewDescriptor, 321
textureWrites
LLGL::FrameProfile, 214
threadCount
LLGL::RenderSystemConfiguration, 308
TimeElapsed
LLGL, 85
Timer
LLGL::InterfaceID, 236
Timer.h, 443
title
LLGL::CanvasDescriptor, 131
LLGL::WindowDescriptor, 402
ToRGBA
LLGL::Color< T, 3u >, 141
ToRGB
LLGL::Color< T, 4u >, 146
ToReportString
LLGL::RenderingDebugger::Message, 238
ToString
Global type-to-string conversion functions., 38, 39
TooManyAttachments
LLGL::ShaderProgram, 336
TriangleFan
LLGL, 83
TriangleList
LLGL, 83
TriangleListAdjacency
LLGL, 83
TriangleStrip
LLGL, 83
TriangleStripAdjacency
LLGL, 83
Triangles
LLGL, 85
type
LLGL::AttachmentDescriptor, 112
LLGL::BindingDescriptor, 118
LLGL::QueryHeapDescriptor, 251
LLGL::ShaderDescriptor, 333
LLGL::ShaderUniform, 346
LLGL::TextureDescriptor, 365
LLGL::TextureViewDescriptor, 374
TypeInfo.h, 444
Types.h, 444

U
LLGL, 79
UInt1
LLGL, 94
UInt16
LLGL, 72
UInt2
LLGL, 94
UInt3
LLGL, 94
UInt32
LLGL, 72
UInt4
LLGL, 94
UInt8
LLGL, 72
Undefined
LLGL::RendererID, 269
LLGL, 67, 68, 72, 74, 86, 89, 91, 92, 94
UndefinedBehavior
LLGL, 72
UniformLocation
LLGL, 67
UniformType

LLGL, 94
uniforms
LLGL::ShaderReflection, 344
Unload
LLGL::RenderSystem, 305
UnmapBuffer
LLGL::RenderSystem, 305
UnsupportedFeature
LLGL, 72
Up
LLGL, 78
UpdateBuffer
LLGL::CommandBuffer, 172
UpperLeft
LLGL, 89
Utility.h, 445

V
LLGL, 79
ValidateMipResolution
LLGL::RenderTarget, 314
ValidateRenderingCaps
LLGL, 100
ValidateRenderingCapsFunc
Global type aliases to callback interfaces., 25
ValidateResolution
LLGL::RenderTarget, 314
ValidateShaderComposition
LLGL::ShaderProgram, 340
values
LLGL::FrameProfile, 214
VaryingBehavior
LLGL, 95
vendor
LLGL::VideoAdapterDescriptor, 384
vendorName
LLGL::RendererInfo, 271
Version.h, 447
VersionBitmask
LLGL, 90
Vertex
LLGL, 89
VertexAttribute
LLGL::VertexAttribute, 378, 379
VertexAttribute.h, 447
vertexAttributes
LLGL::ShaderReflection, 344
VertexBuffer
LLGL::BindFlags, 115
vertexBuffer
LLGL::BufferDescriptor, 127
vertexBufferBindings
LLGL::FrameProfile, 214
VertexBufferDesc
Global utility functions, especially to fill descriptor structures., 50
VertexFormat.h, 448
vertexFormats
LLGL::ShaderProgramDescriptor, 342
VertexID
LLGL, 93
vertexOffset
LLGL::DrawIndexedIndirectArguments, 190
vertexShader
LLGL::ShaderProgramDescriptor, 342
vertexShaderInvocations
LLGL::QueryPipelineStatistics, 254
VertexStage
LLGL::StageFlags, 349
VideoAdapter.h, 448
videoMemory
LLGL::VideoAdapterDescriptor, 384
videoMode
LLGL::RenderContextDescriptor, 264
view
LLGL::NativeHandle, 243
Viewport
LLGL::Viewport, 389
ViewportIndex
LLGL, 93
viewports
LLGL::GraphicsPipelineDescriptor, 218
visible
LLGL::WindowDescriptor, 402
visual
LLGL::NativeContextHandle, 242
LLGL::NativeHandle, 243
VolumeDown
LLGL, 81
VolumeMute
LLGL, 80
VolumeUp
LLGL, 81
vsync
LLGL::RenderContextDescriptor, 265
Vulkan
LLGL::RendererID, 270

W
LLGL, 79
Wait
LLGL, 86
WaitFence
LLGL::CommandQueue, 177
WaitIdle
LLGL::CommandQueue, 177
WaitInverted
LLGL, 86
WarnError
LLGL::ShaderCompileFlags, 329
Warning
LLGL::Log, 102
WarningType
LLGL, 95
width
LLGL::Extent2D, 200
LLGL::Extent3D, 201
LLGL::Scissor, 326

LLGL::Viewport, 390
Win32NativeHandle.h, 449
Window
LLGL::InterfaceID, 236
LLGL::Window::EventListener, 198
window
LLGL::NativeHandle, 243
Window.h, 449
Window_EventListener
LLGL::InterfaceID, 236
windowContext
LLGL::WindowDescriptor, 402
WindowFlags.h, 450
Wireframe
LLGL, 82
Write
LLGL::CPUAccessFlags, 180
WriteBuffer
LLGL::RenderSystem, 305
WriteDiscard
LLGL, 71
writeEnabled
LLGL::DepthDescriptor, 182
writeMask
LLGL::StencilFaceDescriptor, 352
WriteOnly
LLGL, 71
WritePixels
LLGL::Image, 227
WriteTexture
LLGL::RenderSystem, 305

X
LLGL, 79
x
LLGL::Offset2D, 246
LLGL::Offset3D, 247
LLGL::Scissor, 327
LLGL::Viewport, 390
XButton1
LLGL, 78
XButton2
LLGL, 78
XOR
LLGL, 82

Y
LLGL, 79
y
LLGL::Offset2D, 246
LLGL::Offset3D, 247
LLGL::Scissor, 327
LLGL::Viewport, 390

Z
LLGL, 79
z
LLGL::Offset3D, 247
Zero
LLGL, 69, 91, 93
ZeroToOne
LLGL, 70
Zoom
LLGL, 81