LLGL

0.02 Beta

Generated by Doxygen 1.8.11

Contents

1 LLGL 0.02 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. 26
8.4.1 Detailed Description 26
8.4.2 Function Documentation 26
8.4.2.1 IsByteAddressBuffer(const StorageBufferType type) 26
8.4.2.2 IsRWBuffer(const StorageBufferType type) 26
8.4.2.3 IsStructuredBuffer(const StorageBufferType type) 26
8.4.2.4 IsTypedBuffer(const StorageBufferType type) 26
8.5 Hardware format utility functions. 27
8.5.1 Detailed Description 27

8.5.2 Function Documentation 27
8.5.2.1 FormatBitSize(const Format format) 27
8.5.2.2 IsCompressedFormat(const Format format) 28
8.5.2.3 IsDepthFormat(const Format format) 28
8.5.2.4 IsDepthStencilFormat(const Format format) 28
8.5.2.5 IsFloatFormat(const Format format) 28
8.5.2.6 IsIntegralFormat(const Format format) 29
8.5.2.7 IsNormalizedFormat(const Format format) 29
8.5.2.8 IsStencilFormat(const Format format) 29
8.5.2.9 SplitFormat(const Format format, DataType &dataType, std::uint32_t &components) 29
8.6 Data type utility functions. 30
8.6.1 Detailed Description 30
8.6.2 Function Documentation 30
8.6.2.1 DataTypeSize(const DataType dataType) 30
8.6.2.2 IsFloatDataType(const DataType dataType) 30
8.6.2.3 IsIntDataType(const DataType dataType) 30
8.6.2.4 IsUIntDataType(const DataType dataType) 30
8.7 Image utility functions to classify and convert image data. 31
8.7.1 Detailed Description 31
8.7.2 Function Documentation 31
8.7.2.1 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, const Dst↩ImageDescriptor &dstImageDesc, std::size_t threadCount=0) 31
8.7.2.2 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, ImageFormat dstFormat, DataType dstDataType, std::size_t threadCount=0) 32
8.7.2.3 FindSuitableImageFormat(const Format format, ImageFormat &imageFormat, DataType &dataType) 33
8.7.2.4 GenerateEmptyByteBuffer(std::size_t bufferSize, bool initialize=true) 33
8.7.2.5 GenerateImageBuffer(ImageFormat format, DataType dataType, std::size_↩t imageSize, const ColorRGBAd &fillColor) 34
8.7.2.6 ImageDataSize(const ImageFormat imageFormat, const DataType dataType, std::uint32_t numPixels) 34
8.7.2.7 ImageFormatSize(const ImageFormat imageFormat) 35
8.7.2.8 IsCompressedFormat(const ImageFormat imageFormat) 35

8.7.2.9 IsDepthStencilFormat(const ImageFormat imageFormat) 35
8.8 Global type-to-string conversion functions. 36
8.8.1 Detailed Description 36
8.8.2 Function Documentation 36
8.8.2.1 ToString(const ShaderType t) 36
8.8.2.2 ToString(const ErrorType t) 36
8.8.2.3 ToString(const WarningType t) 36
8.8.2.4 ToString(const ShadingLanguage t) 37
8.8.2.5 ToString(const Format t) 37
8.9 Texture utility functions to determine texture dimension and buffer sizes. 38
8.9.1 Detailed Description 38
8.9.2 Function Documentation 38
8.9.2.1 IsArrayTexture(const TextureType type) 38
8.9.2.2 IsCubeTexture(const TextureType type) 38
8.9.2.3 IsMipMappedTexture(const TextureDescriptor &textureDesc) 39
8.9.2.4 IsMultiSampleTexture(const TextureType type) 39
8.9.2.5 NumMipLevels(std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1) 39
8.9.2.6 NumMipLevels(const TextureDescriptor &textureDesc) 39
8.9.2.7 TextureBufferSize(const Format format, std::uint32_t numTexels) 40
8.9.2.8 TextureSize(const TextureDescriptor &textureDesc) 40
8.10 Global utility functions, especially to fill descriptor structures. 41
8.10.1 Detailed Description 42
8.10.2 Function Documentation 42
8.10.2.1 ConstantBufferDesc(uint64_t size, long flags=BufferFlags::DynamicUsage) . . . 42
8.10.2.2 IndexBufferDesc(uint64_t size, const IndexFormat &indexFormat, long flags=0) . 42
8.10.2.3 PipelineLayoutDesc(const ShaderReflectionDescriptor &reflectionDesc) 42
8.10.2.4 PipelineLayoutDesc(const char ∗layoutSignature) 42
8.10.2.5 RenderPassDesc(const RenderTargetDescriptor &renderTargetDesc) 44
8.10.2.6 ShaderDescFromFile(const ShaderType type, const char ∗filename, const char ∗entryPoint=nullptr, const char ∗profile=nullptr, long flags=0) 44

8.10.2.7 ShaderProgramDesc(const std::initializer_list< Shader ∗ > &shaders, const std::initializer_list< VertexFormat > &vertexFormats={}) 44

8.10.2.8 ShaderProgramDesc(const std::vector< Shader ∗ > &shaders, const std↩ ::vector< VertexFormat > &vertexFormats={}) 45

8.10.2.9 StorageBufferDesc(uint64_t size, const StorageBufferType storageType, std↩ ::uint32_t stride, long flags=BufferFlags::MapReadAccess"|BufferFlags::Map↩ WriteAccess) 45

8.10.2.10 Texture1DArrayDesc(Format format, std::uint32_t width, std::uint32_t array↩ Layers, long flags=TextureFlags::Default) 45

8.10.2.11 Texture1DDesc(Format format, std::uint32_t width, long flags=TextureFlags::↩ Default) 46

8.10.2.12 Texture2DArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long flags=TextureFlags::Default) 46

8.10.2.13 Texture2DDesc(Format format, std::uint32_t width, std::uint32_t height, long flags=TextureFlags::Default) 46

8.10.2.14 Texture2DMSArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long flags=TextureFlags::Default) 46

8.10.2.15 Texture2DMSDesc(Format format, std::uint32_t width, std::uint32_t height, std↩ ::uint32_t samples, long flags=TextureFlags::Default) 46

8.10.2.16 Texture3DDesc(Format format, std::uint32_t width, std::uint32_t height, std↩ ::uint32_t depth, long flags=TextureFlags::Default) 47

8.10.2.17 TextureCubeArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long flags=TextureFlags::Default) 47

8.10.2.18 TextureCubeDesc(Format format, std::uint32_t width, std::uint32_t height, long flags=TextureFlags::Default) 47

8.10.2.19 VertexBufferDesc(uint64_t size, const VertexFormat &vertexFormat, long flags=0) 47

9 Namespace Documentation 49

9.1 LLGL Namespace Reference 49

9.1.1 Typedef Documentation 64

9.1.1.1 ByteBuffer 64

9.1.1.2 ColorRGBAb 65

9.1.1.3 ColorRGBAd 65

9.1.1.4 ColorRGBAf 65

9.1.1.5 ColorRGBAT 65

9.1.1.6 ColorRGBAub 65

9.1.1.7 ColorRGBb 65

9.1.1.8 ColorRGBd 65

9.1.1.9 ColorRGBf 65

9.1.1.10 ColorRGBT 65

9.1.1.11 ColorRGBub 65

9.1.1.12 UniformLocation 65

9.1.2 Enumeration Type Documentation 65

9.1.2.1 AttachmentLoadOp 65

9.1.2.2 AttachmentStoreOp 66

9.1.2.3 AttachmentType 66

9.1.2.4 BlendArithmetic 67

9.1.2.5 BlendOp 67

9.1.2.6 BufferType 68

9.1.2.7 ClippingRange 68

9.1.2.8 CompareOp 68

9.1.2.9 CPUAccess 69

9.1.2.10 CullMode 69

9.1.2.11 DataType 70

9.1.2.12 ErrorType 70

9.1.2.13 Format 70

9.1.2.14 ImageFormat 73

9.1.2.15 Key 74

9.1.2.16 LogicOp 78

9.1.2.17 OpenGLContextProfile 78

9.1.2.18 PolygonMode 79

9.1.2.19 PrimitiveTopology 79

9.1.2.20 PrimitiveType 81

9.1.2.21 QueryType 81

9.1.2.22 RenderConditionMode 82

9.1.2.23 ResourceType 82

9.1.2.24 SamplerAddressMode 83

9.1.2.25 SamplerFilter 84
9.1.2.26 ScreenOrigin 85
9.1.2.27 ShaderSourceType 85
9.1.2.28 ShaderType 86
9.1.2.29 ShadingLanguage 86
9.1.2.30 StencilOp 87
9.1.2.31 StorageBufferType 88
9.1.2.32 TextureType 88
9.1.2.33 UniformType 88
9.1.2.34 WarningType 90
9.1.3 Function Documentation 90
9.1.3.1 CastColorValue(const Src &value) 90
9.1.3.2 CastColorValue< bool, bool >(const bool &value) 90
9.1.3.3 CastColorValue< double, double >(const double &value) 90
9.1.3.4 CastColorValue< float, float >(const float &value) 90
9.1.3.5 CastColorValue< std::uint8_t, std::uint8_t >(const std::uint8_t &value) 90
9.1.3.6 GetExtentRatio(const Extent2D &extent) 90
9.1.3.7 GetPrimitiveTopologyPatchSize(const PrimitiveTopology primitiveTopology) . . . 91
9.1.3.8 IsPrimitiveTopologyPatches(const PrimitiveTopology primitiveTopology) 91
9.1.3.9 IsShaderSourceBinary(const ShaderSourceType type) 91
9.1.3.10 IsShaderSourceCode(const ShaderSourceType type) 91
9.1.3.11 MaxColorValue() 91
9.1.3.12 MaxColorValue< bool >() 91
9.1.3.13 MaxColorValue< std::uint8_t >() 91
9.1.3.14 operator"!=(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs) 91
9.1.3.15 operator"!=(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 92
9.1.3.16 operator"!=(const VertexAttribute &lhs, const VertexAttribute &rhs) 92
9.1.3.17 operator"!=(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) 92
9.1.3.18 operator"!=(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 92
9.1.3.19 operator"!=(const Color< T, N > &lhs, const Color< T, N > &rhs) 92

9.1.3.20 operator∗(const Color< T, N > &lhs, const Color< T, N > &rhs) 92
9.1.3.21 operator∗(const Color< T, N > &lhs, const T &rhs) 92
9.1.3.22 operator∗(const T &lhs, const Color< T, N > &rhs) 92
9.1.3.23 operator+(const Color< T, N > &lhs, const Color< T, N > &rhs) 92
9.1.3.24 operator-(const Color< T, N > &lhs, const Color< T, N > &rhs) 92
9.1.3.25 operator/(const Color< T, N > &lhs, const Color< T, N > &rhs) 92
9.1.3.26 operator/(const Color< T, N > &lhs, const T &rhs) 92
9.1.3.27 operator/(const T &lhs, const Color< T, N > &rhs) 92
9.1.3.28 operator==(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs) 92
9.1.3.29 operator==(const StreamOutputAttribute &lhs, const StreamOutputAttribute &rhs) 93
9.1.3.30 operator==(const VertexAttribute &lhs, const VertexAttribute &rhs) 93
9.1.3.31 operator==(const VsyncDescriptor &lhs, const VsyncDescriptor &rhs) 93
9.1.3.32 operator==(const VideoModeDescriptor &lhs, const VideoModeDescriptor &rhs) 93
9.1.3.33 operator==(const Color< T, N > &lhs, const Color< T, N > &rhs) 93
9.1.3.34 ValidateRenderingCaps(const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={}) 93
9.2 LLGL::Constants Namespace Reference 94
9.2.1 Detailed Description 94
9.3 LLGL::Log Namespace Reference 94
9.3.1 Typedef Documentation 95
9.3.1.1 ReportCallback 95
9.3.2 Enumeration Type Documentation 95
9.3.2.1 ReportType 95
9.3.3 Function Documentation 96
9.3.3.1 PostReport(ReportType type, const std::string &message, const std::string &contextInfo="""") 96
9.3.3.2 SetReportCallback(const ReportCallback &callback, void ∗userData=nullptr) . . 96
9.3.3.3 SetReportCallbackStd(std::ostream &stream=std::cerr) 96
9.4 LLGL::Version Namespace Reference 97
9.4.1 Detailed Description 97
9.4.2 Function Documentation 97
9.4.2.1 GetID() 97
9.4.2.2 GetMajor() 97
9.4.2.3 GetMinor() 97
9.4.2.4 GetRevision() 98
9.4.2.5 GetStatus() 98
9.4.2.6 GetString() 98

10 Class Documentation **99**

10.1 LLGL::ApplicationDescriptor Struct Reference 99

10.1.1 Detailed Description 99

10.1.2 Member Data Documentation 99

10.1.2.1 applicationName 99

10.1.2.2 applicationVersion 100

10.1.2.3 engineName 100

10.1.2.4 engineVersion 100

10.2 LLGL::AttachmentClear Struct Reference 100

10.2.1 Detailed Description 101

10.2.2 Constructor & Destructor Documentation 101

10.2.2.1 AttachmentClear()=default 101

10.2.2.2 AttachmentClear(const AttachmentClear &)=default 101

10.2.2.3 AttachmentClear(const ColorRGBAf &color, std::uint32_t colorAttachment) . . . 101

10.2.2.4 AttachmentClear(float depth) 101

10.2.2.5 AttachmentClear(std::uint32_t stencil) 101

10.2.2.6 AttachmentClear(float depth, std::uint32_t stencil) 101

10.2.3 Member Function Documentation 101

10.2.3.1 operator=(const AttachmentClear &)=default 101

10.2.4 Member Data Documentation 101

10.2.4.1 clearValue 101

10.2.4.2 colorAttachment 102

10.2.4.3 flags 102

10.3 LLGL::AttachmentDescriptor Struct Reference 102

10.3.1 Detailed Description 103

10.3.2 Constructor & Destructor Documentation 103

10.3.2.1 AttachmentDescriptor()=default 103

10.3.2.2 AttachmentDescriptor(const AttachmentDescriptor &)=default 103

10.3.2.3 AttachmentDescriptor(AttachmentType type) 103

10.3.2.4 AttachmentDescriptor(AttachmentType type, Texture ∗texture, std::uint32_t mip←
Level=0, std::uint32_t arrayLayer=0) 103

10.3.3 Member Data Documentation 103
10.3.3.1 arrayLayer 103
10.3.3.2 mipLevel 104
10.3.3.3 texture 104
10.3.3.4 type 104
10.4 LLGL::AttachmentFormatDescriptor Struct Reference 104
10.4.1 Detailed Description 105
10.4.2 Constructor & Destructor Documentation 105
10.4.2.1 AttachmentFormatDescriptor()=default 105
10.4.2.2 AttachmentFormatDescriptor(const AttachmentFormatDescriptor &)=default . . . 105
10.4.2.3 AttachmentFormatDescriptor(const Format format, const AttachmentLoad↩ Op loadOp=AttachmentLoadOp::Load, const AttachmentStoreOp storeOp=↩ AttachmentStoreOp::Store) 105
10.4.3 Member Data Documentation 105
10.4.3.1 format 105
10.4.3.2 loadOp 106
10.4.3.3 storeOp 106
10.5 LLGL::BindingDescriptor Struct Reference 106
10.5.1 Detailed Description 107
10.5.2 Constructor & Destructor Documentation 107
10.5.2.1 BindingDescriptor()=default 107
10.5.2.2 BindingDescriptor(const BindingDescriptor &)=default 107
10.5.2.3 BindingDescriptor(ResourceType type, long stageFlags, std::uint32_t slot, std↩ ::uint32_t arraySize=1) 107
10.5.3 Member Data Documentation 107
10.5.3.1 arraySize 107
10.5.3.2 slot 107
10.5.3.3 stageFlags 107
10.5.3.4 type 108
10.6 LLGL::BlendDescriptor Struct Reference 108
10.6.1 Detailed Description 108
10.6.2 Member Data Documentation 108

10.6.2.1 alphaToCoverageEnabled 108
10.6.2.2 blendFactor 109
10.6.2.3 independentBlendEnabled 109
10.6.2.4 logicOp 109
10.6.2.5 targets 110
10.7 LLGL::BlendTargetDescriptor Struct Reference 110
10.7.1 Detailed Description 110
10.7.2 Member Data Documentation 111
10.7.2.1 alphaArithmetic 111
10.7.2.2 blendEnabled 111
10.7.2.3 colorArithmetic 111
10.7.2.4 colorMask 111
10.7.2.5 dstAlpha 111
10.7.2.6 dstColor 111
10.7.2.7 srcAlpha 111
10.7.2.8 srcColor 111
10.8 LLGL::Buffer Class Reference 112
10.8.1 Detailed Description 112
10.8.2 Constructor & Destructor Documentation 112
10.8.2.1 Buffer(const BufferType type) 112
10.8.3 Member Function Documentation 112
10.8.3.1 GetType() const 112
10.8.3.2 QueryResourceType() const override 113
10.9 LLGL::BufferArray Class Reference 113
10.9.1 Detailed Description 113
10.9.2 Constructor & Destructor Documentation 114
10.9.2.1 BufferArray(const BufferType type) 114
10.9.3 Member Function Documentation 114
10.9.3.1 GetType() const 114
10.10LLGL::BufferDescriptor Struct Reference 114

10.10.1 Detailed Description 114
10.10.2 Member Data Documentation 115
10.10.2.1 flags 115
10.10.2.2 indexBuffer 115
10.10.2.3 size 115
10.10.2.4 storageBuffer 115
10.10.2.5 type 115
10.10.2.6 vertexBuffer 115
10.11 LLGL::BufferFlags Struct Reference 116
10.11.1 Detailed Description 116
10.11.2 Member Enumeration Documentation 116
10.11.2.1 anonymous enum 116
10.12 LLGL::Canvas Class Reference 117
10.12.1 Detailed Description 118
10.12.2 Member Function Documentation 118
10.12.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc) override 118
10.12.2.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 118
10.12.2.3 Create(const CanvasDescriptor &desc) 118
10.12.2.4 GetTitle() const =0 118
10.12.2.5 OnProcessEvents()=0 119
10.12.2.6 ProcessEvents() 119
10.12.2.7 RemoveEventListener(const EventListener ∗eventListener) 119
10.12.2.8 SetTitle(const std::wstring &title)=0 119
10.13 LLGL::CanvasDescriptor Struct Reference 119
10.13.1 Detailed Description 119
10.13.2 Member Data Documentation 120
10.13.2.1 borderless 120
10.13.2.2 title 120
10.14 LLGL::ClearFlags Struct Reference 120
10.14.1 Detailed Description 120

10.14.2 Member Enumeration Documentation 120
10.14.2.1 anonymous enum 120
10.15LLGL::ClearValue Struct Reference 121
10.15.1 Detailed Description 121
10.15.2 Member Data Documentation 121
10.15.2.1 color 121
10.15.2.2 depth 121
10.15.2.3 stencil 121
10.16LLGL::Color< T, N > Class Template Reference 122
10.16.1 Detailed Description 122
10.16.2 Constructor & Destructor Documentation 123
10.16.2.1 Color() 123
10.16.2.2 Color(const Color< T, N > &rhs) 123
10.16.2.3 Color(const T &scalar) 123
10.16.2.4 Color(UninitializeTag) 123
10.16.3 Member Function Documentation 123
10.16.3.1 Cast() const 123
10.16.3.2 operator*=(const Color< T, N > &rhs) 124
10.16.3.3 operator*=(const T rhs) 124
10.16.3.4 operator+=(const Color< T, N > &rhs) 124
10.16.3.5 operator-() const 124
10.16.3.6 operator-=(const Color< T, N > &rhs) 124
10.16.3.7 operator/=(const Color< T, N > &rhs) 124
10.16.3.8 operator/=(const T rhs) 124
10.16.3.9 operator[](std::size_t component) 124
10.16.3.10operator[](std::size_t component) const 125
10.16.3.11Ptr() 125
10.16.3.12Ptr() const 125
10.16.4 Member Data Documentation 125
10.16.4.1 components 125

10.17 LLGL::Color< T, 3u > Class Template Reference . . . 125
10.17.1 Detailed Description . . . 127
10.17.2 Constructor & Destructor Documentation . . . 127
10.17.2.1 Color() . . . 127
10.17.2.2 Color(const Color< T, 3 > &rhs) . . . 127
10.17.2.3 Color(const T &scalar) . . . 127
10.17.2.4 Color(const T &r, const T &g, const T &b) . . . 127
10.17.2.5 Color(UninitializeTag) . . . 127
10.17.3 Member Function Documentation . . . 127
10.17.3.1 Cast() const . . . 127
10.17.3.2 operator∗=(const Color< T, 3 > &rhs) . . . 128
10.17.3.3 operator∗=(const T rhs) . . . 128
10.17.3.4 operator+=(const Color< T, 3 > &rhs) . . . 128
10.17.3.5 operator-() const . . . 128
10.17.3.6 operator-=(const Color< T, 3 > &rhs) . . . 128
10.17.3.7 operator/=(const Color< T, 3 > &rhs) . . . 128
10.17.3.8 operator/=(const T rhs) . . . 128
10.17.3.9 operator[](std::size_t component) . . . 128
10.17.3.10 operator[](std::size_t component) const . . . 129
10.17.3.11 Ptr() . . . 129
10.17.3.12 Ptr() const . . . 129
10.17.3.13 ToRGBA() const . . . 129
10.17.4 Member Data Documentation . . . 129
10.17.4.1 b . . . 129
10.17.4.2 components . . . 129
10.17.4.3 g . . . 130
10.17.4.4 r . . . 130
10.18 LLGL::Color< T, 4u > Class Template Reference . . . 130
10.18.1 Detailed Description . . . 131
10.18.2 Constructor & Destructor Documentation . . . 131

10.18.2.1 Color() 131
10.18.2.2 Color(const Color< T, 4 > &rhs) 132
10.18.2.3 Color(const T &scalar) 132
10.18.2.4 Color(const Color< T, 3u > &rhs) 132
10.18.2.5 Color(const T &r, const T &g, const T &b) 132
10.18.2.6 Color(const T &r, const T &g, const T &b, const T &a) 132
10.18.2.7 Color(UninitializeTag) 132
10.18.3 Member Function Documentation 132
10.18.3.1 Cast() const 132
10.18.3.2 operator∗=(const Color< T, 4 > &rhs) 133
10.18.3.3 operator∗=(const T rhs) 133
10.18.3.4 operator+=(const Color< T, 4 > &rhs) 133
10.18.3.5 operator-() const 133
10.18.3.6 operator-=(const Color< T, 4 > &rhs) 133
10.18.3.7 operator/=(const Color< T, 4 > &rhs) 133
10.18.3.8 operator/=(const T rhs) 133
10.18.3.9 operator[](std::size_t component) 133
10.18.3.10operator[](std::size_t component) const 134
10.18.3.11Ptr() 134
10.18.3.12Ptr() const 134
10.18.3.13ToRGB() const 134
10.18.4 Member Data Documentation 134
10.18.4.1 a 134
10.18.4.2 b 134
10.18.4.3 components 134
10.18.4.4 g 135
10.18.4.5 r 135
10.19LLGL::CommandBuffer Class Reference 135
10.19.1 Detailed Description 137
10.19.2 Constructor & Destructor Documentation 137

10.19.2.1 CommandBuffer()=default 137

10.19.3 Member Function Documentation 137

10.19.3.1 Begin()=0 137

10.19.3.2 BeginQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 138

10.19.3.3 BeginRenderCondition(QueryHeap &queryHeap, std::uint32_t query=0, const RenderConditionMode mode=RenderConditionMode::Wait)=0 138

10.19.3.4 BeginRenderPass(RenderTarget &renderTarget, const RenderPass ∗render↩ Pass=nullptr, std::uint32_t numClearValues=0, const ClearValue ∗clear↩ Values=nullptr)=0 139

10.19.3.5 BeginStreamOutput(const PrimitiveType primitiveType)=0 140

10.19.3.6 Clear(long flags)=0 140

10.19.3.7 ClearAttachments(std::uint32_t numAttachments, const AttachmentClear ∗attachments)=0 140

10.19.3.8 CopyBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, Buffer &srcBuffer, std↩ ::uint64_t srcOffset, std::uint64_t size)=0 141

10.19.3.9 Dispatch(std::uint32_t groupSizeX, std::uint32_t groupSizeY, std::uint32_↩ t groupSizeZ)=0 141

10.19.3.10 Draw(std::uint32_t numVertices, std::uint32_t firstVertex)=0 142

10.19.3.11 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex)=0 142

10.19.3.12 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex, std::int32_↩ t vertexOffset)=0 142

10.19.3.13 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex)=0 142

10.19.3.14 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset)=0 143

10.19.3.15 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset, std::uint32_t firstInstance)=0 . . 143

10.19.3.16 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_↩ t numInstances)=0 143

10.19.3.17 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_↩ t numInstances, std::uint32_t firstInstance)=0 143

10.19.3.18 End()=0 144

10.19.3.19 EndQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 144

10.19.3.20 EndRenderCondition()=0 144

10.19.3.21 EndRenderPass()=0 144

10.19.3.22 EndStreamOutput()=0 145

10.19.3.23 SetClearColor(const ColorRGBAf &color)=0 145
10.19.3.24 SetClearDepth(float depth)=0 145
10.19.3.25 SetClearStencil(std::uint32_t stencil)=0 145
10.19.3.26 SetComputePipeline(ComputePipeline &computePipeline)=0 146
10.19.3.27 SetComputeResourceHeap(ResourceHeap &resourceHeap, std::uint32_t first↩ Set=0)=0 146
10.19.3.28 SetGraphicsAPIDependentState(const void ∗stateDesc, std::size_t stateDesc↩ Size)=0 146
10.19.3.29 SetGraphicsPipeline(GraphicsPipeline &graphicsPipeline)=0 147
10.19.3.30 SetGraphicsResourceHeap(ResourceHeap &resourceHeap, std::uint32_t first↩ Set=0)=0 148
10.19.3.31 SetIndexBuffer(Buffer &buffer)=0 148
10.19.3.32 SetScissor(const Scissor &scissor)=0 148
10.19.3.33 SetScissors(std::uint32_t numScissors, const Scissor ∗scissors)=0 149
10.19.3.34 SetStreamOutputBuffer(Buffer &buffer)=0 149
10.19.3.35 SetStreamOutputBufferArray(BufferArray &bufferArray)=0 149
10.19.3.36 SetVertexBuffer(Buffer &buffer)=0 150
10.19.3.37 SetVertexBufferArray(BufferArray &bufferArray)=0 150
10.19.3.38 SetViewport(const Viewport &viewport)=0 150
10.19.3.39 SetViewports(std::uint32_t numViewports, const Viewport ∗viewports)=0 150
10.19.3.40 UpdateBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, const void ∗data, std↩ ::uint16_t dataSize)=0 151
10.20 LLGL::CommandBufferDescriptor Struct Reference 152
10.20.1 Detailed Description 152
10.20.2 Member Data Documentation 152
10.20.2.1 flags 152
10.20.2.2 numNativeBuffers 152
10.21 LLGL::CommandBufferExt Class Reference 153
10.21.1 Detailed Description 153
10.21.2 Constructor & Destructor Documentation 154
10.21.2.1 CommandBufferExt()=default 154
10.21.3 Member Function Documentation 154

10.21.3.1 ResetResourceSlots(const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_t numSlots, long stageFlags=StageFlags::AllStages)=0 154
10.21.3.2 SetConstantBuffer(Buffer &buffer, std::uint32_t slot, long stageFlags=Stage←Flags::AllStages)=0 154
10.21.3.3 SetSampler(Sampler &sampler, std::uint32_t slot, long stageFlags=StageFlags←::AllStages)=0 155
10.21.3.4 SetStorageBuffer(Buffer &buffer, std::uint32_t slot, long stageFlags=Stage←Flags::AllStages)=0 155
10.21.3.5 SetTexture(Texture &texture, std::uint32_t slot, long stageFlags=StageFlags::←AllStages)=0 155
10.22LLGL::CommandBufferFlags Struct Reference 156
10.22.1 Detailed Description 156
10.22.2 Member Enumeration Documentation 156
10.22.2.1 anonymous enum 156
10.23LLGL::CommandQueue Class Reference 156
10.23.1 Detailed Description 157
10.23.2 Constructor & Destructor Documentation 157
10.23.2.1 CommandQueue()=default 157
10.23.3 Member Function Documentation 157
10.23.3.1 QueryResult(QueryHeap &queryHeap, std::uint32_t firstQuery, std::uint32_t numQueries, void ∗data, std::size_t dataSize)=0 157
10.23.3.2 Submit(CommandBuffer &commandBuffer)=0 158
10.23.3.3 Submit(Fence &fence)=0 158
10.23.3.4 WaitFence(Fence &fence, std::uint64_t timeout)=0 158
10.23.3.5 WaitIdle()=0 159
10.24LLGL::ComputePipeline Class Reference 159
10.24.1 Detailed Description 160
10.25LLGL::ComputePipelineDescriptor Struct Reference 160
10.25.1 Detailed Description 160
10.25.2 Constructor & Destructor Documentation 160
10.25.2.1 ComputePipelineDescriptor()=default 160
10.25.2.2 ComputePipelineDescriptor(ShaderProgram ∗shaderProgram, PipelineLayout ∗pipelineLayout=nullptr) 160

10.25.3 Member Data Documentation 161
10.25.3.1 pipelineLayout 161
10.25.3.2 shaderProgram 161
10.26 LLGL::DepthBiasDescriptor Struct Reference 161
10.26.1 Detailed Description 162
10.26.2 Member Data Documentation 162
10.26.2.1 clamp 162
10.26.2.2 constantFactor 162
10.26.2.3 slopeFactor 162
10.27 LLGL::DepthDescriptor Struct Reference 162
10.27.1 Detailed Description 163
10.27.2 Member Data Documentation 163
10.27.2.1 compareOp 163
10.27.2.2 testEnabled 163
10.27.2.3 writeEnabled 163
10.28 LLGL::Display Class Reference 163
10.28.1 Detailed Description 164
10.28.2 Member Function Documentation 165
10.28.2.1 FinalizeDisplayModes(std::vector< DisplayModeDescriptor > &displayModeDescs) 165
10.28.2.2 GetDeviceName() const =0 165
10.28.2.3 GetDisplayMode() const =0 165
10.28.2.4 GetOffset() const =0 165
10.28.2.5 IsCursorShown() 165
10.28.2.6 IsPrimary() const =0 166
10.28.2.7 QueryList() 166
10.28.2.8 QueryPrimary() 166
10.28.2.9 QuerySupportedDisplayModes() const =0 166
10.28.2.10 ResetDisplayMode()=0 166
10.28.2.11 SetDisplayMode(const DisplayModeDescriptor &displayModeDesc)=0 166
10.28.2.12 ShowCursor(bool show) 167

10.29 LLGL::DisplayModeDescriptor Struct Reference 167
10.29.1 Detailed Description 168
10.29.2 Member Data Documentation 168
10.29.2.1 refreshRate 168
10.29.2.2 resolution 168
10.30 LLGL::DstImageDescriptor Struct Reference 168
10.30.1 Detailed Description 169
10.30.2 Constructor & Destructor Documentation 169
10.30.2.1 DstImageDescriptor()=default 169
10.30.2.2 DstImageDescriptor(const DstImageDescriptor &)=default 169
10.30.2.3 DstImageDescriptor(ImageFormat format, DataType dataType, void ∗data, std↩ ::size_t dataSize) 169
10.30.3 Member Data Documentation 169
10.30.3.1 data 169
10.30.3.2 dataSize 169
10.30.3.3 dataType 170
10.30.3.4 format 170
10.31 LLGL::Canvas::EventListener Class Reference 170
10.31.1 Detailed Description 170
10.31.2 Constructor & Destructor Documentation 170
10.31.2.1 ∼EventListener()=default 170
10.31.3 Member Function Documentation 170
10.31.3.1 OnProcessEvents(Canvas &sender) 170
10.31.4 Friends And Related Function Documentation 171
10.31.4.1 Canvas 171
10.32 LLGL::Window::EventListener Class Reference 171
10.32.1 Detailed Description 172
10.32.2 Constructor & Destructor Documentation 172
10.32.2.1 ∼EventListener()=default 172
10.32.3 Member Function Documentation 172
10.32.3.1 OnChar(Window &sender, wchar_t chr) 172

10.32.3.2 OnDoubleClick(Window &sender, Key keyCode) 172
10.32.3.3 OnGetFocus(Window &sender) 172
10.32.3.4 OnGlobalMotion(Window &sender, const Offset2D &motion) 172
10.32.3.5 OnKeyDown(Window &sender, Key keyCode) 173
10.32.3.6 OnKeyUp(Window &sender, Key keyCode) 173
10.32.3.7 OnLocalMotion(Window &sender, const Offset2D &position) 173
10.32.3.8 OnLoseFocus(Window &sender) 173
10.32.3.9 OnProcessEvents(Window &sender) 173
10.32.3.10 OnQuit(Window &sender) 173
10.32.3.11 OnResize(Window &sender, const Extent2D &clientAreaSize) 174
10.32.3.12 OnTimer(Window &sender, std::uint32_t timerID) 174
10.32.3.13 OnWheelMotion(Window &sender, int motion) 174
10.32.4 Friends And Related Function Documentation 174
10.32.4.1 Window 174
10.33 LLGL::Extent2D Struct Reference 174
10.33.1 Detailed Description 175
10.33.2 Constructor & Destructor Documentation 175
10.33.2.1 Extent2D()=default 175
10.33.2.2 Extent2D(const Extent2D &)=default 175
10.33.2.3 Extent2D(std::uint32_t width, std::uint32_t height) 175
10.33.3 Member Data Documentation 175
10.33.3.1 height 175
10.33.3.2 width 175
10.34 LLGL::Extent3D Struct Reference 175
10.34.1 Detailed Description 176
10.34.2 Constructor & Destructor Documentation 176
10.34.2.1 Extent3D()=default 176
10.34.2.2 Extent3D(const Extent3D &)=default 176
10.34.2.3 Extent3D(std::uint32_t width, std::uint32_t height, std::uint32_t depth) 176
10.34.3 Member Data Documentation 176

10.34.3.1 depth 176
10.34.3.2 height 176
10.34.3.3 width 176
10.35LLGL::Fence Class Reference 177
10.35.1 Detailed Description 177
10.36LLGL::FrameProfile Struct Reference 177
10.36.1 Detailed Description 179
10.36.2 Constructor & Destructor Documentation 179
10.36.2.1 FrameProfile() 179
10.36.3 Member Function Documentation 179
10.36.3.1 Accumulate(const FrameProfile &rhs) 179
10.36.3.2 Clear() 179
10.36.4 Member Data Documentation 179
10.36.4.1 "@4 179
10.36.4.2 attachmentClears 179
10.36.4.3 bufferCopies 180
10.36.4.4 bufferMappings 180
10.36.4.5 bufferReads 180
10.36.4.6 bufferUpdates 180
10.36.4.7 bufferWrites 180
10.36.4.8 commandBufferEncodings 180
10.36.4.9 commandBufferSubmittions 181
10.36.4.10computePipelineBindings 181
10.36.4.11computeResourceHeapBindings 181
10.36.4.12constantBufferBindings 181
10.36.4.13dispatchCommands 181
10.36.4.14drawCommands 182
10.36.4.15fenceSubmissions 182
10.36.4.16graphicsPipelineBindings 182
10.36.4.17graphicsResourceHeapBindings 182

10.36.4.18 indexBufferBindings . . . 182
10.36.4.19 mipMapsGenerations . . . 183
10.36.4.20 querySections . . . 183
10.36.4.21 renderConditionSections . . . 183
10.36.4.22 renderPassSections . . . 183
10.36.4.23 samplerBindings . . . 183
10.36.4.24 storageBufferBindings . . . 184
10.36.4.25 streamOutputBufferBindings . . . 184
10.36.4.26 streamOutputSections . . . 184
10.36.4.27 textureBindings . . . 184
10.36.4.28 textureCopies . . . 184
10.36.4.29 textureMappings . . . 185
10.36.4.30 textureReads . . . 185
10.36.4.31 textureWrites . . . 185
10.36.4.32 values . . . 185
10.36.4.33 vertexBufferBindings . . . 185
10.37 LLGL::GraphicsPipeline Class Reference . . . 186
10.37.1 Detailed Description . . . 186
10.38 LLGL::GraphicsPipelineDescriptor Struct Reference . . . 186
10.38.1 Detailed Description . . . 187
10.38.2 Member Data Documentation . . . 187
10.38.2.1 blend . . . 187
10.38.2.2 depth . . . 187
10.38.2.3 pipelineLayout . . . 188
10.38.2.4 primitiveTopology . . . 188
10.38.2.5 rasterizer . . . 188
10.38.2.6 renderPass . . . 188
10.38.2.7 scissors . . . 188
10.38.2.8 shaderProgram . . . 189
10.38.2.9 stencil . . . 189

10.38.2.10 viewports 189
10.39 LLGL::Image Class Reference 189
10.39.1 Detailed Description 191
10.39.2 Constructor & Destructor Documentation 192
10.39.2.1 Image()=default 192
10.39.2.2 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType) 192
10.39.2.3 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, const ColorRGBAd &fillColor) 192
10.39.2.4 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) 192
10.39.2.5 Image(const Image &rhs) 192
10.39.2.6 Image(Image &&rhs) 192
10.39.3 Member Function Documentation 192
10.39.3.1 Blit(Offset3D dstRegionOffset, const Image &srcImage, Offset3D srcRegion↩ Offset, Extent3D srcRegionExtent) 192
10.39.3.2 Convert(const ImageFormat format, const DataType dataType, std::size_↩ t threadCount=0) 193
10.39.3.3 Fill(Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor) 193
10.39.3.4 GetBytesPerPixel() const 193
10.39.3.5 GetData() const 194
10.39.3.6 GetData() 194
10.39.3.7 GetDataSize() const 194
10.39.3.8 GetDataType() const 194
10.39.3.9 GetDepthStride() const 194
10.39.3.10 GetExtent() const 194
10.39.3.11 GetFormat() const 194
10.39.3.12 GetNumPixels() const 195
10.39.3.13 GetRowStride() const 195
10.39.3.14 IsRegionInside(const Offset3D &offset, const Extent3D &extent) const 195
10.39.3.15 MirrorXYPlane() 195
10.39.3.16 MirrorXZPlane() 195
10.39.3.17 MirrorYZPlane() 195

10.39.3.18 operator=(const Image &rhs) . . . 196
10.39.3.19 operator=(Image &&rhs) . . . 196
10.39.3.20 QueryDstDesc() . . . 196
10.39.3.21 QuerySrcDesc() const . . . 196
10.39.3.22 ReadPixels(const Offset3D &offset, const Extent3D &extent, const DstImage←Descriptor &imageDesc, std::size_t threadCount=0) const . . . 196
10.39.3.23 Release() . . . 197
10.39.3.24 Reset() . . . 197
10.39.3.25 Reset(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) . . . 197
10.39.3.26 Resize(const Extent3D &extent) . . . 197
10.39.3.27 Resize(const Extent3D &extent, const ColorRGBAd &fillColor) . . . 197
10.39.3.28 Resize(const Extent3D &extent, const ColorRGBAd &fillColor, const Offset3←D &offset) . . . 198
10.39.3.29 Resize(const Extent3D &extent, const SamplerFilter filter) . . . 198
10.39.3.30 Swap(Image &rhs) . . . 198
10.39.3.31 WritePixels(const Offset3D &offset, const Extent3D &extent, const SrcImage←Descriptor &imageDesc, std::size_t threadCount=0) . . . 198
10.40 LLGL::ImageInitialization Struct Reference . . . 199
10.40.1 Detailed Description . . . 199
10.40.2 Member Data Documentation . . . 199
10.40.2.1 clearValue . . . 199
10.40.2.2 enabled . . . 199
10.41 LLGL::BufferDescriptor::IndexBuffer Struct Reference . . . 200
10.41.1 Detailed Description . . . 200
10.41.2 Member Data Documentation . . . 200
10.41.2.1 format . . . 200
10.42 LLGL::IndexFormat Class Reference . . . 200
10.42.1 Detailed Description . . . 201
10.42.2 Constructor & Destructor Documentation . . . 201
10.42.2.1 IndexFormat()=default . . . 201
10.42.2.2 IndexFormat(const IndexFormat &)=default . . . 201

10.42.2.3 IndexFormat(const DataType dataType) 201
10.42.3 Member Function Documentation 201
10.42.3.1 GetDataType() const 201
10.42.3.2 GetFormatSize() const 201
10.42.3.3 operator=(const IndexFormat &)=default 202
10.43 LLGL::Input Class Reference 202
10.43.1 Detailed Description 203
10.43.2 Constructor & Destructor Documentation 203
10.43.2.1 Input() 203
10.43.3 Member Function Documentation 203
10.43.3.1 GetAnyKeyCount() const 203
10.43.3.2 GetEnteredChars() const 204
10.43.3.3 GetMouseMotion() const 204
10.43.3.4 GetMousePosition() const 204
10.43.3.5 GetWheelMotion() const 204
10.43.3.6 KeyDoubleClick(Key keyCode) const 204
10.43.3.7 KeyDown(Key keyCode) const 204
10.43.3.8 KeyDownRepeated(Key keyCode) const 204
10.43.3.9 KeyPressed(Key keyCode) const 204
10.43.3.10 KeyUp(Key keyCode) const 204
10.43.3.11 OnChar(Window &sender, wchar_t chr) override 205
10.43.3.12 OnDoubleClick(Window &sender, Key keyCode) override 205
10.43.3.13 OnGlobalMotion(Window &sender, const Offset2D &motion) override 205
10.43.3.14 OnKeyDown(Window &sender, Key keyCode) override 205
10.43.3.15 OnKeyUp(Window &sender, Key keyCode) override 205
10.43.3.16 OnLocalMotion(Window &sender, const Offset2D &position) override 205
10.43.3.17 OnLoseFocus(Window &sender) override 205
10.43.3.18 OnProcessEvents(Window &sender) override 206
10.43.3.19 OnWheelMotion(Window &sender, int motion) override 206
10.44 LLGL::RenderingDebugger::Message Class Reference 206

10.44.1 Detailed Description . . . 207
10.44.2 Constructor & Destructor Documentation . . . 207
10.44.2.1 Message()=default . . . 207
10.44.2.2 Message(const Message &)=default . . . 207
10.44.2.3 Message(const std::string &text, const std::string &source) . . . 207
10.44.3 Member Function Documentation . . . 207
10.44.3.1 Block() . . . 207
10.44.3.2 BlockAfter(std::size_t occurrences) . . . 207
10.44.3.3 GetOccurrences() const . . . 207
10.44.3.4 GetSource() const . . . 207
10.44.3.5 GetText() const . . . 207
10.44.3.6 IncOccurrence() . . . 208
10.44.3.7 IsBlocked() const . . . 208
10.44.3.8 operator=(const Message &)=default . . . 208
10.44.4 Friends And Related Function Documentation . . . 208
10.44.4.1 RenderingDebugger . . . 208
10.45LLGL::MultiSamplingDescriptor Struct Reference . . . 208
10.45.1 Detailed Description . . . 208
10.45.2 Constructor & Destructor Documentation . . . 208
10.45.2.1 MultiSamplingDescriptor()=default . . . 208
10.45.2.2 MultiSamplingDescriptor(std::uint32_t samples, std::uint32_t sampleMask=∼0) . 208
10.45.3 Member Function Documentation . . . 209
10.45.3.1 SampleCount() const . . . 209
10.45.4 Member Data Documentation . . . 209
10.45.4.1 enabled . . . 209
10.45.4.2 sampleMask . . . 209
10.45.4.3 samples . . . 209
10.46LLGL::NativeContextHandle Struct Reference . . . 210
10.46.1 Detailed Description . . . 210
10.46.2 Member Data Documentation . . . 210

10.46.2.1 colorMap 210
10.46.2.2 display 210
10.46.2.3 parentView 210
10.46.2.4 parentWindow 210
10.46.2.5 parentWindow 210
10.46.2.6 parentWindow 210
10.46.2.7 screen 210
10.46.2.8 visual 210
10.47LLGL::NativeHandle Struct Reference 211
10.47.1 Detailed Description 211
10.47.2 Member Data Documentation 211
10.47.2.1 display 211
10.47.2.2 view 211
10.47.2.3 visual 211
10.47.2.4 window 211
10.47.2.5 window 211
10.47.2.6 window 211
10.48LLGL::NonCopyable Class Reference 212
10.48.1 Detailed Description 212
10.48.2 Constructor & Destructor Documentation 213
10.48.2.1 NonCopyable(const NonCopyable &)=delete 213
10.48.2.2 ∼NonCopyable()=default 213
10.48.2.3 NonCopyable()=default 213
10.48.3 Member Function Documentation 213
10.48.3.1 operator=(const NonCopyable &)=delete 213
10.49LLGL::Offset2D Struct Reference 213
10.49.1 Detailed Description 213
10.49.2 Constructor & Destructor Documentation 214
10.49.2.1 Offset2D()=default 214
10.49.2.2 Offset2D(const Offset2D &)=default 214

10.49.2.3 Offset2D(std::int32_t x, std::int32_t y) 214
10.49.3 Member Data Documentation 214
10.49.3.1 x 214
10.49.3.2 y 214
10.50 LLGL::Offset3D Struct Reference 214
10.50.1 Detailed Description 215
10.50.2 Constructor & Destructor Documentation 215
10.50.2.1 Offset3D()=default 215
10.50.2.2 Offset3D(const Offset3D &)=default 215
10.50.2.3 Offset3D(std::int32_t x, std::int32_t y, std::int32_t z) 215
10.50.3 Member Data Documentation 215
10.50.3.1 x 215
10.50.3.2 y 215
10.50.3.3 z 215
10.51 LLGL::OpenGLDependentStateDescriptor Struct Reference 215
10.51.1 Detailed Description 216
10.51.2 Member Data Documentation 216
10.51.2.1 invertFrontFace 216
10.51.2.2 originLowerLeft 216
10.52 LLGL::PipelineLayout Class Reference 216
10.52.1 Detailed Description 217
10.53 LLGL::PipelineLayoutDescriptor Struct Reference 217
10.53.1 Detailed Description 217
10.53.2 Member Data Documentation 217
10.53.2.1 bindings 217
10.54 LLGL::ProfileOpenGLDescriptor Struct Reference 218
10.54.1 Detailed Description 218
10.54.2 Member Data Documentation 218
10.54.2.1 contextProfile 218
10.54.2.2 majorVersion 218

10.54.2.3 minorVersion 218

10.55 LLGL::QueryHeap Class Reference 219

10.55.1 Detailed Description 219

10.55.2 Constructor & Destructor Documentation 219

10.55.2.1 QueryHeap(const QueryType type) 219

10.55.3 Member Function Documentation 219

10.55.3.1 GetType() const 219

10.56 LLGL::QueryHeapDescriptor Struct Reference 220

10.56.1 Detailed Description 220

10.56.2 Member Data Documentation 220

10.56.2.1 numQueries 220

10.56.2.2 renderCondition 220

10.56.2.3 type 221

10.57 LLGL::QueryPipelineStatistics Struct Reference 221

10.57.1 Detailed Description 222

10.57.2 Member Data Documentation 222

10.57.2.1 clippingInvocations 222

10.57.2.2 clippingPrimitives 222

10.57.2.3 computeShaderInvocations 222

10.57.2.4 fragmentShaderInvocations 222

10.57.2.5 geometryShaderInvocations 222

10.57.2.6 geometryShaderPrimitives 222

10.57.2.7 inputAssemblyPrimitives 223

10.57.2.8 inputAssemblyVertices 223

10.57.2.9 tessControlShaderInvocations 223

10.57.2.10 tessEvaluationShaderInvocations 223

10.57.2.11 vertexShaderInvocations 223

10.58 LLGL::RasterizerDescriptor Struct Reference 223

10.58.1 Detailed Description 224

10.58.2 Member Data Documentation 224

10.58.2.1 antiAliasedLineEnabled 224
10.58.2.2 conservativeRasterization 224
10.58.2.3 cullMode 224
10.58.2.4 depthBias 225
10.58.2.5 depthClampEnabled 225
10.58.2.6 discardEnabled 225
10.58.2.7 frontCCW 225
10.58.2.8 lineWidth 225
10.58.2.9 multiSampling 225
10.58.2.10 polygonMode 226
10.58.2.11 scissorTestEnabled 226
10.59 LLGL::RenderContext Class Reference 226
10.59.1 Detailed Description 227
10.59.2 Constructor & Destructor Documentation 228
10.59.2.1 RenderContext()=default 228
10.59.2.2 RenderContext(const VideoModeDescriptor &initialVideoMode, const Vsync↩ Descriptor &initialVsync) 228
10.59.3 Member Function Documentation 228
10.59.3.1 GetNumColorAttachments() const final 228
10.59.3.2 GetResolution() const final 228
10.59.3.3 GetSurface() const 228
10.59.3.4 GetVideoMode() const 228
10.59.3.5 GetVsync() const 229
10.59.3.6 HasDepthAttachment() const final 229
10.59.3.7 HasStencilAttachment() const final 229
10.59.3.8 IsRenderContext() const final 229
10.59.3.9 OnSetVideoMode(const VideoModeDescriptor &videoModeDesc)=0 229
10.59.3.10 OnSetVsync(const VsyncDescriptor &vsyncDesc)=0 230
10.59.3.11 Present()=0 230
10.59.3.12 QueryColorFormat() const =0 230
10.59.3.13 QueryDepthStencilFormat() const =0 231

10.59.3.14 SetDisplayModeByVideoMode(Display &display, const VideoModeDescriptor &videoModeDesc) . . . 231
10.59.3.15 SetOrCreateSurface(const std::shared_ptr< Surface > &surface, VideoMode←Descriptor videoModeDesc, const void ∗windowContext) . . . 231
10.59.3.16 SetVideoMode(const VideoModeDescriptor &videoModeDesc) . . . 232
10.59.3.17 SetVsync(const VsyncDescriptor &vsyncDesc) . . . 232
10.59.3.18 ShareSurfaceAndConfig(RenderContext &other) . . . 232
10.59.3.19 SwitchFullscreenMode(const VideoModeDescriptor &videoModeDesc) . . . 233
10.60 LLGL::RenderContextDescriptor Struct Reference . . . 233
10.60.1 Detailed Description . . . 233
10.60.2 Member Data Documentation . . . 233
10.60.2.1 debugCallback . . . 233
10.60.2.2 multiSampling . . . 234
10.60.2.3 profileOpenGL . . . 234
10.60.2.4 videoMode . . . 234
10.60.2.5 vsync . . . 234
10.61 LLGL::RendererID Struct Reference . . . 234
10.61.1 Detailed Description . . . 235
10.61.2 Member Data Documentation . . . 235
10.61.2.1 Direct3D10 . . . 235
10.61.2.2 Direct3D11 . . . 235
10.61.2.3 Direct3D12 . . . 235
10.61.2.4 Direct3D9 . . . 235
10.61.2.5 Metal . . . 235
10.61.2.6 OpenGL . . . 235
10.61.2.7 OpenGLES1 . . . 235
10.61.2.8 OpenGLES2 . . . 236
10.61.2.9 OpenGLES3 . . . 236
10.61.2.10 Reserved . . . 236
10.61.2.11 Undefined . . . 236
10.61.2.12 Vulkan . . . 236

10.62LLGL::RendererInfo Struct Reference 236
10.62.1 Detailed Description 236
10.62.2 Member Data Documentation 237
10.62.2.1 deviceName 237
10.62.2.2 rendererName 237
10.62.2.3 shadingLanguageName 237
10.62.2.4 vendorName 237
10.63LLGL::RenderingCapabilities Struct Reference 237
10.63.1 Detailed Description 238
10.63.2 Member Data Documentation 238
10.63.2.1 clippingRange 238
10.63.2.2 features 238
10.63.2.3 limits 238
10.63.2.4 screenOrigin 238
10.63.2.5 shadingLanguages 238
10.63.2.6 textureFormats 239
10.64LLGL::RenderingDebugger Class Reference 239
10.64.1 Detailed Description 239
10.64.2 Constructor & Destructor Documentation 240
10.64.2.1 ∼RenderingDebugger() 240
10.64.3 Member Function Documentation 240
10.64.3.1 OnError(ErrorType type, Message &message) 240
10.64.3.2 OnWarning(WarningType type, Message &message) 240
10.64.3.3 PostError(const ErrorType type, const std::string &message) 240
10.64.3.4 PostWarning(const WarningType type, const std::string &message) 241
10.64.3.5 SetSource(const char ∗source) 241
10.65LLGL::RenderingFeatures Struct Reference 241
10.65.1 Detailed Description 242
10.65.2 Member Data Documentation 242
10.65.2.1 has3DTextures 242

10.65.2.2 hasArrayTextures 243
10.65.2.3 hasCommandBufferExt 243
10.65.2.4 hasComputeShaders 243
10.65.2.5 hasConservativeRasterization 243
10.65.2.6 hasConstantBuffers 243
10.65.2.7 hasCubeArrayTextures 244
10.65.2.8 hasCubeTextures 244
10.65.2.9 hasGeometryShaders 244
10.65.2.10 hasInstancing 244
10.65.2.11 hasLogicOp 244
10.65.2.12 hasMultiSampleTextures 245
10.65.2.13 hasOffsetInstancing 245
10.65.2.14 hasPipelineStatistics 245
10.65.2.15 hasRenderCondition 245
10.65.2.16 hasRenderTargets 245
10.65.2.17 hasSamplers 245
10.65.2.18 hasStorageBuffers 246
10.65.2.19 hasStreamOutputs 246
10.65.2.20 hasTessellationShaders 246
10.65.2.21 hasUniforms 246
10.65.2.22 hasViewportArrays 246
10.66 LLGL::RenderingLimits Struct Reference 246
10.66.1 Detailed Description 247
10.66.2 Member Data Documentation 247
10.66.2.1 lineWidthRange 247
10.66.2.2 max1DTextureSize 248
10.66.2.3 max2DTextureSize 248
10.66.2.4 max3DTextureSize 248
10.66.2.5 maxAnisotropy 248
10.66.2.6 maxBufferSize 248

10.66.2.7 maxColorAttachments 249
10.66.2.8 maxComputeShaderWorkGroups 249
10.66.2.9 maxComputeShaderWorkGroupSize 249
10.66.2.10 maxConstantBufferSize 249
10.66.2.11 maxCubeTextureSize 249
10.66.2.12 maxPatchVertices 250
10.66.2.13 maxTextureArrayLayers 250
10.66.2.14 maxViewports 250
10.66.2.15 maxViewportSize 250
10.67 LLGL::RenderingProfiler Class Reference 250
10.67.1 Detailed Description 251
10.67.2 Member Function Documentation 251
10.67.2.1 Accumulate(const FrameProfile &profile) 251
10.67.2.2 NextProfile(FrameProfile ∗outputProfile=nullptr) 251
10.67.3 Member Data Documentation 252
10.67.3.1 frameProfile 252
10.68 LLGL::RenderPass Class Reference 252
10.68.1 Detailed Description 252
10.69 LLGL::RenderPassDescriptor Struct Reference 253
10.69.1 Detailed Description 253
10.69.2 Member Data Documentation 253
10.69.2.1 colorAttachments 253
10.69.2.2 depthAttachment 254
10.69.2.3 stencilAttachment 254
10.70 LLGL::RenderSystem Class Reference 254
10.70.1 Detailed Description 257
10.70.2 Constructor & Destructor Documentation 257
10.70.2.1 RenderSystem()=default 257
10.70.3 Member Function Documentation 257
10.70.3.1 AssertCreateBuffer(const BufferDescriptor &desc, std::uint64_t maxSize) 257

10.70.3.2 AssertCreateBufferArray(std::uint32_t numBuffers, Buffer ∗const ∗bufferArray) . 258
10.70.3.3 AssertCreateRenderPass(const RenderPassDescriptor &desc) 258
10.70.3.4 AssertCreateRenderTarget(const RenderTargetDescriptor &desc) 258
10.70.3.5 AssertCreateShader(const ShaderDescriptor &desc) 258
10.70.3.6 AssertCreateShaderProgram(const ShaderProgramDescriptor &desc) 258
10.70.3.7 AssertImageDataSize(std::size_t dataSize, std::size_t requiredDataSize, const char ∗info=nullptr) 258
10.70.3.8 CreateBuffer(const BufferDescriptor &desc, const void ∗initialData=nullptr)=0 . . 258
10.70.3.9 CreateBufferArray(std::uint32_t numBuffers, Buffer ∗const ∗bufferArray)=0 . . . 259
10.70.3.10 CreateCommandBuffer(const CommandBufferDescriptor &desc={})=0 259
10.70.3.11 CreateCommandBufferExt(const CommandBufferDescriptor &desc={})=0 259
10.70.3.12 CreateComputePipeline(const ComputePipelineDescriptor &desc)=0 260
10.70.3.13 CreateFence()=0 260
10.70.3.14 CreateGraphicsPipeline(const GraphicsPipelineDescriptor &desc)=0 260
10.70.3.15 CreatePipelineLayout(const PipelineLayoutDescriptor &desc)=0 261
10.70.3.16 CreateQueryHeap(const QueryHeapDescriptor &desc)=0 261
10.70.3.17 CreateRenderContext(const RenderContextDescriptor &desc, const std↩ ::shared_ptr< Surface > &surface={})=0 261
10.70.3.18 CreateRenderPass(const RenderPassDescriptor &desc)=0 262
10.70.3.19 CreateRenderTarget(const RenderTargetDescriptor &desc)=0 262
10.70.3.20 CreateResourceHeap(const ResourceHeapDescriptor &desc)=0 262
10.70.3.21 CreateSampler(const SamplerDescriptor &desc)=0 263
10.70.3.22 CreateShader(const ShaderDescriptor &desc)=0 264
10.70.3.23 CreateShaderProgram(const ShaderProgramDescriptor &desc)=0 264
10.70.3.24 CreateTexture(const TextureDescriptor &textureDesc, const SrcImageDescriptor ∗imageDesc=nullptr)=0 264
10.70.3.25 FindModules() 265
10.70.3.26 GenerateMips(Texture &texture)=0 265
10.70.3.27 GenerateMips(Texture &texture, std::uint32_t baseMipLevel, std::uint32_t num↩ MipLevels, std::uint32_t baseArrayLayer=0, std::uint32_t numArrayLayers=1)=0 . 265
10.70.3.28 GetCommandQueue()=0 266
10.70.3.29 GetConfiguration() const 266

10.70.3.30 GetName() const 266
10.70.3.31 GetRendererID() const 267
10.70.3.32 GetRendererInfo() const 267
10.70.3.33 GetRenderingCaps() const 267
10.70.3.34 Load(const RenderSystemDescriptor &renderSystemDesc, RenderingProfiler ∗profiler=nullptr, RenderingDebugger ∗debugger=nullptr) 267
10.70.3.35 MapBuffer(Buffer &buffer, const CPUAccess access)=0 268
10.70.3.36 ReadTexture(const Texture &texture, std::uint32_t mipLevel, const DstImage←Descriptor &imageDesc)=0 268
10.70.3.37 Release(RenderContext &renderContext)=0 269
10.70.3.38 Release(CommandBuffer &commandBuffer)=0 269
10.70.3.39 Release(Buffer &buffer)=0 270
10.70.3.40 Release(BufferArray &bufferArray)=0 270
10.70.3.41 Release(Texture &texture)=0 270
10.70.3.42 Release(Sampler &sampler)=0 270
10.70.3.43 Release(ResourceHeap &resourceHeap)=0 270
10.70.3.44 Release(RenderPass &renderPass)=0 270
10.70.3.45 Release(RenderTarget &renderTarget)=0 270
10.70.3.46 Release(Shader &shader)=0 270
10.70.3.47 Release(ShaderProgram &shaderProgram)=0 271
10.70.3.48 Release(PipelineLayout &pipelineLayout)=0 271
10.70.3.49 Release(GraphicsPipeline &graphicsPipeline)=0 271
10.70.3.50 Release(ComputePipeline &computePipeline)=0 271
10.70.3.51 Release(QueryHeap &queryHeap)=0 271
10.70.3.52 Release(Fence &fence)=0 271
10.70.3.53 SetConfiguration(const RenderSystemConfiguration &config) 271
10.70.3.54 SetRendererInfo(const RendererInfo &info) 271
10.70.3.55 SetRenderingCaps(const RenderingCapabilities &caps) 272
10.70.3.56 Unload(std::unique_ptr< RenderSystem > &&renderSystem) 272
10.70.3.57 UnmapBuffer(Buffer &buffer)=0 272
10.70.3.58 WriteBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, const void ∗data, std←::uint64_t dataSize)=0 272

10.70.3.59WriteTexture(Texture &texture, const TextureRegion &textureRegion, const Src←ImageDescriptor &imageDesc)=0 272
10.71LLGL::RenderSystemChild Class Reference 273
10.71.1 Detailed Description 274
10.72LLGL::RenderSystemConfiguration Struct Reference 274
10.72.1 Detailed Description 274
10.72.2 Member Data Documentation 274
10.72.2.1 imageInitialization 274
10.72.2.2 threadCount 274
10.73LLGL::RenderSystemDescriptor Struct Reference 275
10.73.1 Detailed Description 275
10.73.2 Constructor & Destructor Documentation 275
10.73.2.1 RenderSystemDescriptor()=default 275
10.73.2.2 RenderSystemDescriptor(const RenderSystemDescriptor &)=default 275
10.73.2.3 RenderSystemDescriptor(const std::string &moduleName) 275
10.73.2.4 RenderSystemDescriptor(const char ∗moduleName) 276
10.73.3 Member Function Documentation 276
10.73.3.1 operator=(const RenderSystemDescriptor &)=default 276
10.73.4 Member Data Documentation 276
10.73.4.1 moduleName 276
10.73.4.2 rendererConfig 276
10.73.4.3 rendererConfigSize 277
10.74LLGL::RenderTarget Class Reference 277
10.74.1 Detailed Description 278
10.74.2 Member Function Documentation 278
10.74.2.1 GetNumColorAttachments() const =0 278
10.74.2.2 GetRenderPass() const =0 278
10.74.2.3 GetResolution() const =0 279
10.74.2.4 HasDepthAttachment() const =0 279
10.74.2.5 HasStencilAttachment() const =0 279
10.74.2.6 IsRenderContext() const 280

10.74.2.7 ValidateMipResolution(const Texture &texture, std::uint32_t mipLevel) 280
10.74.2.8 ValidateResolution(const Extent2D &resolution) 280
10.75LLGL::RenderTargetDescriptor Struct Reference 280
10.75.1 Detailed Description 281
10.75.2 Member Data Documentation 281
10.75.2.1 attachments 281
10.75.2.2 customMultiSampling 282
10.75.2.3 multiSampling 282
10.75.2.4 renderPass 282
10.75.2.5 resolution 282
10.76LLGL::Resource Class Reference 283
10.76.1 Detailed Description 283
10.76.2 Member Function Documentation 283
10.76.2.1 QueryResourceType() const =0 283
10.77LLGL::ResourceHeap Class Reference 284
10.77.1 Detailed Description 284
10.78LLGL::ResourceHeapDescriptor Struct Reference 284
10.78.1 Detailed Description 285
10.78.2 Member Data Documentation 285
10.78.2.1 pipelineLayout 285
10.78.2.2 resourceViews 285
10.79LLGL::ShaderReflectionDescriptor::ResourceView Struct Reference 286
10.79.1 Detailed Description 286
10.79.2 Member Data Documentation 286
10.79.2.1 arraySize 286
10.79.2.2 constantBufferSize 287
10.79.2.3 name 287
10.79.2.4 slot 287
10.79.2.5 stageFlags 287
10.79.2.6 storageBufferType 287

10.79.2.7 type 288
10.80 LLGL::ResourceViewDescriptor Struct Reference 288
10.80.1 Detailed Description 288
10.80.2 Constructor & Destructor Documentation 288
10.80.2.1 ResourceViewDescriptor()=default 288
10.80.2.2 ResourceViewDescriptor(Resource ∗resource) 288
10.80.3 Member Data Documentation 289
10.80.3.1 resource 289
10.81 LLGL::Sampler Class Reference 289
10.81.1 Detailed Description 289
10.81.2 Constructor & Destructor Documentation 290
10.81.2.1 Sampler()=default 290
10.81.3 Member Function Documentation 290
10.81.3.1 QueryResourceType() const override 290
10.82 LLGL::SamplerDescriptor Struct Reference 290
10.82.1 Detailed Description 291
10.82.2 Member Data Documentation 291
10.82.2.1 addressModeU 291
10.82.2.2 addressModeV 291
10.82.2.3 addressModeW 291
10.82.2.4 borderColor 291
10.82.2.5 compareEnabled 291
10.82.2.6 compareOp 291
10.82.2.7 magFilter 291
10.82.2.8 maxAnisotropy 292
10.82.2.9 maxLOD 292
10.82.2.10 minFilter 292
10.82.2.11 minLOD 292
10.82.2.12 mipMapFilter 292
10.82.2.13 mipMapLODBias 292

10.82.2.14mipMapping . . . 292
10.83LLGL::Scissor Struct Reference . . . 293
10.83.1 Detailed Description . . . 293
10.83.2 Constructor & Destructor Documentation . . . 293
10.83.2.1 Scissor()=default . . . 293
10.83.2.2 Scissor(const Scissor &)=default . . . 293
10.83.2.3 Scissor(std::int32_t x, std::int32_t y, std::int32_t width, std::int32_t height) . . . 293
10.83.2.4 Scissor(const Offset2D &offset, const Extent2D &extent) . . . 294
10.83.3 Member Data Documentation . . . 294
10.83.3.1 height . . . 294
10.83.3.2 width . . . 294
10.83.3.3 x . . . 294
10.83.3.4 y . . . 294
10.84LLGL::Shader Class Reference . . . 294
10.84.1 Detailed Description . . . 295
10.84.2 Constructor & Destructor Documentation . . . 295
10.84.2.1 Shader(const ShaderType type) . . . 295
10.84.3 Member Function Documentation . . . 295
10.84.3.1 Disassemble(int flags=0)=0 . . . 295
10.84.3.2 GetStageFlags() const . . . 296
10.84.3.3 GetType() const . . . 296
10.84.3.4 HasErrors() const =0 . . . 296
10.84.3.5 QueryInfoLog()=0 . . . 296
10.85LLGL::ShaderCompileFlags Struct Reference . . . 296
10.85.1 Detailed Description . . . 297
10.85.2 Member Enumeration Documentation . . . 297
10.85.2.1 anonymous enum . . . 297
10.86LLGL::ShaderDescriptor Struct Reference . . . 297
10.86.1 Detailed Description . . . 298
10.86.2 Constructor & Destructor Documentation . . . 298

10.86.2.1 ShaderDescriptor()=default 298
10.86.2.2 ShaderDescriptor(const ShaderDescriptor &)=default 298
10.86.2.3 ShaderDescriptor(const ShaderType type, const char ∗source) 298
10.86.2.4 ShaderDescriptor(const ShaderType type, const char ∗source, const char ∗entryPoint, const char ∗profile, long flags=0) 298
10.86.3 Member Function Documentation 299
10.86.3.1 operator=(const ShaderDescriptor &)=default 299
10.86.4 Member Data Documentation 299
10.86.4.1 entryPoint 299
10.86.4.2 flags 299
10.86.4.3 profile 299
10.86.4.4 source 299
10.86.4.5 sourceSize 300
10.86.4.6 sourceType 300
10.86.4.7 streamOutput 300
10.86.4.8 type 300
10.87 LLGL::ShaderDisassembleFlags Struct Reference 300
10.87.1 Detailed Description 301
10.87.2 Member Enumeration Documentation 301
10.87.2.1 anonymous enum 301
10.88 LLGL::ShaderProgram Class Reference 301
10.88.1 Detailed Description 302
10.88.2 Member Enumeration Documentation 302
10.88.2.1 LinkError 302
10.88.3 Member Function Documentation 302
10.88.3.1 BindConstantBuffer(const std::string &name, std::uint32_t bindingIndex)=0 302
10.88.3.2 BindStorageBuffer(const std::string &name, std::uint32_t bindingIndex)=0 303
10.88.3.3 FinalizeShaderReflection(ShaderReflectionDescriptor &reflectionDesc) 303
10.88.3.4 HasErrors() const =0 304
10.88.3.5 LinkErrorToString(const LinkError errorCode) 304
10.88.3.6 LockShaderUniform()=0 304

10.88.3.7 QueryInfoLog()=0 304
10.88.3.8 QueryReflectionDesc() const =0 305
10.88.3.9 UnlockShaderUniform()=0 305
10.88.3.10 ValidateShaderComposition(Shader ∗const ∗shaders, std::size_t numShaders) . 305
10.89 LLGL::ShaderProgramDescriptor Struct Reference 306
10.89.1 Detailed Description 306
10.89.2 Member Data Documentation 306
10.89.2.1 computeShader 306
10.89.2.2 fragmentShader 307
10.89.2.3 geometryShader 307
10.89.2.4 tessControlShader 307
10.89.2.5 tessEvaluationShader 307
10.89.2.6 vertexFormats 308
10.89.2.7 vertexShader 308
10.90 LLGL::ShaderReflectionDescriptor Struct Reference 308
10.90.1 Detailed Description 309
10.90.2 Member Data Documentation 309
10.90.2.1 resourceViews 309
10.90.2.2 streamOutputAttributes 309
10.90.2.3 uniforms 309
10.90.2.4 vertexAttributes 309
10.91 LLGL::ShaderUniform Class Reference 309
10.91.1 Detailed Description 310
10.91.2 Member Function Documentation 311
10.91.2.1 SetUniform1f(const UniformLocation location, float value0)=0 311
10.91.2.2 SetUniform1f(const char ∗name, float value0)=0 311
10.91.2.3 SetUniform1fv(const UniformLocation location, const float ∗value, std::size_↩
t count=1)=0 311
10.91.2.4 SetUniform1fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . . 311
10.91.2.5 SetUniform1i(const UniformLocation location, int value0)=0 311
10.91.2.6 SetUniform1i(const char ∗name, int value0)=0 311

10.91.2.7 SetUniform1iv(const UniformLocation location, const int ∗value, std::size_↩t count=1)=0 311
10.91.2.8 SetUniform1iv(const char ∗name, const int ∗value, std::size_t count=1)=0 311
10.91.2.9 SetUniform2f(const UniformLocation location, float value0, float value1)=0 311
10.91.2.10SetUniform2f(const char ∗name, float value0, float value1)=0 311
10.91.2.11SetUniform2fv(const UniformLocation location, const float ∗value, std::size_↩t count=1)=0 311
10.91.2.12SetUniform2fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . . 311
10.91.2.13SetUniform2i(const UniformLocation location, int value0, int value1)=0 311
10.91.2.14SetUniform2i(const char ∗name, int value0, int value1)=0 311
10.91.2.15SetUniform2iv(const UniformLocation location, const int ∗value, std::size_↩t count=1)=0 312
10.91.2.16SetUniform2iv(const char ∗name, const int ∗value, std::size_t count=1)=0 312
10.91.2.17SetUniform2x2fv(const UniformLocation location, const float ∗value, std::size_t count=1)=0 312
10.91.2.18SetUniform2x2fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . 312
10.91.2.19SetUniform3f(const UniformLocation location, float value0, float value1, float value2)=0 312
10.91.2.20SetUniform3f(const char ∗name, float value0, float value1, float value2)=0 312
10.91.2.21SetUniform3fv(const UniformLocation location, const float ∗value, std::size_↩t count=1)=0 312
10.91.2.22SetUniform3fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . . 312
10.91.2.23SetUniform3i(const UniformLocation location, int value0, int value1, int value2)=0 312
10.91.2.24SetUniform3i(const char ∗name, int value0, int value1, int value2)=0 312
10.91.2.25SetUniform3iv(const UniformLocation location, const int ∗value, std::size_↩t count=1)=0 312
10.91.2.26SetUniform3iv(const char ∗name, const int ∗value, std::size_t count=1)=0 312
10.91.2.27SetUniform3x3fv(const UniformLocation location, const float ∗value, std::size_t count=1)=0 312
10.91.2.28SetUniform3x3fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . 312
10.91.2.29SetUniform4f(const UniformLocation location, float value0, float value1, float value2, float value3)=0 312
10.91.2.30SetUniform4f(const char ∗name, float value0, float value1, float value2, float value3)=0 312

10.91.2.31 SetUniform4fv(const UniformLocation location, const float ∗value, std::size_↵t count=1)=0 . . . 313

10.91.2.32 SetUniform4fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . . 313

10.91.2.33 SetUniform4i(const UniformLocation location, int value0, int value1, int value2, int value3)=0 . . . 313

10.91.2.34 SetUniform4i(const char ∗name, int value0, int value1, int value2, int value3)=0 . 313

10.91.2.35 SetUniform4iv(const UniformLocation location, const int ∗value, std::size_↵t count=1)=0 . . . 313

10.91.2.36 SetUniform4iv(const char ∗name, const int ∗value, std::size_t count=1)=0 313

10.91.2.37 SetUniform4x4fv(const UniformLocation location, const float ∗value, std::size_t count=1)=0 . . . 313

10.91.2.38 SetUniform4x4fv(const char ∗name, const float ∗value, std::size_t count=1)=0 . . 313

10.92 LLGL::SrcImageDescriptor Struct Reference . . . 313

10.92.1 Detailed Description . . . 314

10.92.2 Constructor & Destructor Documentation . . . 314

10.92.2.1 SrcImageDescriptor()=default . . . 314

10.92.2.2 SrcImageDescriptor(const SrcImageDescriptor &)=default . . . 314

10.92.2.3 SrcImageDescriptor(ImageFormat format, DataType dataType, const void ∗data, std::size_t dataSize) . . . 314

10.92.3 Member Data Documentation . . . 314

10.92.3.1 data . . . 314

10.92.3.2 dataSize . . . 314

10.92.3.3 dataType . . . 314

10.92.3.4 format . . . 315

10.93 LLGL::StageFlags Struct Reference . . . 315

10.93.1 Detailed Description . . . 315

10.93.2 Member Enumeration Documentation . . . 315

10.93.2.1 anonymous enum . . . 315

10.94 LLGL::StencilDescriptor Struct Reference . . . 316

10.94.1 Detailed Description . . . 316

10.94.2 Member Data Documentation . . . 316

10.94.2.1 back . . . 316

10.94.2.2 front . . . 317

10.94.2.3 testEnabled 317
10.95 LLGL::StencilFaceDescriptor Struct Reference 317
10.95.1 Detailed Description 318
10.95.2 Member Data Documentation 318
10.95.2.1 compareOp 318
10.95.2.2 depthFailOp 318
10.95.2.3 depthPassOp 318
10.95.2.4 readMask 318
10.95.2.5 reference 318
10.95.2.6 stencilFailOp 319
10.95.2.7 writeMask 319
10.96 LLGL::BufferDescriptor::StorageBuffer Struct Reference 319
10.96.1 Detailed Description 319
10.96.2 Member Data Documentation 320
10.96.2.1 format 320
10.96.2.2 storageType 320
10.96.2.3 stride 320
10.97 LLGL::ShaderDescriptor::StreamOutput Struct Reference 320
10.97.1 Detailed Description 321
10.97.2 Member Data Documentation 321
10.97.2.1 format 321
10.98 LLGL::StreamOutputAttribute Struct Reference 321
10.98.1 Detailed Description 321
10.98.2 Constructor & Destructor Documentation 322
10.98.2.1 StreamOutputAttribute()=default 322
10.98.2.2 StreamOutputAttribute(const StreamOutputAttribute &)=default 322
10.98.3 Member Function Documentation 322
10.98.3.1 operator=(const StreamOutputAttribute &)=default 322
10.98.4 Member Data Documentation 322
10.98.4.1 components 322

10.98.4.2 name 322
10.98.4.3 outputSlot 322
10.98.4.4 semanticIndex 322
10.98.4.5 startComponent 323
10.98.4.6 stream 323
10.99 LLGL::StreamOutputFormat Struct Reference 323
10.99.1 Detailed Description 323
10.99.2 Member Function Documentation 323
10.99.2.1 AppendAttribute(const StreamOutputAttribute &attrib) 323
10.99.2.2 AppendAttributes(const StreamOutputFormat &format) 324
10.99.3 Member Data Documentation 324
10.99.3.1 attributes 324
10.100 LLGL::Surface Class Reference 324
10.100.1 Detailed Description 325
10.100.2 Member Function Documentation 325
10.100.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc)=0 325
10.100.2.2 GetContentSize() const =0 326
10.100.2.3 GetNativeHandle(void ∗nativeHandle) const =0 326
10.100.2.4 ResetPixelFormat()=0 326
10.101 LLGL::Texture Class Reference 327
10.101.1 Detailed Description 327
10.101.2 Constructor & Destructor Documentation 327
10.101.2.1 Texture(const TextureType type) 327
10.101.3 Member Function Documentation 327
10.101.3.1 GetType() const 327
10.101.3.2 QueryDesc() const =0 328
10.101.3.3 QueryMipExtent(std::uint32_t mipLevel) const =0 328
10.101.3.4 QueryResourceType() const override 329
10.102 LLGL::TextureDescriptor Struct Reference 329
10.102.1 Detailed Description 329

10.102.2 Member Data Documentation . . . 330
10.102.2.1 arrayLayers . . . 330
10.102.2.2 extent . . . 330
10.102.2.3 flags . . . 330
10.102.2.4 format . . . 331
10.102.2.5 mipLevels . . . 331
10.102.2.6 samples . . . 331
10.102.2.7 type . . . 331
10.103 LLGL::TextureFlags Struct Reference . . . 331
10.103.1 Detailed Description . . . 332
10.103.2 Member Enumeration Documentation . . . 332
10.103.2.1 anonymous enum . . . 332
10.104 LLGL::TextureRegion Struct Reference . . . 333
10.104.1 Detailed Description . . . 333
10.104.2 Member Data Documentation . . . 333
10.104.2.1 extent . . . 333
10.104.2.2 mipLevel . . . 333
10.104.2.3 offset . . . 334
10.105 LLGL::Timer Class Reference . . . 334
10.105.1 Detailed Description . . . 335
10.105.2 Member Function Documentation . . . 335
10.105.2.1 Create() . . . 335
10.105.2.2 GetDeltaTime() const . . . 335
10.105.2.3 GetFrequency() const =0 . . . 335
10.105.2.4 IsRunning() const =0 . . . 335
10.105.2.5 MeasureTime() . . . 336
10.105.2.6 Start()=0 . . . 336
10.105.2.7 Stop()=0 . . . 336
10.106 LLGL::UniformDescriptor Struct Reference . . . 336
10.106.1 Detailed Description . . . 336

10.106.2 Member Data Documentation . . . 337
10.106.2.1 location . . . 337
10.106.2.2 name . . . 337
10.106.2.3 size . . . 337
10.106.2.4 type . . . 337
10.107 LLGL::UninitializeTag Struct Reference . . . 337
10.107.1 Detailed Description . . . 337
10.108 LLGL::VertexAttribute Struct Reference . . . 338
10.108.1 Detailed Description . . . 338
10.108.2 Constructor & Destructor Documentation . . . 338
10.108.2.1 VertexAttribute()=default . . . 338
10.108.2.2 VertexAttribute(const VertexAttribute &)=default . . . 338
10.108.2.3 VertexAttribute(const std::string &name, const Format format, std::uint32_↩
t instanceDivisor=0) . . . 338
10.108.2.4 VertexAttribute(const std::string &semanticName, std::uint32_t semanticIndex,
const Format format, std::uint32_t instanceDivisor=0) . . . 339
10.108.3 Member Function Documentation . . . 339
10.108.3.1 GetSize() const . . . 339
10.108.3.2 operator=(const VertexAttribute &)=default . . . 340
10.108.4 Member Data Documentation . . . 340
10.108.4.1 format . . . 340
10.108.4.2 instanceDivisor . . . 340
10.108.4.3 name . . . 340
10.108.4.4 offset . . . 340
10.108.4.5 semanticIndex . . . 340
10.109 LLGL::BufferDescriptor::VertexBuffer Struct Reference . . . 341
10.109.1 Detailed Description . . . 341
10.109.2 Member Data Documentation . . . 341
10.109.2.1 format . . . 341
10.110 LLGL::VertexFormat Struct Reference . . . 341
10.110.1 Detailed Description . . . 342

10.110.2 Member Function Documentation 342
10.110.2.1 AppendAttribute(const VertexAttribute &attrib, std::uint32_t offset=Constants↩ ::ignoreOffset) 342
10.110.3 Member Data Documentation 343
10.110.3.1 attributes 343
10.110.3.2 inputSlot 343
10.110.3.3 stride 343
10.111 LLGL::VideoAdapterDescriptor Struct Reference 344
10.111.1 Detailed Description 344
10.111.2 Member Data Documentation 344
10.111.2.1 name 344
10.111.2.2 outputs 344
10.111.2.3 vendor 344
10.111.2.4 videoMemory 345
10.112 LLGL::VideoModeDescriptor Struct Reference 345
10.112.1 Detailed Description 345
10.112.2 Member Data Documentation 346
10.112.2.1 colorBits 346
10.112.2.2 depthBits 346
10.112.2.3 fullscreen 346
10.112.2.4 resolution 346
10.112.2.5 stencilBits 347
10.112.2.6 swapChainSize 347
10.113 LLGL::VideoOutputDescriptor Struct Reference 347
10.113.1 Detailed Description 347
10.113.2 Member Data Documentation 348
10.113.2.1 displayModes 348
10.114 LLGL::Viewport Struct Reference 348
10.114.1 Detailed Description 349
10.114.2 Constructor & Destructor Documentation 349
10.114.2.1 Viewport()=default 349

10.114.2.2 Viewport(const Viewport &)=default 349
10.114.2.3 Viewport(float x, float y, float width, float height) 349
10.114.2.4 Viewport(float x, float y, float width, float height, float minDepth, float maxDepth) . 349
10.114.2.5 Viewport(const Extent2D &extent) 349
10.114.2.6 Viewport(const Extent2D &extent, float minDepth, float maxDepth) 349
10.114.2.7 Viewport(const Offset2D &offset, const Extent2D &extent) 349
10.114.2.8 Viewport(const Offset2D &offset, const Extent2D &extent, float minDepth, float maxDepth) 349
10.114.3 Member Data Documentation 350
10.114.3.1 height 350
10.114.3.2 maxDepth 350
10.114.3.3 minDepth 350
10.114.3.4 width 350
10.114.3.5 x 350
10.114.3.6 y 350
10.115 LLGL::VsyncDescriptor Struct Reference 351
10.115.1 Detailed Description 351
10.115.2 Member Data Documentation 351
10.115.2.1 enabled 351
10.115.2.2 interval 351
10.115.2.3 refreshRate 351
10.116 LLGL::VulkanRendererConfiguration Struct Reference 352
10.116.1 Detailed Description 352
10.116.2 Member Data Documentation 352
10.116.2.1 application 352
10.116.2.2 minDeviceMemoryAllocationSize 352
10.116.2.3 reduceDeviceMemoryFragmentation 353
10.117 LLGL::Window Class Reference 353
10.117.1 Detailed Description 355
10.117.2 Member Function Documentation 355
10.117.2.1 AdaptForVideoMode(VideoModeDescriptor &videoModeDesc) override 355

10.117.2.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 355
10.117.2.3 Create(const WindowDescriptor &desc) 355
10.117.2.4 GetBehavior() const 356
10.117.2.5 GetDesc() const =0 356
10.117.2.6 GetPosition() const =0 356
10.117.2.7 GetSize(bool useClientArea=true) const =0 356
10.117.2.8 GetTitle() const =0 356
10.117.2.9 HasFocus() const 356
10.117.2.10 IsShown() const =0 356
10.117.2.11 OnProcessEvents()=0 356
10.117.2.12 PostChar(wchar_t chr) 356
10.117.2.13 PostDoubleClick(Key keyCode) 357
10.117.2.14 PostGetFocus() 357
10.117.2.15 PostGlobalMotion(const Offset2D &motion) 357
10.117.2.16 PostKeyDown(Key keyCode) 357
10.117.2.17 PostKeyUp(Key keyCode) 357
10.117.2.18 PostLocalMotion(const Offset2D &position) 357
10.117.2.19 PostLoseFocus() 357
10.117.2.20 PostQuit() 358
10.117.2.21 PostResize(const Extent2D &clientAreaSize) 358
10.117.2.22 PostTimer(std::uint32_t timerID) 358
10.117.2.23 PostWheelMotion(int motion) 358
10.117.2.24 ProcessEvents() 358
10.117.2.25 RemoveEventListener(const EventListener ∗eventListener) 359
10.117.2.26 SetBehavior(const WindowBehavior &behavior) 359
10.117.2.27 SetDesc(const WindowDescriptor &desc)=0 359
10.117.2.28 SetPosition(const Offset2D &position)=0 359
10.117.2.29 SetSize(const Extent2D &size, bool useClientArea=true)=0 359
10.117.2.30 SetTitle(const std::wstring &title)=0 359
10.117.2.31 Show(bool show=true)=0 359

10.118LLGL::WindowBehavior Struct Reference 360
10.118.1Detailed Description 360
10.118.2Member Data Documentation 360
10.118.2.1disableClearOnResize 360
10.118.2.2moveAndResizeTimerID 360
10.119LLGL::WindowDescriptor Struct Reference 361
10.119.1Detailed Description 361
10.119.2Member Data Documentation 361
10.119.2.1acceptDropFiles 361
10.119.2.2borderless 361
10.119.2.3centered 362
10.119.2.4position 362
10.119.2.5preventForPowerSafe 362
10.119.2.6resizable 362
10.119.2.7size 363
10.119.2.8title 363
10.119.2.9visible 363
10.119.2.10windowContext 363

11 File Documentation 365
11.1 Buffer.h File Reference 365
11.2 BufferArray.h File Reference 365
11.3 BufferFlags.h File Reference 366
11.4 Canvas.h File Reference 367
11.5 CanvasFlags.h File Reference 367
11.6 Color.h File Reference 367
11.7 ColorRGB.h File Reference 369
11.8 ColorRGBA.h File Reference 369
11.9 CommandBuffer.h File Reference 370
11.10CommandBufferExt.h File Reference 370
11.11CommandBufferFlags.h File Reference 370

11.12 CommandQueue.h File Reference 371
11.13 ComputePipeline.h File Reference 372
11.14 ComputePipelineFlags.h File Reference 372
11.15 Constants.h File Reference 372
11.16 Display.h File Reference 372
11.17 DisplayFlags.h File Reference 373
11.18 Export.h File Reference 373
11.18.1 Macro Definition Documentation 374
11.18.1.1 LLGL_EXPORT 374
11.19 Fence.h File Reference 374
11.20 Format.h File Reference 374
11.21 ForwardDecls.h File Reference 376
11.22 GraphicsPipeline.h File Reference 376
11.23 GraphicsPipelineFlags.h File Reference 376
11.24 Image.h File Reference 378
11.25 ImageFlags.h File Reference 379
11.26 IndexFormat.h File Reference 380
11.27 Input.h File Reference 380
11.28 IOSNativeHandle.h File Reference 381
11.29 Key.h File Reference 381
11.30 LinuxNativeHandle.h File Reference 382
11.31 LLGL.h File Reference 383
11.32 Log.h File Reference 383
11.33 MacOSNativeHandle.h File Reference 384
11.34 NativeHandle.h File Reference 384
11.35 NonCopyable.h File Reference 384
11.36 PipelineLayout.h File Reference 385
11.37 PipelineLayoutFlags.h File Reference 385
11.38 Platform.h File Reference 385
11.39 QueryHeap.h File Reference 385

11.40 QueryHeapFlags.h File Reference . . . 386
11.41 RenderContext.h File Reference . . . 386
11.42 RenderContextFlags.h File Reference . . . 387
11.43 RenderingDebugger.h File Reference . . . 388
11.44 RenderingProfiler.h File Reference . . . 388
11.45 RenderPass.h File Reference . . . 389
11.46 RenderPassFlags.h File Reference . . . 389
11.47 RenderSystem.h File Reference . . . 390
11.48 RenderSystemChild.h File Reference . . . 390
11.49 RenderSystemFlags.h File Reference . . . 391
11.50 RenderTarget.h File Reference . . . 392
11.51 RenderTargetFlags.h File Reference . . . 393
11.52 Resource.h File Reference . . . 393
11.53 ResourceFlags.h File Reference . . . 394
11.54 ResourceHeap.h File Reference . . . 394
11.55 ResourceHeapFlags.h File Reference . . . 394
11.56 Sampler.h File Reference . . . 395
11.57 SamplerFlags.h File Reference . . . 395
11.58 Shader.h File Reference . . . 396
11.59 ShaderFlags.h File Reference . . . 396
11.60 ShaderProgram.h File Reference . . . 397
11.61 ShaderProgramFlags.h File Reference . . . 397
11.62 ShaderUniform.h File Reference . . . 398
11.63 ShaderUniformFlags.h File Reference . . . 398
11.64 StreamOutputAttribute.h File Reference . . . 399
11.65 StreamOutputFormat.h File Reference . . . 400
11.66 Strings.h File Reference . . . 400
11.67 Surface.h File Reference . . . 401
11.68 Tags.h File Reference . . . 401
11.69 Texture.h File Reference . . . 401
11.70 TextureFlags.h File Reference . . . 402
11.71 Timer.h File Reference . . . 403
11.72 Types.h File Reference . . . 403
11.73 Utility.h File Reference . . . 404
11.74 Version.h File Reference . . . 406
11.75 VertexAttribute.h File Reference . . . 406
11.76 VertexFormat.h File Reference . . . 407
11.77 VideoAdapter.h File Reference . . . 407
11.78 Win32NativeHandle.h File Reference . . . 408
11.79 Window.h File Reference . . . 408
11.80 WindowFlags.h File Reference . . . 409

Index **411**

Chapter 1

LLGL 0.02 Beta Documentation

LLGL (Low Level Graphics Library)

Overview

- **Version**: 0.02 Beta
- **License**: 3-Clause BSD License

Progress

- **OpenGL Renderer**: ∼90% done
- **Direct3D 11 Renderer**: ∼90% done
- **Direct3D 12 Renderer**: ∼15% done
- **Vulkan Renderer**: ∼30% done
- **Metal enderer**: ∼5% 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 GLsizeiptr indices = 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:  indexTypeSize_ * (static_cast<NSUInteger>(firstIndex))
            instanceCount:                  1
            baseInstance:                   0
        ];
    } else {
        [renderEncoder_
            drawIndexedPrimitives:  primitiveType_
            indexCount:             static_cast<NSUInteger>(numIndices)
            indexType:              indexType_
            indexBuffer:            indexBuffer_
            indexBufferOffset:      indexTypeSize_ * static_cast<NSUInteger>(firstIndex)
        ];
    }
}
```

Chapter 2

Todo List

Member LLGL::BlendDescriptor::blendFactor

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

Member LLGL::BufferType

Maybe replace this enum by "ResourceType".

Member LLGL::ESProfile

This is incomplete, do not use!

Member LLGL::FrameProfile::bufferReads

Not available yet.

Member LLGL::FrameProfile::textureCopies

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::ImageInitialization::clearValue

Currently only supports initialization of color and depth. Default initialization of stencil values is not supported yet.

Class LLGL::RenderingProfiler

Refactor this for the new ResourceHeap and RenderPass interfaces.

Class LLGL::ResourceHeap

Maybe rename to "ResourceViewHeap" again?

Class LLGL::ShaderUniform

Complete documentation.

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. 26
Hardware format utility functions. 27
Data type utility functions. 30
Image utility functions to classify and convert image data. 31
Global type-to-string conversion functions. 36
Texture utility functions to determine texture dimension and buffer sizes. 38
Global utility functions, especially to fill descriptor structures. 41

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

LLGL 49

LLGL::Constants

Namespace with all constants used as default arguments 94

LLGL::Log 94

LLGL::Version

Namespace with functions to determine LLGL version 97

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

LLGL::ApplicationDescriptor 99
LLGL::AttachmentClear 100
LLGL::AttachmentDescriptor 102
LLGL::AttachmentFormatDescriptor 104
LLGL::BindingDescriptor 106
LLGL::BlendDescriptor 108
LLGL::BlendTargetDescriptor 110
LLGL::BufferDescriptor 114
LLGL::BufferFlags 116
LLGL::CanvasDescriptor 119
LLGL::ClearFlags 120
LLGL::ClearValue 121
LLGL::Color< T, N > 122
LLGL::Color< bool > 122
LLGL::Color< float > 122
LLGL::Color< T, 3u > 125
LLGL::Color< T, 4u > 130
LLGL::CommandBufferDescriptor 152
LLGL::CommandBufferFlags 156
LLGL::ComputePipelineDescriptor 160
LLGL::DepthBiasDescriptor 161
LLGL::DepthDescriptor 162
LLGL::DisplayModeDescriptor 167
LLGL::DstImageDescriptor 168
LLGL::Canvas::EventListener 170
LLGL::Window::EventListener 171
    LLGL::Input 202
LLGL::Extent2D 174
LLGL::Extent3D 175
LLGL::FrameProfile 177
LLGL::GraphicsPipelineDescriptor 186
LLGL::Image 189
LLGL::ImageInitialization 199
LLGL::BufferDescriptor::IndexBuffer 200
LLGL::IndexFormat 200

LLGL::RenderingDebugger::Message 206
LLGL::MultiSamplingDescriptor 208
LLGL::NativeContextHandle 210
LLGL::NativeHandle 211
LLGL::NonCopyable 212
  LLGL::Display 163
  LLGL::RenderSystem 254
  LLGL::RenderSystemChild 273
    LLGL::BufferArray 113
    LLGL::CommandBuffer 135
      LLGL::CommandBufferExt 153
    LLGL::CommandQueue 156
    LLGL::ComputePipeline 159
    LLGL::Fence 177
    LLGL::GraphicsPipeline 186
    LLGL::PipelineLayout 216
    LLGL::QueryHeap 219
    LLGL::RenderPass 252
    LLGL::RenderTarget 277
      LLGL::RenderContext 226
    LLGL::Resource 283
      LLGL::Buffer 112
      LLGL::Sampler 289
      LLGL::Texture 327
    LLGL::ResourceHeap 284
    LLGL::Shader 294
    LLGL::ShaderProgram 301
  LLGL::ShaderUniform 309
  LLGL::Surface 324
    LLGL::Canvas 117
    LLGL::Window 353
  LLGL::Timer 334
LLGL::Offset2D 213
LLGL::Offset3D 214
LLGL::OpenGLDependentStateDescriptor 215
LLGL::PipelineLayoutDescriptor 217
LLGL::ProfileOpenGLDescriptor 218
LLGL::QueryHeapDescriptor 220
LLGL::QueryPipelineStatistics 221
LLGL::RasterizerDescriptor 223
LLGL::RenderContextDescriptor 233
LLGL::RendererID 234
LLGL::RendererInfo 236
LLGL::RenderingCapabilities 237
LLGL::RenderingDebugger 239
LLGL::RenderingFeatures 241
LLGL::RenderingLimits 246
LLGL::RenderingProfiler 250
LLGL::RenderPassDescriptor 253
LLGL::RenderSystemConfiguration 274
LLGL::RenderSystemDescriptor 275
LLGL::RenderTargetDescriptor 280
LLGL::ResourceHeapDescriptor 284
LLGL::ShaderReflectionDescriptor::ResourceView 286
LLGL::ResourceViewDescriptor 288
LLGL::SamplerDescriptor 290
LLGL::Scissor 293
LLGL::ShaderCompileFlags 296

LLGL::ShaderDescriptor 297
LLGL::ShaderDisassembleFlags 300
LLGL::ShaderProgramDescriptor 306
LLGL::ShaderReflectionDescriptor 308
LLGL::SrcImageDescriptor 313
LLGL::StageFlags 315
LLGL::StencilDescriptor 316
LLGL::StencilFaceDescriptor 317
LLGL::BufferDescriptor::StorageBuffer 319
LLGL::ShaderDescriptor::StreamOutput 320
LLGL::StreamOutputAttribute 321
LLGL::StreamOutputFormat 323
LLGL::TextureDescriptor 329
LLGL::TextureFlags 331
LLGL::TextureRegion 333
LLGL::UniformDescriptor 336
LLGL::UninitializeTag 337
LLGL::VertexAttribute 338
LLGL::BufferDescriptor::VertexBuffer 341
LLGL::VertexFormat 341
LLGL::VideoAdapterDescriptor 344
LLGL::VideoModeDescriptor 345
LLGL::VideoOutputDescriptor 347
LLGL::Viewport 348
LLGL::VsyncDescriptor 351
LLGL::VulkanRendererConfiguration 352
LLGL::WindowBehavior 360
LLGL::WindowDescriptor 361

Chapter 6

Class Index

6.1 Class List

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

LLGL::ApplicationDescriptor
Application descriptor structure 99
LLGL::AttachmentClear
Attachment clear command structure 100
LLGL::AttachmentDescriptor
Render target attachment descriptor structure 102
LLGL::AttachmentFormatDescriptor
Render target attachment descriptor structure 104
LLGL::BindingDescriptor
Layout structure for a single binding point of the pipeline layout descriptor 106
LLGL::BlendDescriptor
Blending state descriptor structure 108
LLGL::BlendTargetDescriptor
Blend target state descriptor structure 110
LLGL::Buffer
Hardware buffer interface 112
LLGL::BufferArray
Hardware buffer container interface 113
LLGL::BufferDescriptor
Hardware buffer descriptor structure 114
LLGL::BufferFlags
Buffer creation flags enumeration 116
LLGL::Canvas
Canvas interface for mobile platforms 117
LLGL::CanvasDescriptor
Canvas descriptor structure 119
LLGL::ClearFlags
Command buffer clear flags 120
LLGL::ClearValue
Clear value structure for color, depth, and stencil clear operations 121
LLGL::Color< T, N >
Base color class with N components 122
LLGL::Color< T, 3u >
RGB color class with components: r, g, and b 125
LLGL::Color< T, 4u >
RGBA color class with components: r, g, b, and a 130

LLGL::CommandBuffer
Command buffer interface 135
LLGL::CommandBufferDescriptor
Command buffer descriptor structure 152
LLGL::CommandBufferExt
Extended command buffer interface with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers) 153
LLGL::CommandBufferFlags
Command buffer creation flags 156
LLGL::CommandQueue
Command queue interface 156
LLGL::ComputePipeline
Compute pipeline interface 159
LLGL::ComputePipelineDescriptor
Compute pipeline descriptor structure 160
LLGL::DepthBiasDescriptor
Depth bias descriptor structure to control fragment depth values 161
LLGL::DepthDescriptor
Depth state descriptor structure 162
LLGL::Display
Display interface to query the attributes of all connected displays/monitors 163
LLGL::DisplayModeDescriptor
Display mode descriptor structure 167
LLGL::DstImageDescriptor
Descriptor structure for an image that is used as destination for writing the image data 168
LLGL::Canvas::EventListener
Interface for all canvas event listeners 170
LLGL::Window::EventListener
Interface for all window event listeners 171
LLGL::Extent2D
2-Dimensional extent structure 174
LLGL::Extent3D
3-Dimensional extent structure 175
LLGL::Fence
Fence interface for CPU/GPU synchronization 177
LLGL::FrameProfile
Profile of a rendered frame 177
LLGL::GraphicsPipeline
Graphics pipeline interface 186
LLGL::GraphicsPipelineDescriptor
Graphics pipeline descriptor structure 186
LLGL::Image
Utility class to manage the storage and attributes of an image 189
LLGL::ImageInitialization
Structure of image initialization for textures without initial image data 199
LLGL::BufferDescriptor::IndexBuffer
Index buffer specific descriptor structure 200
LLGL::IndexFormat
Index buffer format class 200
LLGL::Input
Default window event listener to receive user input 202
LLGL::RenderingDebugger::Message
Rendering debugger message class 206
LLGL::MultiSamplingDescriptor
Multi-sampling descriptor structure 208
LLGL::NativeContextHandle
IOS native context handle structure 210

LLGL::NativeHandle
IOS native handle structure 211
LLGL::NonCopyable
Base class for all interfaces in LLGL 212
LLGL::Offset2D
2-Dimensional offset structure 213
LLGL::Offset3D
3-Dimensional offset structure 214
LLGL::OpenGLDependentStateDescriptor
Graphics API dependent state descriptor for the OpenGL renderer 215
LLGL::PipelineLayout
Pipeline layout interface 216
LLGL::PipelineLayoutDescriptor
Pipeline layout descritpor structure 217
LLGL::ProfileOpenGLDescriptor
OpenGL profile descriptor structure 218
LLGL::QueryHeap
Query heap interface that holds a certain number of queries that are all of the same type . . . 219
LLGL::QueryHeapDescriptor
Query heap descriptor structure 220
LLGL::QueryPipelineStatistics
Query data structure for pipeline statistics 221
LLGL::RasterizerDescriptor
Rasterizer state descriptor structure 223
LLGL::RenderContext
Render context interface 226
LLGL::RenderContextDescriptor
Render context descriptor structure 233
LLGL::RendererID
Renderer identification number enumeration 234
LLGL::RendererInfo
Renderer basic information structure 236
LLGL::RenderingCapabilities
Structure with all attributes describing the rendering capabilities of the render system 237
LLGL::RenderingDebugger
Rendering debugger interface 239
LLGL::RenderingFeatures
Contains the attributes for all supported rendering features 241
LLGL::RenderingLimits
Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc 246
LLGL::RenderingProfiler
Rendering profiler model class 250
LLGL::RenderPass
Render pass interface 252
LLGL::RenderPassDescriptor
Render pass descriptor structure 253
LLGL::RenderSystem
Render system interface 254
LLGL::RenderSystemChild
Base class for all interfaces whoes instances are owned by the RenderSystem 273
LLGL::RenderSystemConfiguration
Render system configuration structure 274
LLGL::RenderSystemDescriptor
Render system descriptor structure 275
LLGL::RenderTarget
Render target interface 277
LLGL::RenderTargetDescriptor
Render target descriptor structure 280

LLGL::Resource
Base class for all hardware resource interfaces 283
LLGL::ResourceHeap
Resource heap interface 284
LLGL::ResourceHeapDescriptor
Resource heap descriptor structure 284
LLGL::ShaderReflectionDescriptor::ResourceView
Shader reflection resource view structure 286
LLGL::ResourceViewDescriptor
Resource view descriptor structure 288
LLGL::Sampler
Sampler interface 289
LLGL::SamplerDescriptor
Texture sampler descriptor structure 290
LLGL::Scissor
Scissor dimensions 293
LLGL::Shader
Shader interface 294
LLGL::ShaderCompileFlags
Shader compilation flags enumeration 296
LLGL::ShaderDescriptor
Shader source and binary code descriptor structure 297
LLGL::ShaderDisassembleFlags
Shader disassemble flags enumeration 300
LLGL::ShaderProgram
Shader program interface 301
LLGL::ShaderProgramDescriptor
Descriptor structure for shader programs 306
LLGL::ShaderReflectionDescriptor
Shader reflection descriptor structure 308
LLGL::ShaderUniform
Shader uniform setter interface 309
LLGL::SrcImageDescriptor
Descriptor structure for an image that is used as source for reading the image data 313
LLGL::StageFlags
Shader stage flags enumeration 315
LLGL::StencilDescriptor
Stencil state descriptor structure 316
LLGL::StencilFaceDescriptor
Stencil face descriptor structure 317
LLGL::BufferDescriptor::StorageBuffer
Storage buffer specific descriptor structure 319
LLGL::ShaderDescriptor::StreamOutput
Additional descriptor for stream outputs 320
LLGL::StreamOutputAttribute
Stream-output attribute structure 321
LLGL::StreamOutputFormat
Stream-output format descriptor structure 323
LLGL::Surface
The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on movile platforms) 324
LLGL::Texture
Texture interface 327
LLGL::TextureDescriptor
Texture descriptor structure 329
LLGL::TextureFlags
Texture creation flags enumeration 331

LLGL::TextureRegion
Texture region structure 333
LLGL::Timer
Interface for a Timer class 334
LLGL::UniformDescriptor
Shader uniform descriptor structure 336
LLGL::UninitializeTag
Common uninitialize tag 337
LLGL::VertexAttribute
Vertex attribute structure 338
LLGL::BufferDescriptor::VertexBuffer
Vertex buffer specific descriptor structure 341
LLGL::VertexFormat
Vertex format structure 341
LLGL::VideoAdapterDescriptor
Video adapter descriptor structure 344
LLGL::VideoModeDescriptor
Video mode descriptor structure 345
LLGL::VideoOutputDescriptor
Video output structure 347
LLGL::Viewport
Viewport dimensions 348
LLGL::VsyncDescriptor
Vertical-synchronization (Vsync) descriptor structure 351
LLGL::VulkanRendererConfiguration
Structure for a Vulkan renderer specific configuration 352
LLGL::Window
Window interface for desktop platforms 353
LLGL::WindowBehavior
Window behavior structure 360
LLGL::WindowDescriptor
Window descriptor structure 361

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Buffer.h 365
BufferArray.h 365
BufferFlags.h 366
Canvas.h 367
CanvasFlags.h 367
Color.h 367
ColorRGB.h 369
ColorRGBA.h 369
CommandBuffer.h 370
CommandBufferExt.h 370
CommandBufferFlags.h 370
CommandQueue.h 371
ComputePipeline.h 372
ComputePipelineFlags.h 372
Constants.h 372
Display.h 372
DisplayFlags.h 373
Export.h 373
Fence.h 374
Format.h 374
ForwardDecls.h 376
GraphicsPipeline.h 376
GraphicsPipelineFlags.h 376
Image.h 378
ImageFlags.h 379
IndexFormat.h 380
Input.h 380
IOSNativeHandle.h 381
Key.h 381
LinuxNativeHandle.h 382
LLGL.h 383
Log.h 383
MacOSNativeHandle.h 384
NativeHandle.h 384
NonCopyable.h 384

PipelineLayout.h . . . 385
PipelineLayoutFlags.h . . . 385
Platform.h . . . 385
QueryHeap.h . . . 385
QueryHeapFlags.h . . . 386
RenderContext.h . . . 386
RenderContextFlags.h . . . 387
RenderingDebugger.h . . . 388
RenderingProfiler.h . . . 388
RenderPass.h . . . 389
RenderPassFlags.h . . . 389
RenderSystem.h . . . 390
RenderSystemChild.h . . . 390
RenderSystemFlags.h . . . 391
RenderTarget.h . . . 392
RenderTargetFlags.h . . . 393
Resource.h . . . 393
ResourceFlags.h . . . 394
ResourceHeap.h . . . 394
ResourceHeapFlags.h . . . 394
Sampler.h . . . 395
SamplerFlags.h . . . 395
Shader.h . . . 396
ShaderFlags.h . . . 396
ShaderProgram.h . . . 397
ShaderProgramFlags.h . . . 397
ShaderUniform.h . . . 398
ShaderUniformFlags.h . . . 398
StreamOutputAttribute.h . . . 399
StreamOutputFormat.h . . . 400
Strings.h . . . 400
Surface.h . . . 401
Tags.h . . . 401
Texture.h . . . 401
TextureFlags.h . . . 402
Timer.h . . . 403
Types.h . . . 403
Utility.h . . . 404
Version.h . . . 406
VertexAttribute.h . . . 406
VertexFormat.h . . . 407
VideoAdapter.h . . . 407
Win32NativeHandle.h . . . 408
Window.h . . . 408
WindowFlags.h . . . 409

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.

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, Storage↩ BufferType::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, Storage↩ BufferType::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 std::uint32_t LLGL::FormatBitSize (const Format format)

  Returns the bit size of the specified hardware format.
- LLGL_EXPORT bool LLGL::SplitFormat (const Format format, DataType &dataType, std::uint32_t &components)

  Splits the specified hardware format into a data type and the number of components.
- 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.).

8.5.1 Detailed Description

8.5.2 Function Documentation

8.5.2.1 LLGL_EXPORT std::uint32_t LLGL::FormatBitSize (const Format *format*)

Returns the bit size of the specified hardware format.

Returns

Number of bits for one vector of the specified hardware format.

Remarks

This function does not return the size in bytes because some compressed block formats require less than one byte for a single color vector.

See also

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::BC1↩RGBA, 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

8.5.2.9 LLGL_EXPORT bool LLGL::SplitFormat (const Format *format,* DataType & *dataType,* std::uint32_t & *components*)

Splits the specified hardware format into a data type and the number of components.

See also

DataType
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, Data↩Type::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::FindSuitableImageFormat (const Format format, ImageFormat &imageFormat, DataType &dataType)

  Finds a suitable image format for the specified texture hardware format.
- 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 bool LLGL::FindSuitableImageFormat (const Format *format*, ImageFormat & *imageFormat*, DataType & *dataType*)

Finds a suitable image format for the specified texture hardware format.

Parameters

`in`	*textureFormat*	Specifies the input texture format.
`out`	*imageFormat*	Specifies the output image format.
`out`	*dataType*	Specifies the output image data type.

Returns

True if a suitable image format has been found. Otherwise, the output parameter 'imageFormat' and 'dataType' have not been modified.

Remarks

Texture formats that cannot be converted to an image format are all 16-bit floating-point types, and Format↩ ::Undefined.

8.7.2.4 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

8.7.2.5 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 }
);
```

8.7.2.6 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

8.7.2.7 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

8.7.2.8 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

8.7.2.9 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 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.

8.8.1 Detailed Description

8.8.2 Function Documentation

8.8.2.1 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".

8.8.2.2 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".

8.8.2.3 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".

8.8.2.4 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".

8.8.2.5 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 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.

8.9.1 Detailed Description

8.9.2 Function Documentation

8.9.2.1 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.

8.9.2.2 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.

8.9.2.3 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

8.9.2.4 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.

8.9.2.5 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 })).

8.9.2.6 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)

8.9.2.7 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

8.9.2.8 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

8.10 Global utility functions, especially to fill descriptor structures.

Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (Format format, std::uint32_t width, long flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

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

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t size, const IndexFormat &indexFormat, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t size, long flags=BufferFlags::↩DynamicUsage)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  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 ShaderReflectionDescriptor &reflectionDesc)

  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 &render←TargetDesc)

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

8.10.1 Detailed Description

8.10.2 Function Documentation

8.10.2.1 LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t *size,* long *flags =* BufferFlags::DynamicUsage)

Returns a BufferDescriptor structure for a constant buffer.

See also

RenderSystem::CreateBuffer

8.10.2.2 LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t *size,* const IndexFormat & *indexFormat,* long *flags =* 0)

Returns a BufferDescriptor structure for an index buffer.

See also

RenderSystem::CreateBuffer

8.10.2.3 LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const ShaderReflectionDescriptor & *reflectionDesc*)

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.

8.10.2.4 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: • Each type of each binding point (i.e. BindingDescriptor::type) is specified by one of the following identifiers: – `cbuffer` for constant buffers (i.e. ResourceType::ConstantBuffer). – `sbuffer` for storage buffers (i.e. ResourceType::StorageBuffer). – `texture` for textures (i.e. ResourceType::Texture). – `sampler` for sampler states (i.e. ResourceType::Sampler). • The slot of each binding point (i.e. BindingDescriptor::slot) is specified as an integral number within brackets (e.g. `"texture(1)"`). • 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])"`). • Optionally, multiple slots can be specified within the brackets if separated by commas (e.g. `"texture(1[2],3)"`). • Each binding point is separated by a comma, the last comma begin optional (e.g. `"texture(1),sampler(2),"` or `"texture(1),sampler(2)"`). • The stage flags (i.e. BindingDescriptor::stageFlags) can be specified after the each binding point with a preceding colon using the following identifiers: – `vert` for the vertex shader stage (i.e. StageFlags::VertexStage). – `tesc` for the tessellation-control shader stage (i.e. StageFlags::TessControlStage). – `tese` for the tessellation-evaluation shader stage (i.e. StageFlags::TessEvaluationStage). – `geom` for the geometry shader stage (i.e. StageFlags::GeometryStage). – `frag` for the fragment shader stage (i.e. StageFlags::FragmentStage). – `comp` for the compute shader stage (i.e. StageFlags::ComputeStage). • If no stage flag is specified, all shader stages will be used. • 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 {
      LLGL::ResourceType::ConstantBuffer,
      LLGL::StageFlags::FragmentStage |
      LLGL::StageFlags::VertexStage, 0 },
        LLGL::BindingDescriptor {
      LLGL::ResourceType::Texture,
      LLGL::StageFlags::FragmentStage,                              1 },
        LLGL::BindingDescriptor {
      LLGL::ResourceType::Texture,
      LLGL::StageFlags::FragmentStage,                                 2 },
        LLGL::BindingDescriptor {
      LLGL::ResourceType::Sampler,
      LLGL::StageFlags::FragmentStage,                                 3 },
    };
}
auto myLayout = myRenderer->CreatePipelineLayout(myLayoutDescStd);

// Abbreviated way of declaring a pipeline layout using the utility function:
auto myLayoutDescUtil = LLGL::PipelineLayoutDesc(
    "cbuffer(0):frag:vert,"
    "texture(1,2):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.

8.10.2.5 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.

8.10.2.6 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

8.10.2.7 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

8.10.2.8 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

8.10.2.9 LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t *size*, const StorageBufferType *storage↩ Type*, std::uint32_t *stride*, long *flags* = BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

Returns a BufferDescriptor structure for a storage buffer.

See also

RenderSystem::CreateBuffer

8.10.2.10 LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (Format *format*, std::uint32_t *width*, std::uint32_t *arrayLayers*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.11 LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (Format *format*, std::uint32_t *width*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.12 LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.13 LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.14 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, std::uint32_t *samples*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.15 LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *samples*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.16 LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *depth*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.17 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *arrayLayers*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.18 LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (Format *format*, std::uint32_t *width*, std::uint32_t *height*, long *flags* = TextureFlags::Default)

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

See also

RenderSystem::CreateTexture

8.10.2.19 LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (uint64_t *size*, const VertexFormat & *vertexFormat*, long *flags* = 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 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.
- struct BufferFlags

  Buffer creation flags enumeration.

- 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.
- struct CommandBufferDescriptor

    Command buffer descriptor structure.
- class CommandBufferExt

    Extended command buffer interface with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).
- struct CommandBufferFlags

    Command buffer creation flags.
- class CommandQueue

    Command queue interface.
- class ComputePipeline

    Compute pipeline interface.
- struct ComputePipelineDescriptor

    Compute pipeline descriptor structure.
- struct DepthBiasDescriptor

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

    Depth state descriptor structure.
- class Display

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

    Display mode descriptor structure.
- 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 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.
- struct ImageInitialization

  Structure of image initialization for textures without initial image data.
- class IndexFormat

  Index buffer format class.
- class Input

  Default window event listener to receive user input.
- 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 interfaces 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.
- struct ProfileOpenGLDescriptor

  OpenGL profile descriptor 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 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 ShaderDisassembleFlags

  Shader disassemble flags enumeration.
- class ShaderProgram

  Shader program interface.
- struct ShaderProgramDescriptor

  Descriptor structure for shader programs.
- struct ShaderReflectionDescriptor

  Shader reflection descriptor structure.
- class ShaderUniform

  Shader uniform setter interface.
- 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 TextureFlags

  Texture creation flags enumeration.
- struct TextureRegion

  Texture region structure.
- class Timer

  Interface for a Timer class.
- struct UniformDescriptor

  Shader uniform descriptor structure.
- struct UninitializeTag

  Common uninitialize tag.
- struct VertexAttribute

  Vertex 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.
- struct VulkanRendererConfiguration

  Structure for a Vulkan renderer specific configuration.
- 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 BufferType {
  BufferType::Vertex, BufferType::Index, BufferType::Constant, BufferType::Storage,
  BufferType::StreamOutput }

  Hardware buffer type enumeration.
- enum StorageBufferType {
  StorageBufferType::Undefined, StorageBufferType::Buffer, StorageBufferType::StructuredBuffer, Storage↩
  BufferType::ByteAddressBuffer,
  StorageBufferType::RWBuffer, StorageBufferType::RWStructuredBuffer, StorageBufferType::RWByte↩
  AddressBuffer, StorageBufferType::AppendStructuredBuffer,
  StorageBufferType::ConsumeStructuredBuffer }

  Storage buffer type enumeration.
- enum RenderConditionMode {
  RenderConditionMode::Wait, RenderConditionMode::NoWait, RenderConditionMode::ByRegionWait,
  RenderConditionMode::ByRegionNoWait,
  RenderConditionMode::WaitInverted, RenderConditionMode::NoWaitInverted, RenderConditionMode::By↩
  RegionWaitInverted, RenderConditionMode::ByRegionNoWaitInverted }

  Render condition mode enumeration.
- enum Format {

Format::Undefined, 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::RG8UNorm, Format::RG8SNorm, Format::RG8UInt, Format::RG8SInt, Format::RG16UNorm, Format::RG16SNorm, Format::RG16UInt, Format::RG16SInt, Format::RG16Float, Format::RG32UInt, Format::RG32SInt, Format::RG32Float, Format::RGB8UNorm, Format::RGB8SNorm, Format::RGB8UInt, Format::RGB8SInt, Format::RGB16UNorm, Format::RGB16SNorm, Format::RGB16UInt, Format::RGB16SInt, Format::RGB16Float, Format::RGB32UInt, Format::RGB32SInt, Format::RGB32Float, Format::RGBA8UNorm, Format::RGBA8SNorm, Format::RGBA8UInt, Format::RGBA8SInt, Format::RGBA16UNorm, Format::RGBA16SNorm, Format::RGBA16UInt, Format::RGBA16SInt, Format::RGBA16Float, Format::RGBA32UInt, Format::RGBA32SInt, Format::RGBA32Float, Format::R64Float, Format::RG64Float, Format::RGB64Float, Format::RGBA64Float, Format::BGRA8UNorm, Format::BGRA8SNorm, Format::BGRA8UInt, Format::BGRA8SInt, Format::BGRA8sRGB, Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, Format::D32FloatS8X24UInt, Format::BC1RGB, Format::BC1RGBA, Format::BC2RGBA, Format::BC3RGBA }

Hardware vector and pixel format enumeration.

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

  Renderer data types enumeration.

- enum PrimitiveType { PrimitiveType::Points, PrimitiveType::Lines, PrimitiveType::Triangles }

  Primitive type enumeration.

- enum PrimitiveTopology {
PrimitiveTopology::PointList, PrimitiveTopology::LineList, PrimitiveTopology::LineStrip, PrimitiveTopology::↩ LineLoop,
PrimitiveTopology::LineListAdjacency, PrimitiveTopology::LineStripAdjacency, PrimitiveTopology::Triangle↩ List, PrimitiveTopology::TriangleStrip,
PrimitiveTopology::TriangleFan, PrimitiveTopology::TriangleListAdjacency, PrimitiveTopology::TriangleStrip↩ Adjacency, PrimitiveTopology::Patches1,
PrimitiveTopology::Patches2, PrimitiveTopology::Patches3, PrimitiveTopology::Patches4, Primitive↩ Topology::Patches5,
PrimitiveTopology::Patches6, PrimitiveTopology::Patches7, PrimitiveTopology::Patches8, Primitive↩ Topology::Patches9,
PrimitiveTopology::Patches10, PrimitiveTopology::Patches11, PrimitiveTopology::Patches12, Primitive↩ Topology::Patches13,
PrimitiveTopology::Patches14, PrimitiveTopology::Patches15, PrimitiveTopology::Patches16, Primitive↩ Topology::Patches17,
PrimitiveTopology::Patches18, PrimitiveTopology::Patches19, PrimitiveTopology::Patches20, Primitive↩ Topology::Patches21,
PrimitiveTopology::Patches22, PrimitiveTopology::Patches23, PrimitiveTopology::Patches24, Primitive↩ Topology::Patches25,
PrimitiveTopology::Patches26, PrimitiveTopology::Patches27, PrimitiveTopology::Patches28, Primitive↩ Topology::Patches29,
PrimitiveTopology::Patches30, PrimitiveTopology::Patches31, PrimitiveTopology::Patches32 }

  Primitive topology enumeration.

- enum CompareOp {
CompareOp::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 ImageFormat {
  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 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 }

  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), Shading↩Language::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), Shading↩Language::Metal_1_1 = (0x40000 | 110), ShadingLanguage::Metal_1_2 = (0x40000 | 120),
  ShadingLanguage::SPIRV = (0x50000), ShadingLanguage::SPIRV_100 = (0x50000 | 100), Shading↩Language::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, CPU↩Access::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::VertexBuffer, ResourceType::IndexBuffer, ResourceType::↩ConstantBuffer,
  ResourceType::StorageBuffer, ResourceType::StreamOutputBuffer, ResourceType::Texture, Resource↩Type::Sampler }

  Hardware resource type enumeration.

- enum SamplerAddressMode {
  SamplerAddressMode::Repeat, SamplerAddressMode::Mirror, SamplerAddressMode::Clamp, Sampler↩AddressMode::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, ShaderSource↩Type::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::Float3x3, UniformType::Float4x4,
  UniformType::Float2x3, UniformType::Float2x4, UniformType::Float3x2, UniformType::Float3x4,
  UniformType::Float4x2, UniformType::Float4x3, UniformType::Double2x2, UniformType::Double3x3,
  UniformType::Double4x4, UniformType::Double2x3, UniformType::Double2x4, UniformType::Double3x2,
  UniformType::Double3x4, UniformType::Double4x2, UniformType::Double4x3, UniformType::Sampler,
  UniformType::Image, UniformType::AtomicCounter }

  Shader uniform type enumeration.
- enum TextureType {
  TextureType::Texture1D, TextureType::Texture2D, TextureType::Texture3D, TextureType::TextureCube,
  TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, TextureType↩
  ::Texture2DMS,
  TextureType::Texture2DMSArray }

  Texture type enumeration.

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 std::uint32_t FormatBitSize (const Format format)

  Returns the bit size of the specified hardware format.
- LLGL_EXPORT bool SplitFormat (const Format format, DataType &dataType, std::uint32_t &components)

  Splits the specified hardware format into a data type and the number of components.
- 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 FindSuitableImageFormat (const Format format, ImageFormat &imageFormat, Data↩ Type &dataType)

  Finds a suitable image format for the specified texture hardware format.
- 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 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 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 flags=Texture↩ Flags::Default)

  Returns a TextureDescriptor structure with the TextureType::Texture1D type.
- LLGL_EXPORT TextureDescriptor Texture2DDesc (Format format, std::uint32_t width, std::uint32_t height, long flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  Returns a TextureDescriptor structure with the TextureType::Texture3D type.
- LLGL_EXPORT TextureDescriptor TextureCubeDesc (Format format, std::uint32_t width, std::uint32_t height, long flags=TextureFlags::Default)

  Returns a TextureDescriptor structure with the TextureType::TextureCube type.
- LLGL_EXPORT TextureDescriptor Texture1DArrayDesc (Format format, std::uint32_t width, std::uint32_↩ t arrayLayers, long flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

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

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor IndexBufferDesc (uint64_t size, const IndexFormat &indexFormat, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor ConstantBufferDesc (uint64_t size, long flags=BufferFlags::DynamicUsage)

  Returns a BufferDescriptor structure for a constant buffer.

- LLGL_EXPORT BufferDescriptor StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  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 ShaderReflectionDescriptor &reflection↩ Desc)

  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 &renderTarget↩ Desc)

  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 the TextureFlags::ColorAttachmentUsage flag.

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 TextureFlags::DepthStencilAttachmentUsage flag.

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 TextureFlags::DepthStencilAttachmentUsage flag.

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 TextureFlags::DepthStencilAttachmentUsage flag.

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::BufferType [strong]

Hardware buffer type enumeration.

See also

ResourceType

Todo Maybe replace this enum by "ResourceType".

Enumerator

Vertex Vertex buffer type.

Index Index buffer type.

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

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

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

Note

Only supported with: OpenGL, Direct3D 11, Direct3D 12.

9.1.2.7 enum LLGL::ClippingRange [strong]

Clipping depth range enumeration.

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.

9.1.2.8 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.

9.1.2.9 enum LLGL::CPUAccess [strong]

Classifications of CPU access to mapped resources.

See also

RenderSystem::MapBuffer

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.

9.1.2.10 enum LLGL::CullMode [strong]

Polygon culling modes enumeration.

See also

RasterizerDescriptor::cullMode

Enumerator

Disabled No culling.

Front Front face culling.

Back Back face culling.

9.1.2.11 enum **LLGL::DataType** `[strong]`

Renderer data types enumeration.

See also

SrcImageDescriptor::dataType

Enumerator

Int8 8-bit signed integer (char).

UInt8 8-bit unsigned integer (unsigned char).

Int16 16-bit signed integer (short).

UInt16 16-bit unsigned integer (unsigned short).

Int32 32-bit signed integer (int).

UInt32 32-bit unsigned integer (unsiged int).

Float16 16-bit floating-point (half).

Float32 32-bit floating-point (float).

Float64 64-bit real type (double).

9.1.2.12 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.

9.1.2.13 enum **LLGL::Format** `[strong]`

Hardware vector and pixel format enumeration.

Remarks

This enumeration is used for hardware texture formats and vertex attribute formats.

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/b`
`85).aspx`
Metal counterpart `MTLPixelFormat`: `https://developer.apple.com/documentation/metal/mtlpixelf`

Enumerator

Undefined Undefined format.

R8UNorm Color format: red 8-bit normalized unsigned integer component.

R8SNorm Color format: red 8-bit normalized signed integer component.

R8UInt Color format: red 8-bit unsigned integer component.

R8SInt Color format: red 8-bit signed integer component.

R16UNorm Color format: red 16-bit normalized unsigned interger component.

R16SNorm Color format: red 16-bit normalized signed interger component.

R16UInt Color format: red 16-bit unsigned interger component.

R16SInt Color format: red 16-bit signed interger component.

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

R32UInt Color format: red 32-bit unsigned interger component.

R32SInt Color format: red 32-bit signed interger component.

R32Float Color format: red 32-bit floating point component.

RG8UNorm Color format: red, green 8-bit normalized unsigned integer components.

RG8SNorm Color format: red, green 8-bit normalized signed integer components.

RG8UInt Color format: red, green 8-bit unsigned integer components.

RG8SInt Color format: red, green 8-bit signed integer components.

RG16UNorm Color format: red, green 16-bit normalized unsigned interger components.

RG16SNorm Color format: red, green 16-bit normalized signed interger components.

RG16UInt Color format: red, green 16-bit unsigned interger components.

RG16SInt Color format: red, green 16-bit signed interger components.

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

RG32UInt Color format: red, green 32-bit unsigned interger components.

RG32SInt Color format: red, green 32-bit signed interger components.

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

RGB8UNorm Color format: red, green, blue 8-bit normalized unsigned integer components.

Note

Only supported with: OpenGL, Vulkan.

RGB8SNorm Color format: red, green, blue 8-bit normalized signed integer components.

Note

Only supported with: OpenGL, Vulkan.

RGB8UInt Color format: red, green, blue 8-bit unsigned integer components.

Note

Only supported with: OpenGL, Vulkan.

RGB8SInt Color format: red, green, blue 8-bit signed integer components.

Note

Only supported with: OpenGL, Vulkan.

RGB16UNorm Color format: red, green, blue 16-bit normalized unsigned interger components.

Note

Only supported with: OpenGL, Vulkan.

RGB16SNorm Color format: red, green, blue 16-bit normalized signed interger components.

Note

Only supported with: OpenGL, Vulkan.

RGB16UInt Color format: red, green, blue 16-bit unsigned interger components.

Note

Only supported with: OpenGL, Vulkan.

RGB16SInt Color format: red, green, blue 16-bit signed interger components.

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

RGB32UInt Color format: red, green, blue 32-bit unsigned interger components.

RGB32SInt Color format: red, green, blue 32-bit signed interger components.

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

RGBA8UNorm Color format: red, green, blue, alpha 8-bit normalized unsigned integer components.

RGBA8SNorm Color format: red, green, blue, alpha 8-bit normalized signed integer components.

RGBA8UInt Color format: red, green, blue, alpha 8-bit unsigned integer components.

RGBA8SInt Color format: red, green, blue, alpha 8-bit signed integer components.

RGBA16UNorm Color format: red, green, blue, alpha 16-bit normalized unsigned interger components.

RGBA16SNorm Color format: red, green, blue, alpha 16-bit normalized signed interger components.

RGBA16UInt Color format: red, green, blue, alpha 16-bit unsigned interger components.

RGBA16SInt Color format: red, green, blue, alpha 16-bit signed interger components.

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

RGBA32UInt Color format: red, green, blue, alpha 32-bit unsigned interger components.

RGBA32SInt Color format: red, green, blue, alpha 32-bit signed interger components.

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

R64Float Color format: red 64-bit floating point component.

Note

Only supported with: OpenGL, Vulkan.

RG64Float Color format: red, green 64-bit floating point components.

Note

Only supported with: OpenGL, Vulkan.

RGB64Float Color format: red, green, blue 64-bit floating point components.

Note

Only supported with: OpenGL, Vulkan.

RGBA64Float Color format: red, green, blue, alpha 64-bit floating point components.

Note

Only supported with: OpenGL, Vulkan.

BGRA8UNorm Color format: blue, green, red, alpha 8-bit normalized unsigned integer components.

Note

Only supported with: Vulkan, Direct3D 11, Direct3D 12, Metal.

BGRA8SNorm Color format: blue, green, red, alpha 8-bit normalized signed integer components.

Note

Only supported with: Vulkan.

BGRA8UInt Color format: blue, green, red, alpha 8-bit unsigned integer components.

Note

Only supported with: Vulkan.

BGRA8SInt Color format: blue, green, red, alpha 8-bit signed integer components.

Note

Only supported with: Vulkan.

BGRA8sRGB Color format: blue, green, red, alpha 8-bit normalized unsigned integer components in sRGB non-linear color space.

Note

Only supported with: Vulkan, Direct3D 11, Direct3D 12, Metal.

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.

9.1.2.14 enum LLGL::ImageFormat `[strong]`

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

See also

SrcImageDescriptor::format
ImageFormatSize

Enumerator

R	Single color component: Red.
RG	Two color components: Red, Green.
RGB	Three color components: Red, Green, Blue.
BGR	Three color components: Blue, Green, Red.
RGBA	Four color components: Red, Green, Blue, Alpha.
BGRA	Four color components: Blue, Green, Red, Alpha.
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.

9.1.2.15 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.

9.1.2.16 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).

9.1.2.17 enum **LLGL::OpenGLContextProfile** [strong]

OpenGL context profile enumeration.

Remarks

Can be used to specify a specific OpenGL profile other than the default (i.e. compatibility profile).

Enumerator

CompatibilityProfile OpenGL compatibility profile. This is the default.

CoreProfile OpenGL core profile.

ESProfile OpenGL ES profile.

Todo This is incomplete, do not use!

9.1.2.18 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.

9.1.2.19 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.

9.1.2.20 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.

9.1.2.21 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

9.1.2.22 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.

9.1.2.23 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
BufferType

Enumerator

Undefined Undefined resource type.

VertexBuffer Vertex buffer resource.

See also

Buffer
BufferType::Vertex

IndexBuffer Index buffer resource.

See also

Buffer
BufferType::Index

ConstantBuffer Constant buffer (or uniform buffer) resource.

See also

Buffer
BufferType::Constant

StorageBuffer Storage buffer resource.

See also

Buffer
BufferType::Storage

StreamOutputBuffer Stream-output buffer resource.

See also

Buffer
BufferType::StreamOutput

Texture Texture resource.

See also

Texture
TextureType

Sampler Sampler state resource.

See also

Sampler

9.1.2.24 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).

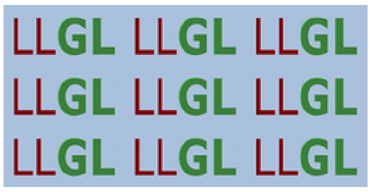


Figure 9.1 SamplerAddressMode::Repeat example

Mirror Flip texture coordinates at each integer junction.

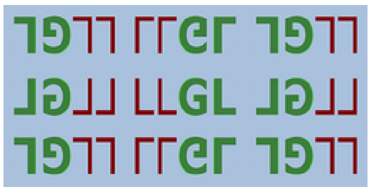


Figure 9.2 SamplerAddressMode::Mirror example

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

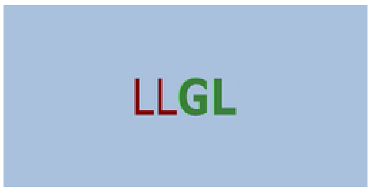


Figure 9.3 SamplerAddressMode::Clamp example

Border Sample border color for texture coordinates that are outside the interval [0, 1].

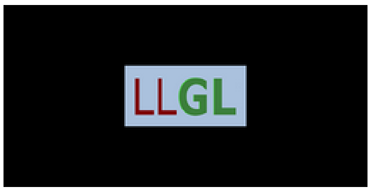


Figure 9.4 SamplerAddressMode::Border example

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

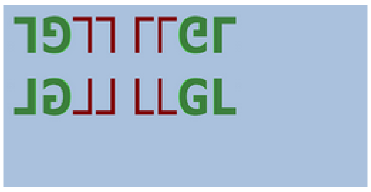


Figure 9.5 SamplerAddressMode::MirrorOnce example

9.1.2.25 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.

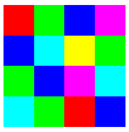


Figure 9.6 SamplerFilter::Nearest example

Linear Interpolate between multiple texture samples.

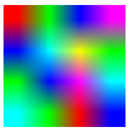


Figure 9.7 SamplerFilter::Linear example

9.1.2.26 enum LLGL::ScreenOrigin `[strong]`

Screen coordinate system origin enumeration.

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.

9.1.2.27 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).

9.1.2.28 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.

9.1.2.29 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.

Note

Not supported yet

Metal_1_0 Metal 1.0 (since iOS 8.0).

Note

Not supported yet

Metal_1_1 Metal 1.1 (since iOS 9.0 and OS X 10.11).

Note

Not supported yet

Metal_1_2 Metal 1.2 (since iOS 10.0 and macOS 10.12).

Note

Not supported yet

SPIRV SPIR-V Shading Language.

SPIRV_100 SPIR-V 1.0.

VersionBitmask Bitmask for the version number of each shading language enumeration entry.

9.1.2.30 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.

9.1.2.31 enum LLGL::StorageBufferType [strong]

Storage buffer type enumeration.

Note

Only supported with: Direct3D 11, Direct3D 12.

Enumerator

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

9.1.2.32 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.

9.1.2.33 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.
Float3x3	float3x3/ mat3 uniform.
Float4x4	float4x4/ mat4 uniform.
Float2x3	float2x3/ mat2x3 uniform.
Float2x4	float2x4/ mat2x4 uniform.
Float3x2	float3x2/ mat3x2 uniform.
Float3x4	float3x4/ mat3x4 uniform.
Float4x2	float4x2/ mat4x2 uniform.
Float4x3	float4x3/ mat4x3 uniform.
Double2x2	double2x2/ dmat2 uniform.
Double3x3	double3x3/ dmat3 uniform.
Double4x4	double4x4/ dmat4 uniform.
Double2x3	double2x3/ dmat2x3 uniform.
Double2x4	double2x4/ dmat2x4 uniform.
Double3x2	double3x2/ dmat3x2 uniform.
Double3x4	double3x4/ dmat3x4 uniform.
Double4x2	double4x2/ dmat4x2 uniform.
Double4x3	double4x3/ dmat4x3 uniform.
Sampler	Sampler uniform (e.g. "sampler2D").
Image	Image uniform (e.g. "image2D").
AtomicCounter	Atomic counter uniform (e.g. "atomic_uint").

9.1.2.34 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).

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 **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

9.1.3.7 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.

9.1.3.8 LLGL_EXPORT bool LLGL::IsPrimitiveTopologyPatches (const PrimitiveTopology *primitiveTopology*)

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

9.1.3.9 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

9.1.3.10 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

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

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

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

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

9.1.3.13 template<> std::uint8_t LLGL::MaxColorValue< std::uint8_t > ( ) [inline]

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

9.1.3.14 LLGL_EXPORT bool LLGL::operator!= (const DisplayModeDescriptor & *lhs*, const DisplayModeDescriptor & *rhs*)

Compares the two specified display mode descriptors on inequality.

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

9.1.3.16 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).

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

Compares the two specified V-sync descriptors on inequality.

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

Compares the two specified video mode descriptors on inequality.

9.1.3.19 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.

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

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

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

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

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 Color< T, N > & *rhs*)

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

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

9.1.3.28 LLGL_EXPORT bool LLGL::operator== (const DisplayModeDescriptor & *lhs*, const DisplayModeDescriptor & *rhs*)

Compares the two specified display mode descriptors on equality.

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

9.1.3.30 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).

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

Compares the two specified V-sync descriptors on equality.

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

Compares the two specified video mode descriptors on equality.

9.1.3.33 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.

9.1.3.34 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 &context↩ Info="")
- 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.

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

9.4 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 <RenderSystemFlags.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

VulkanRendererConfiguration::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:

- RenderSystemFlags.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::Texture2DM↩
SArray). 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↩
::arrayLayer member.

See also

TextureDescriptor::arrayLayer

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).

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 with the flag 'TextureFlags::BindRenderTarget'.

See also

AttachmentDescriptor::type
TextureFlags::BindRenderTarget

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 Render↩ Context::QueryColorFormat function, and the appropriate depth-stencil format can be determined by the RenderContext::QueryDepthStencilFormat 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::QueryColorFormat
RenderContext::QueryDepthStencilFormat

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::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 stageFlags, std::uint32_t slot, std::uint32_t arraySize=1)

  Constructors with all attributes and a default value for a uniform array.

Public Attributes

- ResourceType type = ResourceType::Undefined

  Resource view type for this layout binding. By default ResourceType::Undefined.
- 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.

10.5.1 Detailed Description

Layout structure for a single binding point of the pipeline layout descriptor.

See also

PipelineLayoutDescriptor::bindings

10.5.2 Constructor & Destructor Documentation

10.5.2.1 LLGL::BindingDescriptor::BindingDescriptor ( ) `[default]`

10.5.2.2 LLGL::BindingDescriptor::BindingDescriptor (const **BindingDescriptor** &) `[default]`

10.5.2.3 LLGL::BindingDescriptor::BindingDescriptor (**ResourceType** *type*, long *stageFlags*, std::uint32_t *slot*, std::uint32_t *arraySize* = `1`) `[inline]`

Constructors with all attributes and a default value for a uniform array.

10.5.3 Member Data Documentation

10.5.3.1 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).

10.5.3.2 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.

10.5.3.3 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

10.5.3.4 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

10.6 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.

10.6.1 Detailed Description

Blending state descriptor structure.

See also

GraphicsPipelineDescriptor::blend

10.6.2 Member Data Documentation

10.6.2.1 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).

10.6.2.2 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 Blend↩ Op::InvBlendFactor.

See also

BlendOp::BlendFactor
BlendOp::InvBlendFactor

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

10.6.2.3 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

10.6.2.4 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

10.6.2.5 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

10.7 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).

10.7.1 Detailed Description

Blend target state descriptor structure.

See also

BlendDescriptor::targets

10.7.2 Member Data Documentation

10.7.2.1 BlendArithmetic LLGL::BlendTargetDescriptor::alphaArithmetic = BlendArithmetic::Add

Alpha blending arithmetic. By default BlendArithmetic::Add.

10.7.2.2 bool LLGL::BlendTargetDescriptor::blendEnabled = false

Specifies whether blending is enabled or disabled for the respective color attachment.

10.7.2.3 BlendArithmetic LLGL::BlendTargetDescriptor::colorArithmetic = BlendArithmetic::Add

Color blending arithmetic. By default BlendArithmetic::Add.

10.7.2.4 ColorRGBAb LLGL::BlendTargetDescriptor::colorMask = { true, true, true, true }

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

10.7.2.5 BlendOp LLGL::BlendTargetDescriptor::dstAlpha = BlendOp::InvSrcAlpha

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

10.7.2.6 BlendOp LLGL::BlendTargetDescriptor::dstColor = BlendOp::InvSrcAlpha

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

10.7.2.7 BlendOp LLGL::BlendTargetDescriptor::srcAlpha = BlendOp::SrcAlpha

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

10.7.2.8 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:

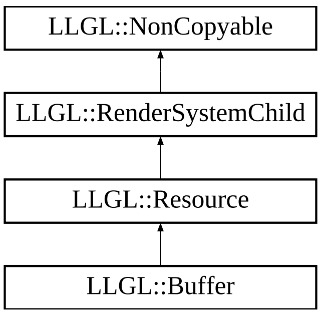
- GraphicsPipelineFlags.h

10.8 LLGL::Buffer Class Reference

Hardware buffer interface.

```
#include <Buffer.h>
```

Inheritance diagram for LLGL::Buffer:



Public Member Functions

- ResourceType QueryResourceType () const override

  Returns the ResourceType for the respective BufferType.
- BufferType GetType () const

  Returns the type of this buffer.

Protected Member Functions

- Buffer (const BufferType type)

10.8.1 Detailed Description

Hardware buffer interface.

See also

RenderSystem::CreateBuffer

10.8.2 Constructor & Destructor Documentation

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

10.8.3 Member Function Documentation

10.8.3.1 BufferType LLGL::Buffer::GetType ( ) const `[inline]`

Returns the type of this buffer.

10.8.3.2 ResourceType LLGL::Buffer::QueryResourceType ( ) const `[override], [virtual]`

Returns the ResourceType for the respective BufferType.

Implements LLGL::Resource.

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

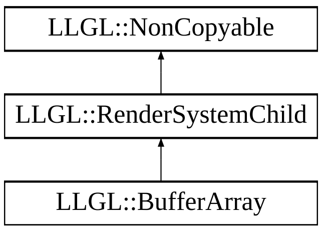
- Buffer.h

10.9 LLGL::BufferArray Class Reference

Hardware buffer container interface.

`#include <BufferArray.h>`

Inheritance diagram for LLGL::BufferArray:



Public Member Functions

- BufferType GetType () const

  Returns the type of buffers this array contains.

Protected Member Functions

- BufferArray (const BufferType type)

10.9.1 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

10.9.2 Constructor & Destructor Documentation

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

10.9.3 Member Function Documentation

10.9.3.1 **BufferType** LLGL::BufferArray::GetType ( ) const `[inline]`

Returns the type of buffers this array contains.

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

- BufferArray.h

10.10 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

- BufferType type = BufferType::Vertex

  Hardware buffer type. By default BufferType::Vertex.
- long flags = 0

  Specifies the buffer creation flags. By default 0.
- std::uint64_t size = 0

  Buffer size (in bytes). This must not be larger than 'RenderingLimits::maxBufferSize'. By default 0.
- VertexBuffer vertexBuffer

  Vertex buffer type descriptor appendix.
- IndexBuffer indexBuffer

  Index buffer type descriptor appendix.
- StorageBuffer storageBuffer

  Storage buffer type descriptor appendix.

10.10.1 Detailed Description

Hardware buffer descriptor structure.

10.10.2 Member Data Documentation

10.10.2.1 long LLGL::BufferDescriptor::flags = 0

Specifies the buffer creation flags. By default 0.

Remarks

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

See also

BufferFlags

10.10.2.2 IndexBuffer LLGL::BufferDescriptor::indexBuffer

Index buffer type descriptor appendix.

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

See also

RenderingLimits::maxBufferSize

10.10.2.4 StorageBuffer LLGL::BufferDescriptor::storageBuffer

Storage buffer type descriptor appendix.

10.10.2.5 BufferType LLGL::BufferDescriptor::type = BufferType::Vertex

Hardware buffer type. By default BufferType::Vertex.

10.10.2.6 VertexBuffer LLGL::BufferDescriptor::vertexBuffer

Vertex buffer type descriptor appendix.

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

- BufferFlags.h

10.11 LLGL::BufferFlags Struct Reference

Buffer creation flags enumeration.

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

Public Types

- enum { MapReadAccess = (1 << 0), MapWriteAccess = (1 << 1), MapReadWriteAccess = (MapRead↩
  Access | MapWriteAccess), DynamicUsage = (1 << 2) }

10.11.1 Detailed Description

Buffer creation flags enumeration.

See also

BufferDescriptor::flags

10.11.2 Member Enumeration Documentation

10.11.2.1 anonymous enum

Enumerator

MapReadAccess Buffer mapping with CPU read access is required.

See also

RenderSystem::MapBuffer

MapWriteAccess Buffer mapping with CPU write access is required.

See also

RenderSystem::MapBuffer

MapReadWriteAccess

DynamicUsage Hint to the renderer that the buffer 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

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

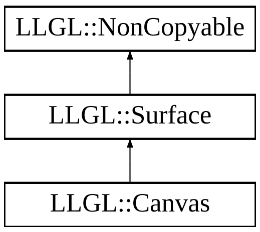
- BufferFlags.h

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.
- bool AdaptForVideoMode (VideoModeDescriptor &videoModeDesc) override

  This default implementation ignores the video mode descriptor completely and always return false.
- void ProcessEvents ()

  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.

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 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

10.12.2.6 void LLGL::Canvas::ProcessEvents ( )

Processes the events for this canvas (i.e. touch input, key presses etc.).

10.12.2.7 void LLGL::Canvas::RemoveEventListener (const EventListener * *eventListener*)

Removes the specified event listener from this canvas.

10.12.2.8 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
ImageInitialization::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:

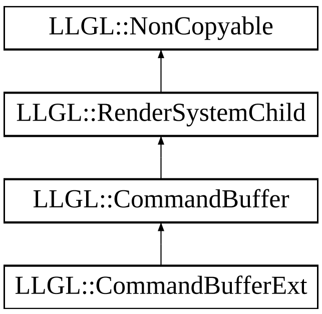
- ColorRGBA.h

10.19 LLGL::CommandBuffer Class Reference

Command buffer interface.

```
#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 UpdateBuffer (Buffer &dstBuffer, std::uint64_t dstOffset, const void ∗data, std::uint16_t data←Size)=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 SetGraphicsAPIDependentState (const void ∗stateDesc, std::size_t stateDescSize)=0

  Sets a few low-level graphics API dependent states.
- 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 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 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 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 Render↩ ConditionMode 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 first↩ Index)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t first↩ Index, std::int32_t vertexOffset)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t first↩ Index, 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 Dispatch (std::uint32_t groupSizeX, std::uint32_t groupSizeY, std::uint32_t groupSizeZ)=0

  Dispachtes a compute command.

Protected Member Functions

- CommandBuffer ()=default

10.19.1 Detailed Description

Command buffer interface.

Remarks

This is the main interface to encode graphics and compute 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. There are only a few exceptions of functions that can be used outside of encoding, which are CommandBuffer::SetClearColor, CommandBuffer::SetClearDepth, and CommandBuffer::SetClearStencil.

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`, except for the following:

- CommandBuffer::SetClearColor
- CommandBuffer::SetClearDepth
- CommandBuffer::SetClearStencil

See also

End
RecordingFlags

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.
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. A simple frame setup could look like this:

```
myCmdQueue->Begin(*myCmdBuffer);
{
    myCmdBuffer->BeginRenderPass(*myRenderTarget);
    {
        myCmdBuffer->SetGraphicsPipeline(*myGfxPipeline);
        myCmdBuffer->SetGraphicsResourceHeap(*myResourceHeap);
        myCmdBuffer->Draw(...);
    }
    myCmdBuffer->EndRenderPass();
}
myCmdQueue->End(*myCmdBuffer);
myRenderContext->Present();
```

The following commands can only appear inside a render pass section:

- Drawing commands (i.e. `Draw`, `DrawInstanced`, `DrawIndexed`, and `DrawIndexed↩ Instanced`).
- Clear attachment commands (i.e. `Clear`, and `ClearAttachments`).

The following commands can only appear outside a render pass section:

- Dispatch compute commands (i.e. `Dispatch`).
- Resource read/write from the RenderSystem (i.e. `WriteBuffer`, `MapBuffer` etc.).

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

It is recommended to call this outside of a render pass. Otherwise, LLGL needs to pause and resume the render pass for the Vulkan backend via a secondary render pass object.

10.19.3.9 virtual void LLGL::CommandBuffer::Dispatch (std::uint32_t *groupSizeX,* std::uint32_t *groupSizeY,* std::uint32_t *groupSizeZ*) `[pure virtual]`

Dispachtes a compute command.

Parameters

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

See also

SetComputePipeline
RenderingLimits::maxComputeShaderWorkGroups

10.19.3.10 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`.

10.19.3.11 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)

10.19.3.12 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.

10.19.3.13 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)

10.19.3.14 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)

10.19.3.15 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`.

10.19.3.16 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)

10.19.3.17 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_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`. 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).

10.19.3.18 virtual void LLGL::CommandBuffer::End ( ) `[pure virtual]`

Ends the encoding (also referred to as "recording") of this command buffer.

See also

Begin
CommandQueue::Submit(CommandBuffer&)

10.19.3.19 virtual void LLGL::CommandBuffer::EndQuery (**QueryHeap** & *queryHeap*, std::uint32_t *query* = `0`) `[pure virtual]`

Ends the specified query.

See also

BeginQuery

10.19.3.20 virtual void LLGL::CommandBuffer::EndRenderCondition ( ) `[pure virtual]`

Ends the current render condition.

See also

BeginRenderCondition

10.19.3.21 virtual void LLGL::CommandBuffer::EndRenderPass ( ) `[pure virtual]`

Ends the current render pass.

See also

BeginRenderPass

10.19.3.22 virtual void LLGL::CommandBuffer::EndStreamOutput ( ) `[pure virtual]`

Ends the current stream-output.

See also

BeginStreamOutput

10.19.3.23 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

10.19.3.24 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

10.19.3.25 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

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

Sets the active compute pipeline state.

Parameters

in	*computePipeline*	Specifies the compupite 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

10.19.3.27 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.

10.19.3.28 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;
myCommandBuffer->SetGraphicsAPIDependentState(&myOpenGLStateDesc, sizeof(myOpenGLStateDesc));
```

Note

Invalid arguments are ignored by this function silently (except for corrupted pointers).

See also

OpenGLDependentStateDescriptor

10.19.3.29 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

10.19.3.30 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.

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

Sets the active index buffer for subsequent drawing operations.

Parameters

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

Remarks

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

See also

RenderSystem::WriteIndexBuffer

10.19.3.32 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

10.19.3.33 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.screenSpaceOriginLowerLeft' is false, the origin of each scissor rectangle is on the upper-left (like for all other render systems). If 'stateOpenGL.screen↩SpaceOriginLowerLeft' is true, the origin of each scissor rectangle is on the lower-left.

See also

SetGraphicsAPIDependentState
RasterizerDescriptor::scissorTestEnabled

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

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

Parameters

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

See also

RenderSystem::MapBuffer
RenderSystem::UnmapBuffer

10.19.3.35 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

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

Sets the specified vertex buffer for subsequent drawing operations.

Parameters

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

See also

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
SetVertexBufferArray

10.19.3.37 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

10.19.3.38 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

10.19.3.39 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 'stateOpenGL.screenSpaceOriginLowerLeft' is false, the origin of each viewport is on the upper-left (like for all other render systems). If 'stateOpenGL.screenSpaceOrigin↩LowerLeft' is true, the origin of each viewport is on the lower-left.

Note

This state is guaranteed to be persistent.

See also

SetGraphicsAPIDependentState
RenderingLimits::maxViewports

10.19.3.40 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. It is recommended to call this outside of a render pass. Otherwise, LLGL needs to pause and resume the render pass for the Vulkan backend via a secondary render pass object.

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:

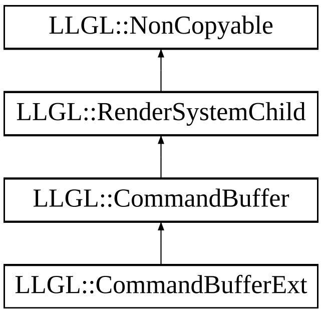
- CommandBufferFlags.h

10.21 LLGL::CommandBufferExt Class Reference

Extended command buffer interface with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).

```
#include <CommandBufferExt.h>
```

Inheritance diagram for LLGL::CommandBufferExt:



Public Member Functions

- virtual void SetConstantBuffer (Buffer &buffer, std::uint32_t slot, long stageFlags=StageFlags::AllStages)=0

    Sets the active constant buffer at the specified slot index for subsequent drawing and compute operations.
- virtual void SetStorageBuffer (Buffer &buffer, std::uint32_t slot, long stageFlags=StageFlags::AllStages)=0

    Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.
- virtual void SetTexture (Texture &texture, std::uint32_t slot, long stageFlags=StageFlags::AllStages)=0

    Sets the active texture of the specified slot index for subsequent drawing and compute operations.
- virtual void SetSampler (Sampler &sampler, std::uint32_t slot, long stageFlags=StageFlags::AllStages)=0

    Sets the active sampler of the specified slot index for subsequent drawing and compute operations.
- virtual void ResetResourceSlots (const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_↩ t numSlots, long stageFlags=StageFlags::AllStages)=0

    Resets the binding slots for the specified resources.

Protected Member Functions

- CommandBufferExt ()=default

10.21.1 Detailed Description

Extended command buffer interface with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).

Remarks

This is an extended command interface for the legacy graphics APIs such as OpenGL and Direct3D 11 to dynamically change bounded shader resources.

Note

Only supported with: OpenGL, Direct3D 11.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 LLGL::CommandBufferExt::CommandBufferExt ( ) `[protected]`, `[default]`

10.21.3 Member Function Documentation

10.21.3.1 virtual void LLGL::CommandBufferExt::ResetResourceSlots (const ResourceType *resourceType*, std::uint32_t *firstSlot*, std::uint32_t *numSlots*, 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 bindings slots to reset. If this is zero, the function has no effect.
`in`	*stageFlags*	Specifies which shader stages are affected. This can be a bitwise OR combination of the StageFlags entries. By default StageFlags::AllStages.

See also

StageFlags

10.21.3.2 virtual void LLGL::CommandBufferExt::SetConstantBuffer (Buffer & *buffer*, std::uint32_t *slot*, long *stageFlags* = StageFlags::AllStages) `[pure virtual]`

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

Parameters

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

See also

RenderSystem::WriteBuffer
StageFlags

10.21.3.3 virtual void LLGL::CommandBufferExt::SetSampler (**Sampler** & *sampler*, std::uint32_t *slot*, long *stageFlags* = **StageFlags::AllStages**) [pure virtual]

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

Parameters

in	*sampler*	Specifies the sampler to set.
in	*slot*	Specifies the slot index where to put the sampler.

See also

RenderSystem::CreateSampler

10.21.3.4 virtual void LLGL::CommandBufferExt::SetStorageBuffer (**Buffer** & *buffer*, std::uint32_t *slot*, long *stageFlags* = **StageFlags::AllStages**) [pure virtual]

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

Parameters

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

See also

RenderSystem::MapBuffer
RenderSystem::UnmapBuffer
StageFlags::ReadOnlyResource

10.21.3.5 virtual void LLGL::CommandBufferExt::SetTexture (**Texture** & *texture*, std::uint32_t *slot*, long *stageFlags* = **StageFlags::AllStages**) [pure virtual]

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

Parameters

in	*texture*	Specifies the texture to set.
in	*slot*	Specifies the slot index where to put the texture.

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

- CommandBufferExt.h

10.22 LLGL::CommandBufferFlags Struct Reference

Command buffer creation flags.

```
#include <CommandBufferFlags.h>
```

Public Types

- enum { DeferredSubmit = (1 << 0) }

10.22.1 Detailed Description

Command buffer creation flags.

See also

CommandBufferDescriptor::flags

10.22.2 Member Enumeration Documentation

10.22.2.1 anonymous enum

Enumerator

DeferredSubmit 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.

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

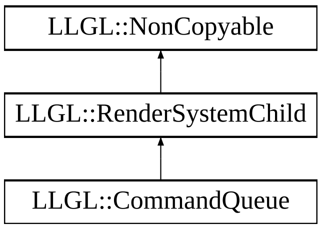
- CommandBufferFlags.h

10.23 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

10.23.1 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.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 LLGL::CommandQueue::CommandQueue ( ) `[protected]`, `[default]`

10.23.3 Member Function Documentation

10.23.3.1 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:
		• std::uint32_t • std::uint64_t • QueryPipelineStatistics If the function return false, the content of this array is

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));
```

10.23.3.2 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 ∗)

10.23.3.3 virtual void LLGL::CommandQueue::Submit (**Fence** & *fence*) [pure virtual]

Submits the specified fence to the command queue for CPU/GPU synchronization.

10.23.3.4 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

10.23.3.5 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:

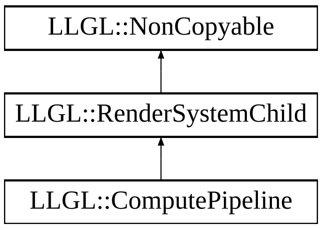
- CommandQueue.h

10.24 LLGL::ComputePipeline Class Reference

Compute pipeline interface.

`#include <ComputePipeline.h>`

Inheritance diagram for LLGL::ComputePipeline:



Additional Inherited Members

10.24.1 Detailed Description

Compute pipeline interface.

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

- ComputePipeline.h

10.25 LLGL::ComputePipelineDescriptor Struct Reference

Compute pipeline descriptor structure.

`#include <ComputePipelineFlags.h>`

Public Member Functions

- ComputePipelineDescriptor ()=default
- ComputePipelineDescriptor (ShaderProgram ∗shaderProgram, PipelineLayout ∗pipelineLayout=nullptr)

  Constructor to initialize the entire descriptor.

Public Attributes

- ShaderProgram ∗ shaderProgram = nullptr

  Pointer to the shader program for the compute pipeline.
- PipelineLayout ∗ pipelineLayout = nullptr

  Pointer to an optional pipeline layout for the graphics pipeline.

10.25.1 Detailed Description

Compute pipeline descriptor structure.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor ( ) `[default]`

10.25.2.2 LLGL::ComputePipelineDescriptor::ComputePipelineDescriptor (**ShaderProgram** ∗ *shaderProgram*, **PipelineLayout** ∗ *pipelineLayout* = `nullptr`) `[inline]`

Constructor to initialize the entire descriptor.

10.25.3 Member Data Documentation

10.25.3.1 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

10.25.3.2 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

10.26 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.

Remarks

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

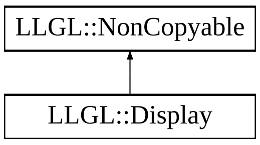
- GraphicsPipelineFlags.h

10.28 LLGL::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 > QuerySupportedDisplayModes () 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 > > QueryList ()

  Queries the list of all connected displays.
- static std::unique_ptr< Display > QueryPrimary ()

  Queries 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 QuerySupportedDisplayModes function, and removes duplicate entries.

Additional Inherited Members

10.28.1 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::QueryList();
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;
}
```

10.28.2 Member Function Documentation

10.28.2.1 static void LLGL::Display::FinalizeDisplayModes (std::vector< DisplayModeDescriptor > & *displayModeDescs*) [static], [protected]

Sorts the specified list of display modes as described in the QuerySupportedDisplayModes function, and removes duplicate entries.

See also

QuerySupportedDisplayModes

10.28.2.2 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.

10.28.2.3 virtual DisplayModeDescriptor LLGL::Display::GetDisplayMode ( ) const [pure virtual]

Returns the current display mode of this display.

See also

SetDisplayMode

10.28.2.4 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

10.28.2.5 static bool LLGL::Display::IsCursorShown ( ) [static]

Returns true if the cursor is currently being shown on any display.

See also

ShowCursor

10.28.2.6 virtual bool LLGL::Display::IsPrimary ( ) const [pure virtual]

Returns true if this is the primary display, as configured by the host system.

10.28.2.7 static std::vector<std::unique_ptr<Display> > LLGL::Display::QueryList ( ) [static]

Queries the list of all connected displays.

10.28.2.8 static std::unique_ptr<Display> LLGL::Display::QueryPrimary ( ) [static]

Queries the primary display.

Returns

Unique pointer to a Display instance that represents the primary display, or null on failure.

10.28.2.9 virtual std::vector<DisplayModeDescriptor> LLGL::Display::QuerySupportedDisplayModes ( ) 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

10.28.2.10 virtual bool LLGL::Display::ResetDisplayMode ( ) [pure virtual]

Resets the display mode to its default value depending on the host system configuration.

See also

SetDisplayMode

10.28.2.11 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

10.28.2.12 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

10.29 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).

10.29.1 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

10.29.2 Member Data Documentation

10.29.2.1 std::uint32_t LLGL::DisplayModeDescriptor::refreshRate = 0

Display refresh rate (in Hz).

10.29.2.2 Extent2D LLGL::DisplayModeDescriptor::resolution

Display resolution (in pixels).

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

- DisplayFlags.h

10.30 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.

10.30.1 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

10.30.2 Constructor & Destructor Documentation

10.30.2.1 LLGL::DstImageDescriptor::DstImageDescriptor ( ) [default]

10.30.2.2 LLGL::DstImageDescriptor::DstImageDescriptor (const DstImageDescriptor &) [default]

10.30.2.3 LLGL::DstImageDescriptor::DstImageDescriptor (ImageFormat *format*, DataType *dataType*, void ∗ *data*, std::size_t *dataSize*) [inline]

Constructor to initialize all attributes.

10.30.3 Member Data Documentation

10.30.3.1 void∗ LLGL::DstImageDescriptor::data = nullptr

Pointer to the read/write image data.

10.30.3.2 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.

10.30.3.3 DataType LLGL::DstImageDescriptor::dataType = DataType::UInt8

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

10.30.3.4 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

10.31 LLGL::Canvas::EventListener Class Reference

Interface for all canvas event listeners.

`#include <Canvas.h>`

Public Member Functions

- virtual ∼EventListener ()=default

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.

Friends

- class Canvas

10.31.1 Detailed Description

Interface for all canvas event listeners.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 virtual LLGL::Canvas::EventListener::∼EventListener ( ) `[virtual]`, `[default]`

10.31.3 Member Function Documentation

10.31.3.1 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

10.31.4 Friends And Related Function Documentation

10.31.4.1 friend class Canvas `[friend]`

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

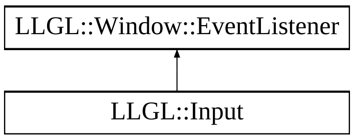
- Canvas.h

10.32 LLGL::Window::EventListener Class Reference

Interface for all window event listeners.

`#include <Window.h>`

Inheritance diagram for LLGL::Window::EventListener:



Public Member Functions

- virtual ∼EventListener ()=default

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 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 OnLoseFocus (Window &sender)

  Send when the window loses the keyboard focus.
- virtual bool OnQuit (Window &sender)

  Send when the window is about to be quit.
- 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

10.32.1 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

10.32.2 Constructor & Destructor Documentation

10.32.2.1 virtual LLGL::Window::EventListener::∼EventListener ( ) `[virtual]`, `[default]`

10.32.3 Member Function Documentation

10.32.3.1 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.

10.32.3.2 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.

10.32.3.3 virtual void LLGL::Window::EventListener::OnGetFocus (Window & *sender*) `[protected]`, `[virtual]`

Send when the window gets the keyboard focus.

10.32.3.4 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.

10.32.3.5 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.

10.32.3.6 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.

10.32.3.7 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.

10.32.3.8 virtual void LLGL::Window::EventListener::OnLoseFocus (**Window** & *sender*) `[protected]`, `[virtual]`

Send when the window loses the keyboard focus.

Reimplemented in LLGL::Input.

10.32.3.9 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.

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

Send when the window is about to be quit.

Returns

True if the sender window can quit. In this case "ProcessEvents" returns false from now on. Otherwise the quit can be prevented. Returns true by default.

See also

Window::ProcessEvents

10.32.3.11 virtual void LLGL::Window::EventListener::OnResize (**Window** & *sender,* const **Extent2D** & *clientAreaSize*) [protected], [virtual]

Send when the window has been resized.

10.32.3.12 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.

10.32.3.13 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.

10.32.4 Friends And Related Function Documentation

10.32.4.1 friend class **Window** [friend]

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

- Window.h

10.33 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.

10.33.1 Detailed Description

2-Dimensional extent structure.

Remarks

Used for unsigned integral 2D extents (for sizes in window-space, screen-space, and texture-space).

10.33.2 Constructor & Destructor Documentation

10.33.2.1 LLGL::Extent2D::Extent2D ( ) `[default]`

10.33.2.2 LLGL::Extent2D::Extent2D (const Extent2D &) `[default]`

10.33.2.3 LLGL::Extent2D::Extent2D (std::uint32_t *width*, std::uint32_t *height*) `[inline]`

10.33.3 Member Data Documentation

10.33.3.1 std::uint32_t LLGL::Extent2D::height = 0

Extent Y axis, i.e. height.

10.33.3.2 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

10.34 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.

10.34.1 Detailed Description

3-Dimensional extent structure.

Remarks

Used for unsigned integral 3D extents (for sizes in texture-space).

10.34.2 Constructor & Destructor Documentation

10.34.2.1 LLGL::Extent3D::Extent3D ( ) [default]

10.34.2.2 LLGL::Extent3D::Extent3D (const Extent3D &) [default]

10.34.2.3 LLGL::Extent3D::Extent3D (std::uint32_t *width*, std::uint32_t *height*, std::uint32_t *depth*) [inline]

10.34.3 Member Data Documentation

10.34.3.1 std::uint32_t LLGL::Extent3D::depth = 0

Extent Z axis, i.e. depth.

10.34.3.2 std::uint32_t LLGL::Extent3D::height = 0

Extent Y axis, i.e. height.

10.34.3.3 std::uint32_t LLGL::Extent3D::width = 0

Extent X axis, i.e. width.

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

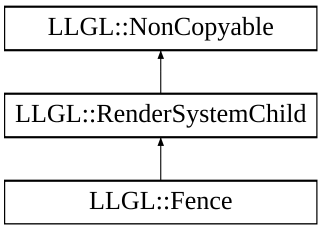
- Types.h

10.35 LLGL::Fence Class Reference

Fence interface for CPU/GPU synchronization.

```
#include <Fence.h>
```

Inheritance diagram for LLGL::Fence:



Additional Inherited Members

10.35.1 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

10.36 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 (i.e. without a ResourceHeap).
    std::uint32_t storageBufferBindings
      Counter for all individual storage buffer bindings (i.e. without a ResourceHeap).
    std::uint32_t textureBindings
      Counter for all individual texture bindings (i.e. without a ResourceHeap).
    std::uint32_t samplerBindings
      Counter for all individual sampler bindings (i.e. without a ResourceHeap).
    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 [31]
All proflile values as linear array.
};

10.36.1 Detailed Description

Profile of a rendered frame.

See also

RenderingProfiler::NextFrame

10.36.2 Constructor & Destructor Documentation

10.36.2.1 LLGL::FrameProfile::FrameProfile ( ) [inline]

Default constructor that initializes all counter values to zero.

10.36.3 Member Function Documentation

10.36.3.1 void LLGL::FrameProfile::Accumulate (const FrameProfile & *rhs*) [inline]

Accumulates the specified profile with this profile.

10.36.3.2 void LLGL::FrameProfile::Clear ( ) [inline]

Clears all counter values.

10.36.4 Member Data Documentation

10.36.4.1 union { ... }

10.36.4.2 std::uint32_t LLGL::FrameProfile::attachmentClears

Counter for all framebuffer attachment clear operations.

See also

CommandBuffer::Clear
CommandBuffer::ClearAttachments

10.36.4.3 std::uint32_t LLGL::FrameProfile::bufferCopies

Counter for all buffer copies during command encoding.

See also

CommandBuffer::CopyBuffer

10.36.4.4 std::uint32_t LLGL::FrameProfile::bufferMappings

Counter for all buffer map/unmap operations outside of command encoding.

See also

RenderSystem::MapBuffer.
RenderSystem::UnmapBuffer.

10.36.4.5 std::uint32_t LLGL::FrameProfile::bufferReads

Counter for all buffer write operations outside of command encoding.

Todo Not available yet.

10.36.4.6 std::uint32_t LLGL::FrameProfile::bufferUpdates

Counter for all buffer updates during command encoding.

See also

CommandBuffer::UpdateBuffer

10.36.4.7 std::uint32_t LLGL::FrameProfile::bufferWrites

Counter for all buffer write operations outside of command encoding.

See also

RenderSystem::WriteBuffer

10.36.4.8 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

10.36.4.9 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 ∗)

10.36.4.10 std::uint32_t LLGL::FrameProfile::computePipelineBindings

Counter for all compute pipeline bindings.

See also

CommandBuffer::SetComputePipeline

10.36.4.11 std::uint32_t LLGL::FrameProfile::computeResourceHeapBindings

Counter for all resource heap bindings on the compute pipeline.

See also

CommandBuffer::SetComputeResourceHeap

10.36.4.12 std::uint32_t LLGL::FrameProfile::constantBufferBindings

Counter for all individual constant buffer bindings (i.e. without a ResourceHeap).

See also

CommandBufferExt::SetConstantBuffer

10.36.4.13 std::uint32_t LLGL::FrameProfile::dispatchCommands

Counter for dispatch compute commands.

See also

CommandBuffer::Dispatch

10.36.4.14 std::uint32_t LLGL::FrameProfile::drawCommands

Counter for all draw commands.

See also

CommandBuffer::Draw
CommandBuffer::DrawIndexed
CommandBuffer::DrawInstanced
CommandBuffer::DrawIndexedInstanced

10.36.4.15 std::uint32_t LLGL::FrameProfile::fenceSubmissions

Counter for all fences that were submitted to the queue.

See also

CommandQueue::Submit(Fence&)

10.36.4.16 std::uint32_t LLGL::FrameProfile::graphicsPipelineBindings

Counter for all graphics pipeline bindings.

See also

CommandBuffer::SetGraphicsPipeline

10.36.4.17 std::uint32_t LLGL::FrameProfile::graphicsResourceHeapBindings

Counter for all resource heap bindings on the graphics pipeline.

See also

CommandBuffer::SetGraphicsResourceHeap

10.36.4.18 std::uint32_t LLGL::FrameProfile::indexBufferBindings

Counter for all index buffer bindings.

See also

CommandBuffer::SetIndexBuffer

10.36.4.19 std::uint32_t LLGL::FrameProfile::mipMapsGenerations

Counter for all MIP-map generations.

See also

RenderSystem::GenerateMips

10.36.4.20 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

10.36.4.21 std::uint32_t LLGL::FrameProfile::renderConditionSections

Counter for all command buffer sections that are enclosed by a call to `BeginRenderCondition` and `End↩RenderCondition`.

See also

CommandBuffer::BeginRenderCondition
CommandBuffer::EndRenderCondition

10.36.4.22 std::uint32_t LLGL::FrameProfile::renderPassSections

Counter for all command buffer sections that are enclosed by a call to `BeginRenderPass` and `EndRender↩Pass`.

See also

CommandBuffer::BeginRenderPass
CommandBuffer::EndRenderPass

10.36.4.23 std::uint32_t LLGL::FrameProfile::samplerBindings

Counter for all individual sampler bindings (i.e. without a ResourceHeap).

See also

CommandBufferExt::SetSampler

10.36.4.24 std::uint32_t LLGL::FrameProfile::storageBufferBindings

Counter for all individual storage buffer bindings (i.e. without a ResourceHeap).

See also

CommandBufferExt::SetStorageBuffer

10.36.4.25 std::uint32_t LLGL::FrameProfile::streamOutputBufferBindings

Counter for all stream-output buffer and stream-output buffer array bindings.

See also

CommandBuffer::SetStreamOutputBuffer
CommandBuffer::SetStreamOutputBufferArray

10.36.4.26 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

10.36.4.27 std::uint32_t LLGL::FrameProfile::textureBindings

Counter for all individual texture bindings (i.e. without a ResourceHeap).

See also

CommandBufferExt::SetTexture

10.36.4.28 std::uint32_t LLGL::FrameProfile::textureCopies

Counter for all texture copies during command encoding.

Todo Not available yet.

10.36.4.29 std::uint32_t LLGL::FrameProfile::textureMappings

Counter for all texture write operations outside of command encoding.

Todo Not available yet.

10.36.4.30 std::uint32_t LLGL::FrameProfile::textureReads

Counter for all texture write operations outside of command encoding.

See also

RenderSystem::ReadTexture.

10.36.4.31 std::uint32_t LLGL::FrameProfile::textureWrites

Counter for all texture write operations outside of command encoding.

See also

RenderSystem::WriteTexture.

10.36.4.32 std::uint32_t LLGL::FrameProfile::values[31]

All proflile values as linear array.

10.36.4.33 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:

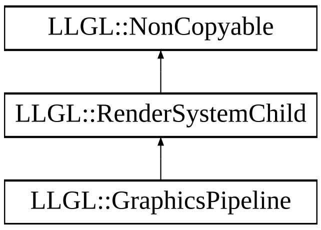
- RenderingProfiler.h

10.37 LLGL::GraphicsPipeline Class Reference

Graphics pipeline interface.

`#include <GraphicsPipeline.h>`

Inheritance diagram for LLGL::GraphicsPipeline:



Additional Inherited Members

10.37.1 Detailed Description

Graphics pipeline interface.

See also

RenderSystem::CreateGraphicsPipeline
CommandBuffer::SetGraphicsPipeline

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

- GraphicsPipeline.h

10.38 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.

10.38.1 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

10.38.2 Member Data Documentation

10.38.2.1 BlendDescriptor LLGL::GraphicsPipelineDescriptor::blend

Specifies the state descriptor for the blend stage.

10.38.2.2 DepthDescriptor LLGL::GraphicsPipelineDescriptor::depth

Specifies the depth state for the depth-stencil stage.

10.38.2.3 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.

10.38.2.4 PrimitiveTopology LLGL::GraphicsPipelineDescriptor::primitiveTopology = PrimitiveTopology::TriangleList

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

See also

PrimitiveTopology

10.38.2.5 RasterizerDescriptor LLGL::GraphicsPipelineDescriptor::rasterizer

Specifies the state for the rasterizer stage.

10.38.2.6 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

10.38.2.7 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

10.38.2.8 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

10.38.2.9 StencilDescriptor LLGL::GraphicsPipelineDescriptor::stencil

Specifies the stencil state for the depth-stencil stage.

10.38.2.10 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

10.39 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 src←
RegionExtent)

  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 QuerySrcDesc () const

  Returns a source image descriptor for this image with read-only access to the image data.
- DstImageDescriptor QueryDstDesc ()

  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.

10.39.1 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

10.39.2 Constructor & Destructor Documentation

10.39.2.1 LLGL::Image::Image ( ) `[default]`

10.39.2.2 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

10.39.2.3 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

10.39.2.4 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&&)

10.39.2.5 LLGL::Image::Image (const **Image** & *rhs*)

Copy constructor which copies the entire image buffer from the specified source image.

10.39.2.6 LLGL::Image::Image (**Image** && *rhs*)

Move constructor which takes the ownership of the specified source image.

10.39.3 Member Function Documentation

10.39.3.1 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

10.39.3.2 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

10.39.3.3 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.

10.39.3.4 std::uint32_t LLGL::Image::GetBytesPerPixel ( ) const

Returns the size (in bytes) for each pixel.

See also

GetFormat
ImageFormatSize
GetDataType
DataTypeSize

10.39.3.5 const void* LLGL::Image::GetData ( ) const [inline]

Returns the image data buffer as constant raw pointer.

10.39.3.6 void* LLGL::Image::GetData ( ) [inline]

Returns the image data buffer as raw pointer.

10.39.3.7 std::uint32_t LLGL::Image::GetDataSize ( ) const

Returns the size (in bytes) of the image buffer.

See also

GetBytesPerPixel
GetNumPixels

10.39.3.8 DataType LLGL::Image::GetDataType ( ) const [inline]

Returns the data type for each pixel component. By default DataType::UInt8.

10.39.3.9 std::uint32_t LLGL::Image::GetDepthStride ( ) const

Returns the stride (in bytes) for each depth slice.

10.39.3.10 const Extent3D& LLGL::Image::GetExtent ( ) const [inline]

Returns the extent of the image as 3D vector.

10.39.3.11 ImageFormat LLGL::Image::GetFormat ( ) const [inline]

Returns the format for each pixel. By default ImageFormat::RGBA.

10.39.3.12 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

10.39.3.13 std::uint32_t LLGL::Image::GetRowStride ( ) const

Returns the stride (in bytes) for each row.

10.39.3.14 bool LLGL::Image::IsRegionInside (const Offset3D & *offset,* const Extent3D & *extent*) const

Returns true if the specified sub-image region is inside the image.

10.39.3.15 void LLGL::Image::MirrorXYPlane ( )

Mirrors the image at the XY plane.

Todo Not implemented yet

10.39.3.16 void LLGL::Image::MirrorXZPlane ( )

Mirrors the image at the XZ plane.

Todo Not implemented yet

10.39.3.17 void LLGL::Image::MirrorYZPlane ( )

Mirrors the image at the YZ plane.

Todo Not implemented yet

10.39.3.18 Image& LLGL::Image::operator= (const Image & *rhs*)

Copy operator which copies the entire image buffer and attributes.

10.39.3.19 Image& LLGL::Image::operator= (Image && *rhs*)

Move operator which takes the ownership of the image buffer.

10.39.3.20 DstImageDescriptor LLGL::Image::QueryDstDesc ( )

Returns a destination image descriptor for this image with read/write access to the image data.

10.39.3.21 SrcImageDescriptor LLGL::Image::QuerySrcDesc ( ) const

Returns a source image descriptor for this image with read-only access to the image data.

10.39.3.22 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

10.39.3.23 ByteBuffer LLGL::Image::Release ( )

Releases the ownership of the image buffer and resets all attributes.

10.39.3.24 void LLGL::Image::Reset ( )

Resets all image attributes to its default values.

10.39.3.25 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

10.39.3.26 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!

10.39.3.27 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

10.39.3.28 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

10.39.3.29 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.

10.39.3.30 void LLGL::Image::Swap (Image & *rhs*)

Swaps all attributes with the specified image.

10.39.3.31 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

10.40 LLGL::ImageInitialization Struct Reference

Structure of image initialization for textures without initial image data.

```
#include <RenderSystemFlags.h>
```

Public Attributes

- bool enabled = true

  Enables or disables the default initialization of texture images. By default true.
- ClearValue clearValue

  Specifies the default value to clear uninitialized textures.

10.40.1 Detailed Description

Structure of image initialization for textures without initial image data.

10.40.2 Member Data Documentation

10.40.2.1 ClearValue LLGL::ImageInitialization::clearValue

Specifies the default value to clear uninitialized textures.

Todo Currently only supports initialization of color and depth. Default initialization of stencil values is not supported yet.

10.40.2.2 bool LLGL::ImageInitialization::enabled = true

Enables or disables the default initialization of texture images. By default true.

Remarks

This will be used when a texture is created and no initial image data is specified. If this is false and a texture is created without initial image data, the texture remains uninitialized.

Note

Reading or sampling uninitialized textures is undefined behavior.

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

- RenderSystemFlags.h

10.41 LLGL::BufferDescriptor::IndexBuffer Struct Reference

Index buffer specific descriptor structure.

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

Public Attributes

- IndexFormat format

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

10.41.1 Detailed Description

Index buffer specific descriptor structure.

10.41.2 Member Data Documentation

10.41.2.1 IndexFormat LLGL::BufferDescriptor::IndexBuffer::format

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

Remarks

The only valid format types for an index buffer are: DataType::UByte, DataType::UShort, and DataType::UInt.

See also

DataType

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

- BufferFlags.h

10.42 LLGL::IndexFormat Class Reference

Index buffer format class.

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

Public Member Functions

- IndexFormat ()=default
- IndexFormat (const IndexFormat &)=default
- IndexFormat & operator= (const IndexFormat &)=default
- IndexFormat (const DataType dataType)

    Constructor to initialize the index format with the specified data type.
- DataType GetDataType () const

    Returns the data type of this index format.
- std::uint32_t GetFormatSize () const

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

10.42.1 Detailed Description

Index buffer format class.

See also

BufferDescriptor::IndexBuffer

10.42.2 Constructor & Destructor Documentation

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

10.42.2.2 LLGL::IndexFormat::IndexFormat (const IndexFormat &) [default]

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

Constructor to initialize the index format with the specified data type.

Remarks

This will automatically determine the format size.

See also

GetFormatSize

10.42.3 Member Function Documentation

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

Returns the data type of this index format.

10.42.3.2 std::uint32_t LLGL::IndexFormat::GetFormatSize ( ) const [inline]

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

10.42.3.3 IndexFormat& LLGL::IndexFormat::operator= (const IndexFormat &) [default]

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

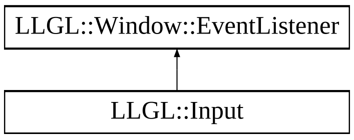
- IndexFormat.h

10.43 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 OnLoseFocus (Window &sender) override

  Send when the window loses the keyboard focus.

10.43.1 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 ...
}
```

10.43.2 Constructor & Destructor Documentation

10.43.2.1 LLGL::Input::Input ( )

10.43.3 Member Function Documentation

10.43.3.1 std::size_t LLGL::Input::GetAnyKeyCount ( ) const [inline]

Returns the number of any keys being pressed.

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

Returns the entered characters.

10.43.3.3 const Offset2D& LLGL::Input::GetMouseMotion ( ) const [inline]

Returns the global mouse motion.

10.43.3.4 const Offset2D& LLGL::Input::GetMousePosition ( ) const [inline]

Returns the local mouse position.

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

Returns the mouse wheel motion.

10.43.3.6 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.

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

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

10.43.3.8 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).

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

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

10.43.3.10 bool LLGL::Input::KeyUp (Key *keyCode*) const

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

10.43.3.11 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.

10.43.3.12 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.

10.43.3.13 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.

10.43.3.14 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.

10.43.3.15 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.

10.43.3.16 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.

10.43.3.17 void LLGL::Input::OnLoseFocus (Window & *sender*) [override], [protected], [virtual]

Send when the window loses the keyboard focus.

Reimplemented from LLGL::Window::EventListener.

10.43.3.18 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.

10.43.3.19 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

10.44 LLGL::RenderingDebugger::Message Class Reference

Rendering debugger message class.

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

Public Member Functions

- Message ()=default
- Message (const Message &)=default
- Message & operator= (const Message &)=default
- Message (const std::string &text, const std::string &source)
- void Block ()

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

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

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

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

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

  Returns true if this message has already been blocked.

Protected Member Functions

- void IncOccurrence ()

Friends

- class RenderingDebugger

10.44.1 Detailed Description

Rendering debugger message class.

10.44.2 Constructor & Destructor Documentation

10.44.2.1 LLGL::RenderingDebugger::Message::Message ( ) `[default]`

10.44.2.2 LLGL::RenderingDebugger::Message::Message (const Message &) `[default]`

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

10.44.3 Member Function Documentation

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

Blocks further occurrences of this message.

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

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

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

Returns the number of occurrences of this message.

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

Returns the source function where this message occurred.

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

Returns the message text.

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

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

Returns true if this message has already been blocked.

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

10.44.4 Friends And Related Function Documentation

10.44.4.1 friend class RenderingDebugger [friend]

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

- RenderingDebugger.h

10.45 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 = ∼0

  Specifies the bitmask for sample coverage. By default `0xFFFFFFFF`.

10.45.1 Detailed Description

Multi-sampling descriptor structure.

See also

RasterizerDescriptor::multiSampling

10.45.2 Constructor & Destructor Documentation

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

10.45.2.2 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.

10.45.3 Member Function Documentation

10.45.3.1 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.

10.45.4 Member Data Documentation

10.45.4.1 bool LLGL::MultiSamplingDescriptor::enabled = false

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

10.45.4.2 std::uint32_t LLGL::MultiSamplingDescriptor::sampleMask = ∼0

Specifies the bitmask for sample coverage. By default `0xFFFFFFFF`.

10.45.4.3 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

10.46 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

10.46.1 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!

10.46.2 Member Data Documentation

10.46.2.1 ::Colormap LLGL::NativeContextHandle::colorMap

10.46.2.2 ::Display∗ LLGL::NativeContextHandle::display

10.46.2.3 UIView∗ LLGL::NativeContextHandle::parentView

10.46.2.4 NSWindow∗ LLGL::NativeContextHandle::parentWindow

10.46.2.5 ::Window LLGL::NativeContextHandle::parentWindow

10.46.2.6 HWND LLGL::NativeContextHandle::parentWindow

10.46.2.7 int LLGL::NativeContextHandle::screen

10.46.2.8 ::XVisualInfo∗ LLGL::NativeContextHandle::visual

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

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

10.47 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

10.47.1 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!

10.47.2 Member Data Documentation

10.47.2.1 ::Display∗ LLGL::NativeHandle::display

10.47.2.2 UIView∗ LLGL::NativeHandle::view

10.47.2.3 ::XVisualInfo∗ LLGL::NativeHandle::visual

10.47.2.4 NSWindow∗ LLGL::NativeHandle::window

10.47.2.5 ::Window LLGL::NativeHandle::window

10.47.2.6 HWND LLGL::NativeHandle::window

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

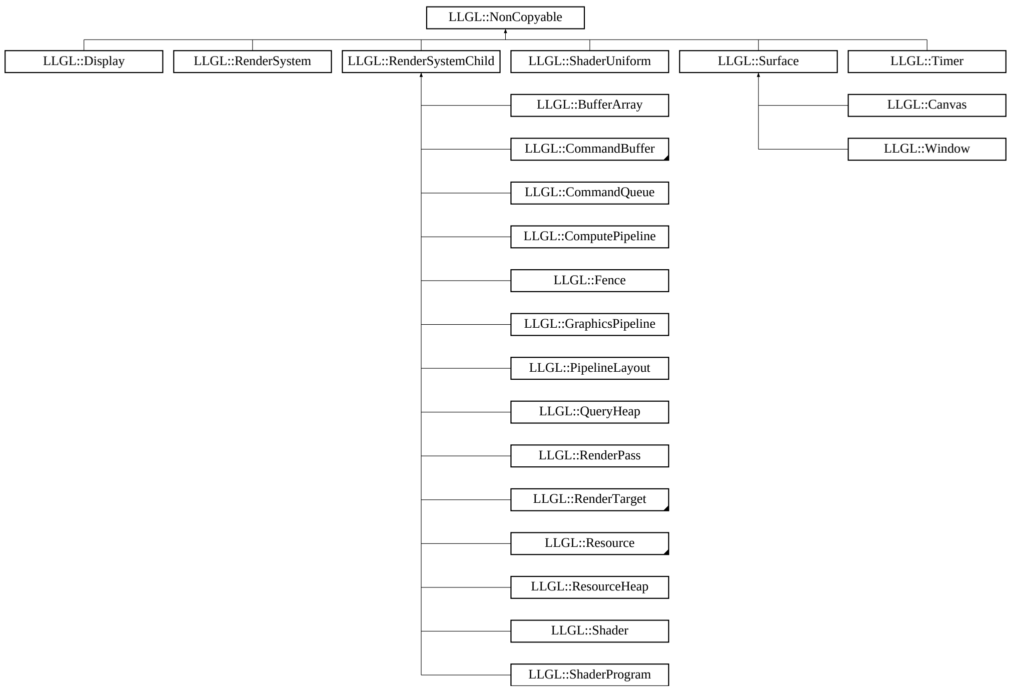
- IOSNativeHandle.h
- LinuxNativeHandle.h
- MacOSNativeHandle.h
- Win32NativeHandle.h

10.48 LLGL::NonCopyable Class Reference

Base class for all interfaces 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

10.48.1 Detailed Description

Base class for all interfaces 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

Display
RenderSystem
RenderSystemChild
ShaderUniform
Surface
Timer

10.48.2 Constructor & Destructor Documentation

10.48.2.1 LLGL::NonCopyable::NonCopyable (const NonCopyable &) `[delete]`

10.48.2.2 virtual LLGL::NonCopyable::∼NonCopyable ( ) `[virtual], [default]`

10.48.2.3 LLGL::NonCopyable::NonCopyable ( ) `[protected], [default]`

10.48.3 Member Function Documentation

10.48.3.1 NonCopyable& LLGL::NonCopyable::operator= (const NonCopyable &) `[delete]`

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

- NonCopyable.h

10.49 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.

10.49.1 Detailed Description

2-Dimensional offset structure.

Remarks

Used for signed integral 2D offsets (for coordinates in window-space, screen-space, and texture-space).

10.49.2 Constructor & Destructor Documentation

10.49.2.1 LLGL::Offset2D::Offset2D ( ) [default]

10.49.2.2 LLGL::Offset2D::Offset2D (const Offset2D &) [default]

10.49.2.3 LLGL::Offset2D::Offset2D (std::int32_t *x*, std::int32_t *y*) [inline]

10.49.3 Member Data Documentation

10.49.3.1 std::int32_t LLGL::Offset2D::x = 0

Offset X axis.

10.49.3.2 std::int32_t LLGL::Offset2D::y = 0

Offset Y axis.

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

- Types.h

10.50 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.

10.50.1 Detailed Description

3-Dimensional offset structure.

Remarks

Used for signed integral 3D offsets (for coordinates in texture-space).

10.50.2 Constructor & Destructor Documentation

10.50.2.1 LLGL::Offset3D::Offset3D ( ) `[default]`

10.50.2.2 LLGL::Offset3D::Offset3D (const Offset3D &) `[default]`

10.50.2.3 LLGL::Offset3D::Offset3D (std::int32_t *x*, std::int32_t *y*, std::int32_t *z*) `[inline]`

10.50.3 Member Data Documentation

10.50.3.1 std::int32_t LLGL::Offset3D::x = 0

Offset X axis.

10.50.3.2 std::int32_t LLGL::Offset3D::y = 0

Offset Y axis.

10.50.3.3 std::int32_t LLGL::Offset3D::z = 0

Offset Z axis.

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

- Types.h

10.51 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.

10.51.1 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

RenderContext::SetGraphicsAPIDependentState

10.51.2 Member Data Documentation

10.51.2.1 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.

10.51.2.2 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:

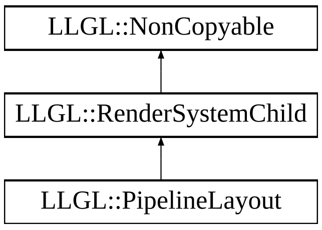
- CommandBufferFlags.h

10.52 LLGL::PipelineLayout Class Reference

Pipeline layout interface.

```
#include <PipelineLayout.h>
```

Inheritance diagram for LLGL::PipelineLayout:



Additional Inherited Members

10.52.1 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

10.53 LLGL::PipelineLayoutDescriptor Struct Reference

Pipeline layout descritpor structure.

```
#include <PipelineLayoutFlags.h>
```

Public Attributes

- std::vector< BindingDescriptor > bindings

  List of layout resource bindings.

10.53.1 Detailed Description

Pipeline layout descritpor structure.

Remarks

Contains all layout bindings that will be used by graphics and compute pipelines.

10.53.2 Member Data Documentation

10.53.2.1 std::vector<BindingDescriptor> LLGL::PipelineLayoutDescriptor::bindings

List of layout resource bindings.

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

- PipelineLayoutFlags.h

10.54 LLGL::ProfileOpenGLDescriptor Struct Reference

OpenGL profile descriptor structure.

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

Public Attributes

- OpenGLContextProfile contextProfile = OpenGLContextProfile::CompatibilityProfile

  Specifies the requested OpenGL context profile. By default OpenGLContextProfile::CompatibilityProfile.
- int majorVersion = -1

  Specifies the requested OpenGL context major version. By default -1 to indicate to use the highest version possible.
- int minorVersion = -1

  Specifies the requested OpenGL context minor version. By default -1 to indicate to use the highest version possible.

10.54.1 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.

10.54.2 Member Data Documentation

10.54.2.1 OpenGLContextProfile LLGL::ProfileOpenGLDescriptor::contextProfile = OpenGLContextProfile::↩ CompatibilityProfile

Specifies the requested OpenGL context profile. By default OpenGLContextProfile::CompatibilityProfile.

10.54.2.2 int LLGL::ProfileOpenGLDescriptor::majorVersion = -1

Specifies the requested OpenGL context major version. By default -1 to indicate to use the highest version possible.

Remarks

This member is ignored if 'contextProfile' is 'OpenGLContextProfile::CompatibilityProfile'.

10.54.2.3 int LLGL::ProfileOpenGLDescriptor::minorVersion = -1

Specifies the requested OpenGL context minor version. By default -1 to indicate to use the highest version possible.

Remarks

This member is ignored if 'contextProfile' is 'OpenGLContextProfile::CompatibilityProfile'.

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

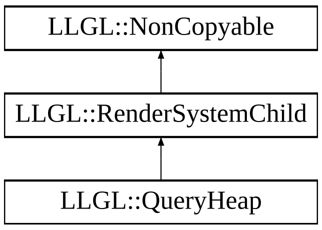
- RenderContextFlags.h

10.55 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)

10.55.1 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

10.55.2 Constructor & Destructor Documentation

10.55.2.1 LLGL::QueryHeap::QueryHeap (const QueryType *type*) `[protected]`

10.55.3 Member Function Documentation

10.55.3.1 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

10.56 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.

10.56.1 Detailed Description

Query heap descriptor structure.

See also

RenderSystem::CreateQueryHeap

10.56.2 Member Data Documentation

10.56.2.1 std::uint32_t LLGL::QueryHeapDescriptor::numQueries = 1

Specifies the number of queries in the heap. This must be greater than zero. By default 1.

10.56.2.2 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.

10.56.2.3 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

10.57 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.

10.57.1 Detailed Description

Query data structure for pipeline statistics.

Remarks

This structure is designed to be compatible to the equivalent in Direct3D 11 (i.e. `D3D11_QUERY_DATA_PIPELINE_STATISTICS`), Direct3D 12 (i.e. `D3D12_QUERY_DATA_PIPELINE_STATISTICS`), and Vulkan (i.e. `VkQueryPipelineStatisticFlagBits`).

See also

QueryType::PipelineStatistics
CommandQueue::QueryResult
RenderingFeatures::hasPipelineStatistics
`https://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_query_data_pipeline_statistics`
`https://docs.microsoft.com/en-us/windows/desktop/api/d3d12/ns-d3d12-d3d12_query_data_pipeline_statistics`
`https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/VkQueryPipelineStatisticFlagBits.html`

10.57.2 Member Data Documentation

10.57.2.1 std::uint64_t LLGL::QueryPipelineStatistics::clippingInvocations = 0

Number of primitives that reached the primitive clipping stage.

10.57.2.2 std::uint64_t LLGL::QueryPipelineStatistics::clippingPrimitives = 0

Number of primitives that passed the primitive clipping stage.

10.57.2.3 std::uint64_t LLGL::QueryPipelineStatistics::computeShaderInvocations = 0

Number of compute shader invocations.

10.57.2.4 std::uint64_t LLGL::QueryPipelineStatistics::fragmentShaderInvocations = 0

Number of fragment shader invocations.

10.57.2.5 std::uint64_t LLGL::QueryPipelineStatistics::geometryShaderInvocations = 0

Number of geometry shader invocations.

10.57.2.6 std::uint64_t LLGL::QueryPipelineStatistics::geometryShaderPrimitives = 0

Number of primitives generated by the geometry shader.

10.57.2.7 std::uint64_t LLGL::QueryPipelineStatistics::inputAssemblyPrimitives = 0

Number of primitives submitted to the input-assembly.

10.57.2.8 std::uint64_t LLGL::QueryPipelineStatistics::inputAssemblyVertices = 0

Number of vertices submitted to the input-assembly.

10.57.2.9 std::uint64_t LLGL::QueryPipelineStatistics::tessControlShaderInvocations = 0

Number of tessellation-control shader invocations.

10.57.2.10 std::uint64_t LLGL::QueryPipelineStatistics::tessEvaluationShaderInvocations = 0

Number of tessellation-evaluation shader invocations.

10.57.2.11 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

10.58 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.

10.58.1 Detailed Description

Rasterizer state descriptor structure.

See also

GraphicsPipelineDescriptor::rasterizer

10.58.2 Member Data Documentation

10.58.2.1 bool LLGL::RasterizerDescriptor::antiAliasedLineEnabled = false

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

10.58.2.2 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

10.58.2.3 CullMode LLGL::RasterizerDescriptor::cullMode = CullMode::Disabled

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

10.58.2.4 DepthBiasDescriptor LLGL::RasterizerDescriptor::depthBias

Specifies the parameters to bias fragment depth values.

10.58.2.5 bool LLGL::RasterizerDescriptor::depthClampEnabled = false

If enabled, there is effectively no near and far clipping plane. By default disabled.

10.58.2.6 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.

10.58.2.7 bool LLGL::RasterizerDescriptor::frontCCW = false

If enabled, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding. By default disabled.

10.58.2.8 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

10.58.2.9 MultiSamplingDescriptor LLGL::RasterizerDescriptor::multiSampling

(Multi-)sampling descriptor.

10.58.2.10 PolygonMode LLGL::RasterizerDescriptor::polygonMode = PolygonMode::Fill

Polygon render mode. By default PolygonMode::Fill.

10.58.2.11 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:

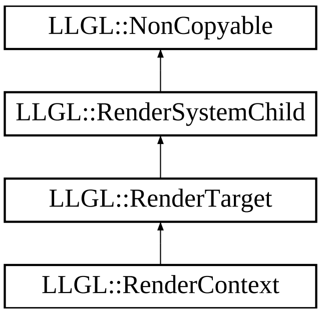
- GraphicsPipelineFlags.h

10.59 LLGL::RenderContext Class Reference

Render context interface.

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

Inheritance diagram for LLGL::RenderContext:



Public Member Functions

- bool IsRenderContext () const final

  Returns true.
- 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 QueryColorFormat () const =0

  Returns the color format of this render context.
- virtual Format QueryDepthStencilFormat () 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)

10.59.1 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&)

10.59.2 Constructor & Destructor Documentation

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

Default constructor with no effect.

10.59.2.2 LLGL::RenderContext::RenderContext (const VideoModeDescriptor & *initialVideoMode,* const VsyncDescriptor & *initialVsync*) [protected]

Constructor to initialize the render context with the specified video mode and V-sync.

10.59.3 Member Function Documentation

10.59.3.1 std::uint32_t LLGL::RenderContext::GetNumColorAttachments ( ) const [final], [virtual]

Returns 1, since each render context has always a single color attachment.

Implements LLGL::RenderTarget.

10.59.3.2 Extent2D LLGL::RenderContext::GetResolution ( ) const [final], [virtual]

Returns the resolution of the current video mode.

See also

GetVideoMode

Implements LLGL::RenderTarget.

10.59.3.3 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());
```

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

Returns the video mode for this render context.

10.59.3.5 const VsyncDescriptor& LLGL::RenderContext::GetVsync ( ) const [inline]

Returns the V-snyc configuration for this render context.

10.59.3.6 bool LLGL::RenderContext::HasDepthAttachment ( ) const [final], [virtual]

Returns true if this render context has a depth format.

See also

QueryDepthStencilFormat
IsDepthFormat

Implements LLGL::RenderTarget.

10.59.3.7 bool LLGL::RenderContext::HasStencilAttachment ( ) const [final], [virtual]

Returns true if this render context has a stencil format.

See also

QueryDepthStencilFormat
IsStencilFormat

Implements LLGL::RenderTarget.

10.59.3.8 bool LLGL::RenderContext::IsRenderContext ( ) const [final], [virtual]

Returns true.

Reimplemented from LLGL::RenderTarget.

10.59.3.9 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

10.59.3.10 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.

10.59.3.11 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).

10.59.3.12 virtual Format LLGL::RenderContext::QueryColorFormat ( ) 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

10.59.3.13 virtual Format LLGL::RenderContext::QueryDepthStencilFormat ( ) 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

10.59.3.14 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

10.59.3.15 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

10.59.3.16 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&)

10.59.3.17 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.

10.59.3.18 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.

10.59.3.19 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

10.60 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.
- ProfileOpenGLDescriptor profileOpenGL

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

  Debuging callback function object.

10.60.1 Detailed Description

Render context descriptor structure.

10.60.2 Member Data Documentation

10.60.2.1 DebugCallback LLGL::RenderContextDescriptor::debugCallback

Debuging callback function object.

10.60.2.2 MultiSamplingDescriptor LLGL::RenderContextDescriptor::multiSampling

Multi-sampling descriptor.

10.60.2.3 ProfileOpenGLDescriptor LLGL::RenderContextDescriptor::profileOpenGL

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

10.60.2.4 VideoModeDescriptor LLGL::RenderContextDescriptor::videoMode

Video mode descriptor.

10.60.2.5 VsyncDescriptor LLGL::RenderContextDescriptor::vsync

Vertical-synchronization (Vsync) descriptor.

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

- RenderContextFlags.h

10.61 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.

10.61.1 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

10.61.2 Member Data Documentation

10.61.2.1 const int LLGL::RendererID::Direct3D10 = 0x00000006 `[static]`

ID number for a Direct3D 10 renderer.

10.61.2.2 const int LLGL::RendererID::Direct3D11 = 0x00000007 `[static]`

ID number for a Direct3D 11 renderer.

10.61.2.3 const int LLGL::RendererID::Direct3D12 = 0x00000008 `[static]`

ID number for a Direct3D 12 renderer.

10.61.2.4 const int LLGL::RendererID::Direct3D9 = 0x00000005 `[static]`

ID number for a Direct3D 9 renderer.

10.61.2.5 const int LLGL::RendererID::Metal = 0x0000000a `[static]`

ID number for a Metal renderer.

10.61.2.6 const int LLGL::RendererID::OpenGL = 0x00000001 `[static]`

ID number for an OpenGL renderer.

10.61.2.7 const int LLGL::RendererID::OpenGLES1 = 0x00000002 `[static]`

ID number for an OpenGL ES 1 renderer.

10.61.2.8 const int LLGL::RendererID::OpenGLES2 = 0x00000003 [static]

ID number for an OpenGL ES 2 renderer.

10.61.2.9 const int LLGL::RendererID::OpenGLES3 = 0x00000004 [static]

ID number for an OpenGL ES 3 renderer.

10.61.2.10 const int LLGL::RendererID::Reserved = 0x000000ff [static]

Highest ID number for reserved future renderers. Value is 0x000000ff.

10.61.2.11 const int LLGL::RendererID::Undefined = 0x00000000 [static]

Undefined ID number.

10.61.2.12 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

10.62 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").

10.62.1 Detailed Description

Renderer basic information structure.

10.62.2 Member Data Documentation

10.62.2.1 std::string LLGL::RendererInfo::deviceName

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

10.62.2.2 std::string LLGL::RendererInfo::rendererName

Rendering API name and version (e.g. "OpenGL 4.6").

10.62.2.3 std::string LLGL::RendererInfo::shadingLanguageName

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

10.62.2.4 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

10.63 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.

10.63.1 Detailed Description

Structure with all attributes describing the rendering capabilities of the render system.

See also

RenderSystem::GetRenderingCaps

10.63.2 Member Data Documentation

10.63.2.1 ClippingRange LLGL::RenderingCapabilities::clippingRange = ClippingRange::ZeroToOne

Specifies the clipping depth range.

10.63.2.2 RenderingFeatures LLGL::RenderingCapabilities::features

Specifies all supported hardware features.

Remarks

Especially with OpenGL these features can vary between different hardware and GL versions.

10.63.2.3 RenderingLimits LLGL::RenderingCapabilities::limits

Specifies all rendering limitations.

Remarks

Especially with OpenGL these features can vary between different hardware and GL versions.

10.63.2.4 ScreenOrigin LLGL::RenderingCapabilities::screenOrigin = ScreenOrigin::UpperLeft

Screen coordinate system origin.

Remarks

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

10.63.2.5 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

10.63.2.6 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

10.64 LLGL::RenderingDebugger Class Reference

Rendering debugger interface.

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

Classes

- class Message

  Rendering debugger message class.

Public Member Functions

- virtual ∼RenderingDebugger ()
- void SetSource (const char ∗source)

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

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

  Posts a warning message.

Protected Member Functions

- 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.

10.64.1 Detailed Description

Rendering debugger interface.

Remarks

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

10.64.2 Constructor & Destructor Documentation

10.64.2.1 virtual LLGL::RenderingDebugger::∼RenderingDebugger ( ) [virtual]

10.64.3 Member Function Documentation

10.64.3.1 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

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

Callback function when a warning was posted.

See also

RenderingDebugger::PostWarning
OnError

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

Posts an error message.

Parameters

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

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

Posts a warning message.

Parameters

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

10.64.3.5 void LLGL::RenderingDebugger::SetSource (const char ∗ *source*)

Sets the new source function name.

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

- RenderingDebugger.h

10.65 LLGL::RenderingFeatures Struct Reference

Contains the attributes for all supported rendering features.

```
#include <RenderSystemFlags.h>
```

Public Attributes

- bool hasCommandBufferExt = false

  Specifies whether the render system supports extended command buffers with dynamic state access for shader resources.
- 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 hasSamplers = false

  Specifies whether samplers are supported.
- bool hasConstantBuffers = false

  Specifies whether constant buffers (also "uniform buffer objects") are supported.
- bool hasStorageBuffers = false

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

- bool hasUniforms = false

  Specifies whether individual shader uniforms are supported (typically only for OpenGL 2.0+).
- bool hasGeometryShaders = false

  Specifies whether geometry shaders are supported.
- bool hasTessellationShaders = false

  Specifies whether tessellation shaders are supported.
- bool hasComputeShaders = false

  Speciifes whether compute shaders are supported.
- bool hasInstancing = false

  Specifies whether hardware instancing is supported.
- bool hasOffsetInstancing = false

  Specifies whether hardware instancing with instance offsets is supported.
- bool hasViewportArrays = false

  Specifies whether multiple viewports, depth-ranges, and scissors 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.

10.65.1 Detailed Description

Contains the attributes for all supported rendering features.

See also

RenderingCapabilities

10.65.2 Member Data Documentation

10.65.2.1 bool LLGL::RenderingFeatures::has3DTextures = false

Specifies whether 3D textures are supported.

See also

TextureType::Texture3D

10.65.2.2 bool LLGL::RenderingFeatures::hasArrayTextures = false

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

See also

TextureType::Texture1DArray
TextureType::Texture2DArray

10.65.2.3 bool LLGL::RenderingFeatures::hasCommandBufferExt = false

Specifies whether the render system supports extended command buffers with dynamic state access for shader resources.

Remarks

This is only supported by older graphics APIs such as OpenGL and Direct3D 11.

See also

RenderSystem::CreateCommandBufferExt
CommandBufferExt

10.65.2.4 bool LLGL::RenderingFeatures::hasComputeShaders = false

Speciifes whether compute shaders are supported.

10.65.2.5 bool LLGL::RenderingFeatures::hasConservativeRasterization = false

Specifies whether conservative rasterization is supported.

See also

RasterizerDescriptor::conservativeRasterization

10.65.2.6 bool LLGL::RenderingFeatures::hasConstantBuffers = false

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

See also

BufferType::Constant

10.65.2.7 bool LLGL::RenderingFeatures::hasCubeArrayTextures = false

Specifies whether cube array textures are supported.

See also

TextureType::TextureCubeArray

10.65.2.8 bool LLGL::RenderingFeatures::hasCubeTextures = false

Specifies whether cube textures are supported.

See also

TextureType::TextureCube

10.65.2.9 bool LLGL::RenderingFeatures::hasGeometryShaders = false

Specifies whether geometry shaders are supported.

10.65.2.10 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)

10.65.2.11 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

10.65.2.12 bool LLGL::RenderingFeatures::hasMultiSampleTextures = false

Specifies whether multi-sample textures are supported.

See also

TextureType::Texture2DMS
TextureType::Texture2DMSArray

10.65.2.13 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)

10.65.2.14 bool LLGL::RenderingFeatures::hasPipelineStatistics = false

Specifies whether queries for pipeline statistics are supported.

See also

QueryType::PipelineStatistics
QueryPipelineStatistics

10.65.2.15 bool LLGL::RenderingFeatures::hasRenderCondition = false

Specifies whether queries for conditional rendering are supported.

See also

QueryHeapDescriptor::renderCondition
CommandBuffer:BeginRenderCondition

10.65.2.16 bool LLGL::RenderingFeatures::hasRenderTargets = false

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

10.65.2.17 bool LLGL::RenderingFeatures::hasSamplers = false

Specifies whether samplers are supported.

10.65.2.18 bool LLGL::RenderingFeatures::hasStorageBuffers = false

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

See also

BufferType::Storage

10.65.2.19 bool LLGL::RenderingFeatures::hasStreamOutputs = false

Specifies whether stream-output is supported.

See also

ShaderSource::streamOutput
CommandBuffer::BeginStreamOutput

10.65.2.20 bool LLGL::RenderingFeatures::hasTessellationShaders = false

Specifies whether tessellation shaders are supported.

10.65.2.21 bool LLGL::RenderingFeatures::hasUniforms = false

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

See also

ShaderProgram::LockShaderUniform

10.65.2.22 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

10.66 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.

10.66.1 Detailed Description

Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.

See also

RenderingCapabilities

10.66.2 Member Data Documentation

10.66.2.1 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

10.66.2.2 std::uint32_t LLGL::RenderingLimits::max1DTextureSize = 0

Specifies the maximum size of each 1D texture.

See also

TextureDescriptor::extent

10.66.2.3 std::uint32_t LLGL::RenderingLimits::max2DTextureSize = 0

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

See also

TextureDescriptor::extent

10.66.2.4 std::uint32_t LLGL::RenderingLimits::max3DTextureSize = 0

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

See also

TextureDescriptor::extent

10.66.2.5 std::uint32_t LLGL::RenderingLimits::maxAnisotropy = 0

Specifies the maximum anisotropy texture filter.

See also

SamplerDescriptor::maxAnisotropy

10.66.2.6 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

10.66.2.7 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

10.66.2.8 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

10.66.2.9 std::uint32_t LLGL::RenderingLimits::maxComputeShaderWorkGroupSize[3] = { 0, 0, 0 }

Specifies the maximum work group size in a compute shader.

10.66.2.10 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

10.66.2.11 std::uint32_t LLGL::RenderingLimits::maxCubeTextureSize = 0

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

See also

TextureDescriptor::extent

10.66.2.12 std::uint32_t LLGL::RenderingLimits::maxPatchVertices = 0

Specifies the maximum number of patch control points.

See also

PrimitiveTopology::Patches1
PrimitiveTopology::Patches32

10.66.2.13 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

10.66.2.14 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

10.66.2.15 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

10.67 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.

10.67.1 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.

10.67.2 Member Function Documentation

10.67.2.1 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

10.67.2.2 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.

10.67.3 Member Data Documentation

10.67.3.1 FrameProfile LLGL::RenderingProfiler::frameProfile

Current frame profile with all counter values.

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

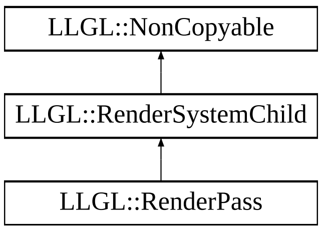
- RenderingProfiler.h

10.68 LLGL::RenderPass Class Reference

Render pass interface.

```
#include <RenderPass.h>
```

Inheritance diagram for LLGL::RenderPass:



Additional Inherited Members

10.68.1 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

10.69 LLGL::RenderPassDescriptor Struct Reference

Render pass descriptor structure.

```
#include <RenderPassFlags.h>
```

Public Attributes

- std::vector< AttachmentFormatDescriptor > colorAttachments
- AttachmentFormatDescriptor depthAttachment

    Specifies the depth attachment used within the render pass.
- AttachmentFormatDescriptor stencilAttachment

    Specifies the depth attachment used within the render pass.

10.69.1 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

10.69.2 Member Data Documentation

10.69.2.1 std::vector<**AttachmentFormatDescriptor**> LLGL::RenderPassDescriptor::colorAttachments

Specifies the color attachments used within the render pass.

Remarks

A render context usually uses an BGRA format instead of an RGBA format.

See also

RenderingLimits::maxColorAttachments
Format::BGRA8UNorm
Format::BGRA8sRGB

10.69.2.2 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.

10.69.2.3 AttachmentFormatDescriptor LLGL::RenderPassDescriptor::stencilAttachment

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.

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

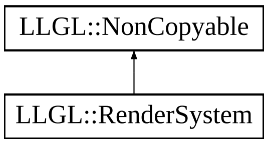
- RenderPassFlags.h

10.70 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 CommandBufferExt ∗ CreateCommandBufferExt (const CommandBufferDescriptor &desc={})=0

  Creates a new extended command buffer (if supported) with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).
- 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 void GenerateMips (Texture &texture)=0

  Generates all MIP-maps for the specified texture.
- virtual void GenerateMips (Texture &texture, std::uint32_t baseMipLevel, std::uint32_t numMipLevels, std↩ ::uint32_t baseArrayLayer=0, std::uint32_t numArrayLayers=1)=0

  Generates the specified range of MIP-maps for the specified texture.
- virtual Sampler ∗ CreateSampler (const SamplerDescriptor &desc)=0

  Creates a new Sampler object.
- virtual void Release (Sampler &sampler)=0

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

10.70.1 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);
```

10.70.2 Constructor & Destructor Documentation

10.70.2.1 LLGL::RenderSystem::RenderSystem ( ) [protected], [default]

10.70.3 Member Function Documentation

10.70.3.1 void LLGL::RenderSystem::AssertCreateBuffer (const **BufferDescriptor** & *desc*, std::uint64_t *maxSize*) [protected]

Validates the specified buffer descriptor to be used for buffer creation.

10.70.3.2 void LLGL::RenderSystem::AssertCreateBufferArray (std::uint32_t *numBuffers,* Buffer ∗const ∗ *bufferArray*) [protected]

Validates the specified arguments to be used for buffer array creation.

10.70.3.3 void LLGL::RenderSystem::AssertCreateRenderPass (const RenderPassDescriptor & *desc*) [protected]

Validates the specified render pass descriptor.

10.70.3.4 void LLGL::RenderSystem::AssertCreateRenderTarget (const RenderTargetDescriptor & *desc*) [protected]

Validates the specified render target descriptor.

10.70.3.5 void LLGL::RenderSystem::AssertCreateShader (const ShaderDescriptor & *desc*) [protected]

Validates the specified shader descriptor.

10.70.3.6 void LLGL::RenderSystem::AssertCreateShaderProgram (const ShaderProgramDescriptor & *desc*) [protected]

Validates the specified shader program descriptor.

10.70.3.7 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).

10.70.3.8 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

10.70.3.9 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

This array can only contain buffers which are all from the same type, like an array of vertex buffers for instance. The buffers inside this array must persist as long as this buffer array is used, and the individual buffers are still required to read and write its data from and to the GPU.

Exceptions

std::invalid_argument	If 'numBuffers' is 0, if 'bufferArray' is null, if any of the pointers in the array are null, if not all buffers have the same type, or if the buffer array type is not one of these: BufferType::Vertex, BufferType::Constant, BufferType::Storage, or BufferType::StreamOutput.

10.70.3.10 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.

10.70.3.11 virtual **CommandBufferExt**∗ LLGL::RenderSystem::CreateCommandBufferExt (const **CommandBufferDescriptor** & *desc* = { }) [pure virtual]

Creates a new extended command buffer (if supported) with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).

Returns

Pointer to the new CommandBufferExt object, or null if the render system does not support extended command buffers.

Remarks

For those render systems that do not support dynamic state access for shader resources, use the Resource↩ Heap interface.

Note

Only supported with: OpenGL, Direct3D 11, Metal.

See also

RenderingCapabilities::hasCommandBufferExt
CreateResourceHeap

10.70.3.12 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

10.70.3.13 virtual Fence∗ LLGL::RenderSystem::CreateFence ( ) `[pure virtual]`

Creates a new fence (used for CPU/GPU synchronization).

See also

CommandBuffer::SubmitFence
CommandBuffer::WaitFence

10.70.3.14 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

10.70.3.15 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

CommandBufferExt
CreateResourceHeap

10.70.3.16 virtual **QueryHeap**∗ LLGL::RenderSystem::CreateQueryHeap (const **QueryHeapDescriptor** & *desc*) `[pure virtual]`

Creates a new query heap.

10.70.3.17 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

10.70.3.18 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

10.70.3.19 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).

10.70.3.20 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

10.70.3.21 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

10.70.3.22 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 `QueryInfoLog` functions of the Shader interface.

See also

Shader::HasErrors
Shader::QueryInfoLog
ShaderDescriptor
ShaderDescFromFile

10.70.3.23 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 `QueryInfoLog` functions of the ShaderProgram interface.

See also

ShaderProgram::HasErrors
ShaderProgram::QueryInfoLog
ShaderProgramDescriptor
ShaderProgramDesc

10.70.3.24 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 is non-null, it is used to initialize the texture data. This parameter will be ignored if the texture type is a multi-sampled texture (i.e. TextureType::Texture2DMS or TextureType::Texture2DMSArray).

See also

WriteTexture
RenderSystemConfiguration::imageInitialization

10.70.3.25 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" }.

10.70.3.26 virtual void LLGL::RenderSystem::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.

Remarks

To generate only a small amout of MIP levels, use the secondary `GenerateMips` function. To update the MIP levels during encoding a command buffer, use CommandBuffer::GenerateMips.

See also

GenerateMips(Texture&, std::uint32_t, std::uint32_t, std::uint32_t, std::uint32_t)

10.70.3.27 virtual void LLGL::RenderSystem::GenerateMips (Texture & *texture*, std::uint32_t *baseMipLevel*, std::uint32_t *numMipLevels*, std::uint32_t *baseArrayLayer* = `0`, std::uint32_t *numArrayLayers* = `1`) `[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.
`in`	*baseMipLevel*	Specifies the zero-based index of the first MIP-map level.
`in`	*numMipLevels*	Specifies the number of MIP-maps to generate. This also includes the base MIP-map level, so a number of less than 2 has no effect.
`in`	*baseArrayLayer*	Specifies the zero-based index of the first array layer (if an array texture is used). By default 0.
`in`	*numArrayLayers*	Specifies the number of array layers. For both array textures and non-array textures this must be at least 1. By default 1.

Remarks

This function only guarantees to generate at least the specified amount of MIP-maps. It may also update all other MIP-maps if the respective rendering API does not support hardware accelerated generation of a sub-range of MIP-maps. To update the MIP levels during encoding a command buffer, use CommandBuffer↩ ::GenerateMips.

Note

Only use this function 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&)
NumMipLevels

10.70.3.28 virtual **CommandQueue**∗ LLGL::RenderSystem::GetCommandQueue ( ) `[pure virtual]`

Returns the single instance of the command queue.

10.70.3.29 const **RenderSystemConfiguration**& LLGL::RenderSystem::GetConfiguration ( ) const `[inline]`

Returns the basic configuration.

See also

SetConfiguration

10.70.3.30 const std::string& LLGL::RenderSystem::GetName ( ) const `[inline]`

Returns the name of this render system.

10.70.3.31 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

10.70.3.32 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!

10.70.3.33 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!

10.70.3.34 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 Log::SetReportCallback or Log::SetReportCallbackStd.

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

10.70.3.35 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

10.70.3.36 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::QueryDesc or Texture::QueryMipExtent functions can be used to determine the texture dimensions.

```
// Query texture size attribute
auto myTextureExtent = myTexture->QueryMipExtent(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::QueryDesc
Texture::QueryMipExtent

10.70.3.37 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.

10.70.3.38 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.

Remarks

This can be used for both CommandBuffer and CommandBufferExt objects as the latter one inherits from the former one.

See also

CreateCommandBuffer
CreateCommandBufferExt

10.70.3.39 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.

10.70.3.40 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.

10.70.3.41 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.

10.70.3.42 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.

10.70.3.43 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.

10.70.3.44 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.

10.70.3.45 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.

10.70.3.46 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.

10.70.3.47 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.

10.70.3.48 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.

10.70.3.49 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.

10.70.3.50 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.

10.70.3.51 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.

10.70.3.52 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.

10.70.3.53 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

10.70.3.54 void LLGL::RenderSystem::SetRendererInfo (const **RendererInfo** & *info*) `[protected]`

Sets the renderer information.

10.70.3.55 void LLGL::RenderSystem::SetRenderingCaps (const RenderingCapabilities & *caps*) [protected]

Sets the rendering capabilities.

10.70.3.56 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!

10.70.3.57 virtual void LLGL::RenderSystem::UnmapBuffer (Buffer & *buffer*) [pure virtual]

Unmaps the specified buffer.

See also

MapBuffer

10.70.3.58 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 Command↩ Buffer::UpdateBuffer.

10.70.3.59 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.
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:

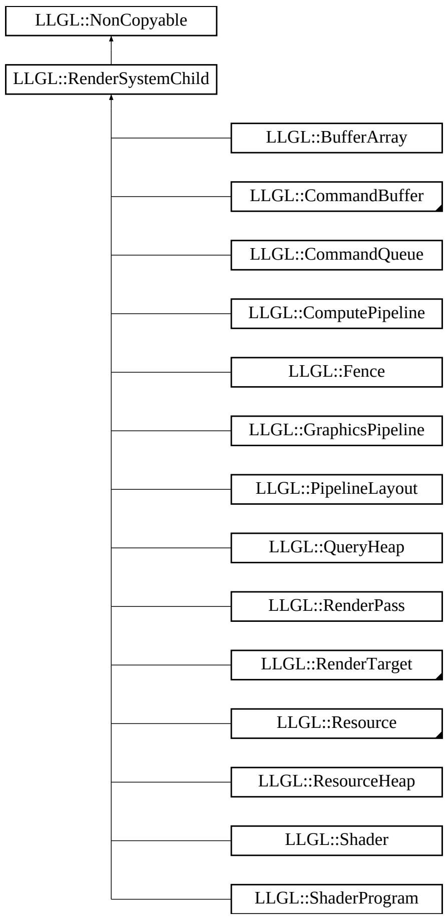
- RenderSystem.h

10.71 LLGL::RenderSystemChild Class Reference

Base class for all interfaces whoes instances are owned by the RenderSystem.

`#include <RenderSystemChild.h>`

Inheritance diagram for LLGL::RenderSystemChild:



Additional Inherited Members

10.71.1 Detailed Description

Base class for all interfaces whoes instances are owned by the RenderSystem.

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

- RenderSystemChild.h

10.72 LLGL::RenderSystemConfiguration Struct Reference

Render system configuration structure.

```
#include <RenderSystemFlags.h>
```

Public Attributes

- ImageInitialization imageInitialization

    Image initialization for textures without initial image data.
- 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.

10.72.1 Detailed Description

Render system configuration structure.

10.72.2 Member Data Documentation

10.72.2.1 ImageInitialization LLGL::RenderSystemConfiguration::imageInitialization

Image initialization for textures without initial image data.

10.72.2.2 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

10.73 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.
- 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.

10.73.1 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

10.73.2 Constructor & Destructor Documentation

10.73.2.1 LLGL::RenderSystemDescriptor::RenderSystemDescriptor ( ) [default]

10.73.2.2 LLGL::RenderSystemDescriptor::RenderSystemDescriptor (const RenderSystemDescriptor &) [default]

10.73.2.3 LLGL::RenderSystemDescriptor::RenderSystemDescriptor (const std::string & *moduleName*) [inline]

Constructor to initialize the descriptor with the module name form an std::string.

10.73.2.4 LLGL::RenderSystemDescriptor::RenderSystemDescriptor (const char ∗ *moduleName*) [inline]

Constructor to initialize the descriptor with the module name form a null terminated string.

10.73.3 Member Function Documentation

10.73.3.1 RenderSystemDescriptor& LLGL::RenderSystemDescriptor::operator= (const RenderSystemDescriptor &) [default]

10.73.4 Member Data Documentation

10.73.4.1 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 "LLG↩
L_OpenGLD.dll", if compiled on Windows in Debug mode.

10.73.4.2 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::VulkanRendererConfiguration 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
VulkanRendererConfiguration

10.73.4.3 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:

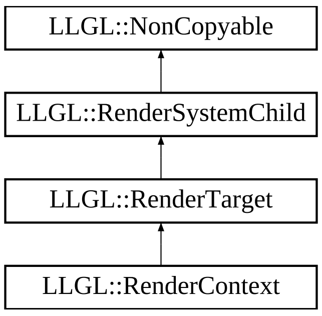
- RenderSystemFlags.h

10.74 LLGL::RenderTarget Class Reference

Render target interface.

```
#include <RenderTarget.h>
```

Inheritance diagram for LLGL::RenderTarget:



Public Member Functions

- virtual bool IsRenderContext () const

  Returns true if this render target is an instance of RenderContext. By default false.
- 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.

10.74.1 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

10.74.2 Member Function Documentation

10.74.2.1 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::QueryColorFormat

Implemented in LLGL::RenderContext.

10.74.2.2 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

10.74.2.3 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.

10.74.2.4 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::QueryDepthStencilFormat

Implemented in LLGL::RenderContext.

10.74.2.5 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::QueryDepthStencilFormat

Implemented in LLGL::RenderContext.

10.74.2.6 virtual bool LLGL::RenderTarget::IsRenderContext ( ) const [virtual]

Returns true if this render target is an instance of RenderContext. By default false.

Remarks

Do not override this function. Only the sub class RenderContext is supposed to override it.

See also

RenderContext::IsRenderContext

Reimplemented in LLGL::RenderContext.

10.74.2.7 void LLGL::RenderTarget::ValidateMipResolution (const Texture & *texture,* std::uint32_t *mipLevel*) [protected]

Applies the resolution of the texture MIP level.

See also

Texture::QueryMipExtent
ValidateResolution

10.74.2.8 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.

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

- RenderTarget.h

10.75 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.

10.75.1 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->QueryMipExtent(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

10.75.2 Member Data Documentation

10.75.2.1 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.

10.75.2.2 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.

10.75.2.3 MultiSamplingDescriptor LLGL::RenderTargetDescriptor::multiSampling

Multi-sampling descriptor. By default, multi-sampling is disabled.

10.75.2.4 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

10.75.2.5 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::QueryMipExtent

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

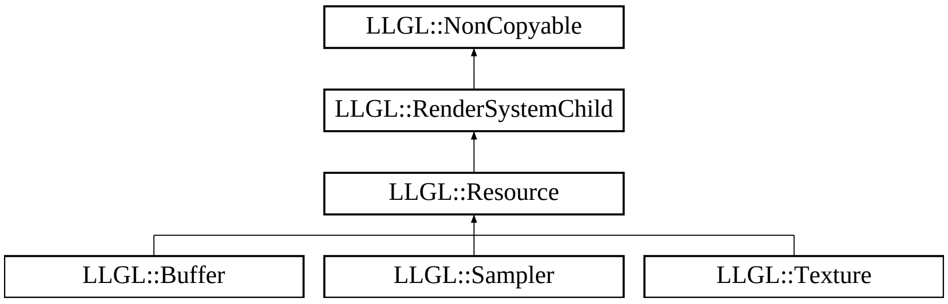
- RenderTargetFlags.h

10.76 LLGL::Resource Class Reference

Base class for all hardware resource interfaces.

`#include <Resource.h>`

Inheritance diagram for LLGL::Resource:



Public Member Functions

- virtual ResourceType QueryResourceType () const =0

  Returns the type of this resource object.

Additional Inherited Members

10.76.1 Detailed Description

Base class for all hardware resource interfaces.

See also

Buffer
Texture
Sampler

10.76.2 Member Function Documentation

10.76.2.1 virtual ResourceType LLGL::Resource::QueryResourceType ( ) 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

Implemented in LLGL::Texture, LLGL::Buffer, and LLGL::Sampler.

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

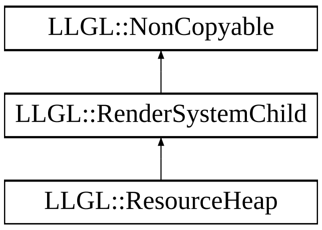
- Resource.h

10.77 LLGL::ResourceHeap Class Reference

Resource heap interface.

```
#include <ResourceHeap.h>
```

Inheritance diagram for LLGL::ResourceHeap:



Additional Inherited Members

10.77.1 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

10.78 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.

10.78.1 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

10.78.2 Member Data Documentation

10.78.2.1 PipelineLayout∗ LLGL::ResourceHeapDescriptor::pipelineLayout = nullptr

Reference to the pipeline layout. This must not be null, when a resource heap is created.

10.78.2.2 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

10.79 LLGL::ShaderReflectionDescriptor::ResourceView Struct Reference

Shader reflection resource view structure.

`#include <ShaderProgramFlags.h>`

Public Attributes

- std::string name

  Name of the shader resource, i.e. the identifier used in the shader.
- ResourceType type = ResourceType::Undefined

  Resource view type for this layout binding. By default ResourceType::Undefined.
- long stageFlags = 0

  Specifies in which shader stages the resource is located. 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.
- 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.

10.79.1 Detailed Description

Shader reflection resource view structure.

Remarks

A mapping between this structure and a binding descriptor may look like this:

```
auto myShaderReflectionDesc = myShaderProgram->QueryReflectionDesc();
LLGL::PipelineLayoutDescriptor myPipelineLayoutDesc;
for (const auto& myResourceView : myShaderReflectionDesc) {
    BindingDescriptor myBindingDesc;
    myBindingDesc.type          = myResourceView.type;
    myBindingDesc.stageFlags    = myResourceView.stageFlags;
    myBindingDesc.slot          = myResourceView.slot;
    myBindingDesc.arraySize     = myResourceView.arraySize;
    myPipelineLayoutDesc.bindings.push_back(myBindingDesc);
}
```

See also

BindingDescriptor

10.79.2 Member Data Documentation

10.79.2.1 std::uint32_t LLGL::ShaderReflectionDescriptor::ResourceView::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).

10.79.2.2 std::uint32_t LLGL::ShaderReflectionDescriptor::ResourceView::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

10.79.2.3 std::string LLGL::ShaderReflectionDescriptor::ResourceView::name

Name of the shader resource, i.e. the identifier used in the shader.

10.79.2.4 std::uint32_t LLGL::ShaderReflectionDescriptor::ResourceView::slot = 0

Specifies the zero-based binding slot. By default 0.

Remarks

If the binding slot could be not queried by the shader reflection, the value is Constants::invalidSlot.

See also

Constants::invalidSlot

10.79.2.5 long LLGL::ShaderReflectionDescriptor::ResourceView::stageFlags = 0

Specifies in which shader stages the resource is located. By default 0.

Remarks

This can be a bitwise OR combination of the StageFlags bitmasks.

See also

StageFlags

10.79.2.6 StorageBufferType LLGL::ShaderReflectionDescriptor::ResourceView::storageBufferType = StorageBufferType::Undefined

Specifies the sub-type of a storage buffer resource.

Remarks

Additional attribute exclusively used for storage buffer resources.

10.79.2.7 ResourceType LLGL::ShaderReflectionDescriptor::ResourceView::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:

- ShaderProgramFlags.h

10.80 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 Buffer resource view.

Public Attributes

- Resource ∗ resource = nullptr

  Pointer to the hardware resoudce.

10.80.1 Detailed Description

Resource view descriptor structure.

See also

ResourceHeapDescriptor::resourceViews

10.80.2 Constructor & Destructor Documentation

10.80.2.1 LLGL::ResourceViewDescriptor::ResourceViewDescriptor ( ) [default]

Default constructor to initialize the resource view with a null pointer.

10.80.2.2 LLGL::ResourceViewDescriptor::ResourceViewDescriptor (Resource ∗ *resource*) [inline]

Constructor to initialize the descriptor with a Buffer resource view.

10.80.3 Member Data Documentation

10.80.3.1 Resource∗ LLGL::ResourceViewDescriptor::resource = nullptr

Pointer to the hardware resoudce.

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

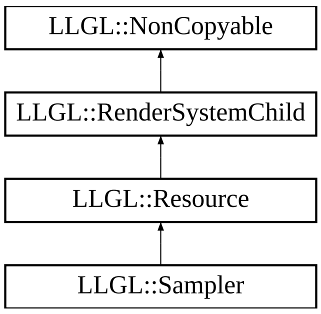
- ResourceHeapFlags.h

10.81 LLGL::Sampler Class Reference

Sampler interface.

```
#include <Sampler.h>
```

Inheritance diagram for LLGL::Sampler:



Public Member Functions

- ResourceType QueryResourceType () const override
  Returns ResourceType::Sampler.

Protected Member Functions

- Sampler ()=default

10.81.1 Detailed Description

Sampler interface.

See also

RenderSystem::CreateSampler

10.81.2 Constructor & Destructor Documentation

10.81.2.1 LLGL::Sampler::Sampler ( ) [protected], [default]

10.81.3 Member Function Documentation

10.81.3.1 ResourceType LLGL::Sampler::QueryResourceType ( ) const [override], [virtual]

Returns ResourceType::Sampler.

Implements LLGL::Resource.

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

- Sampler.h

10.82 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).

10.82.1 Detailed Description

Texture sampler descriptor structure.

10.82.2 Member Data Documentation

10.82.2.1 SamplerAddressMode LLGL::SamplerDescriptor::addressModeU = SamplerAddressMode::Repeat

Sampler address mode in U direction (also X axis). By default SamplerAddressMode::Repeat.

10.82.2.2 SamplerAddressMode LLGL::SamplerDescriptor::addressModeV = SamplerAddressMode::Repeat

Sampler address mode in V direction (also Y axis). By default SamplerAddressMode::Repeat.

10.82.2.3 SamplerAddressMode LLGL::SamplerDescriptor::addressModeW = SamplerAddressMode::Repeat

Sampler address mode in W direction (also Z axis). By default SamplerAddressMode::Repeat.

10.82.2.4 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}`

10.82.2.5 bool LLGL::SamplerDescriptor::compareEnabled = false

Specifies whether the compare operation for depth textures is to be used or not. By default false.

10.82.2.6 CompareOp LLGL::SamplerDescriptor::compareOp = CompareOp::Less

Compare operation for depth textures. By default CompareOp::Less.

10.82.2.7 SamplerFilter LLGL::SamplerDescriptor::magFilter = SamplerFilter::Linear

Magnification filter. By default SamplerFilter::Linear.

10.82.2.8 std::uint32_t LLGL::SamplerDescriptor::maxAnisotropy = 1

Maximal anisotropy in the range [1, 16].

10.82.2.9 float LLGL::SamplerDescriptor::maxLOD = 1000.0f

Upper end of the MIP-map range. Must be greater than or equal to "minLOD". By default 1000.

10.82.2.10 SamplerFilter LLGL::SamplerDescriptor::minFilter = SamplerFilter::Linear

Minification filter. By default SamplerFilter::Linear.

10.82.2.11 float LLGL::SamplerDescriptor::minLOD = 0.0f

Lower end of the MIP-map range. By default 0.

10.82.2.12 SamplerFilter LLGL::SamplerDescriptor::mipMapFilter = SamplerFilter::Linear

MIP-mapping filter. By default SamplerFilter::Linear.

10.82.2.13 float LLGL::SamplerDescriptor::mipMapLODBias = 0.0f

MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.

10.82.2.14 bool LLGL::SamplerDescriptor::mipMapping = true

Specifies whether MIP-maps are used or not. By default true.

Note

Sampling a texture object that was not created with the 'TextureFlags::GenerateMips' flag while MIP-mapping is enabled is considered undefined behavior.

See also

TextureFlags::GenerateMips
TextureDescriptor::flags

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

- SamplerFlags.h

10.83 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.

10.83.1 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

10.83.2 Constructor & Destructor Documentation

10.83.2.1 LLGL::Scissor::Scissor ( ) `[default]`

10.83.2.2 LLGL::Scissor::Scissor (const Scissor &) `[default]`

10.83.2.3 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.

10.83.2.4 LLGL::Scissor::Scissor (const Offset2D & *offset*, const Extent2D & *extent*) [inline]

Scissor constructor with offset and extent parameters.

10.83.3 Member Data Documentation

10.83.3.1 std::int32_t LLGL::Scissor::height = 0

Right-bottom height.

10.83.3.2 std::int32_t LLGL::Scissor::width = 0

Right-bottom width.

10.83.3.3 std::int32_t LLGL::Scissor::x = 0

Left-top X coordinate.

10.83.3.4 std::int32_t LLGL::Scissor::y = 0

Left-top Y coordinate.

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

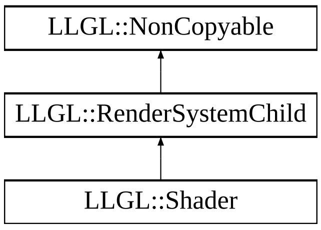
- GraphicsPipelineFlags.h

10.84 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 Disassemble (int flags=0)=0

  Disassembles the previously compiled shader byte code.
- virtual std::string QueryInfoLog ()=0

  Returns the information log after the shader compilation.
- 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)

10.84.1 Detailed Description

Shader interface.

See also

RenderSystem::CreateShader

10.84.2 Constructor & Destructor Documentation

10.84.2.1 LLGL::Shader::Shader (const ShaderType *type*) `[protected]`

10.84.3 Member Function Documentation

10.84.3.1 virtual std::string LLGL::Shader::Disassemble (int *flags* = `0`) `[pure virtual]`

Disassembles the previously compiled shader byte code.

Parameters

`in`	*flags*	Specifies optional disassemble flags. This can be a bitwise OR combination of the 'ShaderDisassembleFlags' enumeration entries. By default 0.

Returns

Disassembled assembler code or an empty string if disassembling was not possible.

Note

Only supported with: Direct3D 11, Direct3D 12.

10.84.3.2 long LLGL::Shader::GetStageFlags ( ) const

Returns the shader stage bitmask for this shader object.

See also

StageFlags

10.84.3.3 ShaderType LLGL::Shader::GetType ( ) const [inline]

Returns the type of this shader.

10.84.3.4 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 QueryInfoLog function.

See also

QueryInfoLog

10.84.3.5 virtual std::string LLGL::Shader::QueryInfoLog ( ) [pure virtual]

Returns the information log after the shader compilation.

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

- Shader.h

10.85 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) }

10.85.1 Detailed Description

Shader compilation flags enumeration.

Note

Only supported with: Direct3D 11, Direct3D 12.

10.85.2 Member Enumeration Documentation

10.85.2.1 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

10.86 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
- 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).

10.86.1 Detailed Description

Shader source and binary code descriptor structure.

See also

RenderSystem::CreateShader

10.86.2 Constructor & Destructor Documentation

10.86.2.1 LLGL::ShaderDescriptor::ShaderDescriptor ( ) `[default]`

10.86.2.2 LLGL::ShaderDescriptor::ShaderDescriptor (const ShaderDescriptor &) `[default]`

10.86.2.3 LLGL::ShaderDescriptor::ShaderDescriptor (const ShaderType *type*, const char ∗ *source*) `[inline]`

Constructor to initialize the shader descriptor with a source filename.

10.86.2.4 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.

10.86.3 Member Function Documentation

10.86.3.1 ShaderDescriptor& LLGL::ShaderDescriptor::operator= (const ShaderDescriptor &) [default]

10.86.4 Member Data Documentation

10.86.4.1 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.

10.86.4.2 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: Direct3D 11, Direct3D 12.

See also

ShaderCompileFlags

10.86.4.3 const char∗ LLGL::ShaderDescriptor::profile = nullptr

10.86.4.4 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

10.86.4.5 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

10.86.4.6 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

10.86.4.7 StreamOutput LLGL::ShaderDescriptor::streamOutput

Optional stream output descriptor for a geometry shader (or a vertex shader when used with OpenGL).

10.86.4.8 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

10.87 LLGL::ShaderDisassembleFlags Struct Reference

Shader disassemble flags enumeration.

```
#include <ShaderFlags.h>
```

Public Types

- enum { InstructionOnly = (1 << 0) }

10.87.1 Detailed Description

Shader disassemble flags enumeration.

10.87.2 Member Enumeration Documentation

10.87.2.1 anonymous enum

Enumerator

InstructionOnly Show only instructions in disassembly output.

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

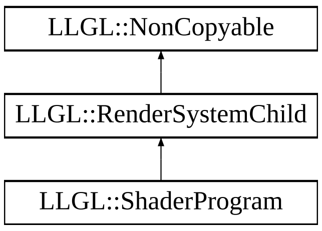
- ShaderFlags.h

10.88 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 QueryInfoLog ()=0

  Returns the information log after the shader linkage.
- virtual ShaderReflectionDescriptor QueryReflectionDesc () const =0

  Returns a descriptor of the shader pipeline layout with all required shader resources.
- virtual void BindConstantBuffer (const std::string &name, std::uint32_t bindingIndex)=0

  Binds the specified constant buffer to this shader.
- virtual void BindStorageBuffer (const std::string &name, std::uint32_t bindingIndex)=0

  Binds the specified storage buffer to this shader.
- virtual ShaderUniform ∗ LockShaderUniform ()=0

  Locks the shader uniform handler.
- virtual void UnlockShaderUniform ()=0

  Unlocks the shader uniform handler.

Protected 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 FinalizeShaderReflection (ShaderReflectionDescriptor &reflectionDesc)

  Sorts the resource views of the specified shader reflection descriptor as described in the QueryReflectionDesc 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

10.88.1 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

10.88.2 Member Enumeration Documentation

10.88.2.1 enum LLGL::ShaderProgram::LinkError [strong], [protected]

Linker error codes for internal error checking.

Enumerator

NoError

InvalidComposition

InvalidByteCode

TooManyAttachments

IncompleteAttachments

10.88.3 Member Function Documentation

10.88.3.1 virtual void LLGL::ShaderProgram::BindConstantBuffer (const std::string & *name*, std::uint32_t *bindingIndex*) [pure virtual]

Binds the specified constant buffer to this shader.

Parameters

in	*name*	Specifies the name of the constant buffer within this shader.
in	*bindingIndex*	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindConstantBuffer".

Remarks

This function is only necessary if the binding index does not match the default binding index of the constant buffer within the shader.

See also

QueryConstantBuffers
RenderContext::BindConstantBuffer

10.88.3.2 virtual void LLGL::ShaderProgram::BindStorageBuffer (const std::string & *name*, std::uint32_t *bindingIndex*) [pure virtual]

Binds the specified storage buffer to this shader.

Parameters

in	*name*	Specifies the name of the storage buffer within this shader.
in	*bindingIndex*	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindStorageBuffer".

Remarks

This function is only necessary if the binding index does not match the default binding index of the storage buffer within the shader.

See also

RenderContext::BindStorageBuffer

10.88.3.3 static void LLGL::ShaderProgram::FinalizeShaderReflection (ShaderReflectionDescriptor & *reflectionDesc*) [static], [protected]

Sorts the resource views of the specified shader reflection descriptor as described in the QueryReflectionDesc function.

See also

QueryReflectionDesc

10.88.3.4 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 QueryInfoLog function.

See also

QueryInfoLog

10.88.3.5 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).

10.88.3.6 virtual ShaderUniform∗ LLGL::ShaderProgram::LockShaderUniform ( ) [pure virtual]

Locks the shader uniform handler.

Returns

Pointer to the shader uniform handler or null if the render system does not support individual shader uniforms.

Remarks

This must be called to set individual shader uniforms.

```
if (auto myUniformHandler = myShaderProgram->LockShaderUniform()) {
    myUniformHandler->SetUniform1i("mySampler1", 0);
    myUniformHandler->SetUniform1i("mySampler2", 1);
    myUniformHandler->SetUniform4x4fv("projection", &myProjectionMatrix[0]);
    myShaderProgram->UnlockShaderUniform();
}
```

Note

Only supported with: OpenGL.

See also

UnlockShaderUniform

10.88.3.7 virtual std::string LLGL::ShaderProgram::QueryInfoLog ( ) [pure virtual]

Returns the information log after the shader linkage.

10.88.3.8 virtual ShaderReflectionDescriptor LLGL::ShaderProgram::QueryReflectionDesc ( ) const [pure virtual]

Returns a descriptor of the shader pipeline layout with all required shader resources.

Remarks

The list of resource views in the output descriptor (i.e. 'resourceViews' attribute) is always sorted in the following manner: First sorting criterion is the resource type (in ascending order), second sorting criterion is the binding slot (in ascending order). Here is an example of such a sorted list (pseudocode):

```
resourceViews[0] = { type: ResourceType::ConstantBuffer, slot: 0 }
resourceViews[1] = { type: ResourceType::ConstantBuffer, slot: 2 }
resourceViews[2] = { type: ResourceType::Texture, slot: 0 }
resourceViews[3] = { type: ResourceType::Texture, slot: 1 }
resourceViews[4] = { type: ResourceType::Texture, slot: 2 }
resourceViews[5] = { type: ResourceType::Sampler, slot: 2 }
```

See also

ShaderReflectionDescriptor::resourceViews

Exceptions

std::runtime_error	If shader reflection failed.

10.88.3.9 virtual void LLGL::ShaderProgram::UnlockShaderUniform ( ) [pure virtual]

Unlocks the shader uniform handler.

See also

LockShaderUniform

10.88.3.10 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

10.89 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.

10.89.1 Detailed Description

Descriptor structure for shader programs.

See also

RenderSystem::CreateShaderProgram
RenderSystem::CreateShader

10.89.2 Member Data Documentation

10.89.2.1 Shader∗ LLGL::ShaderProgramDescriptor::computeShader = nullptr

Specifies the compute shader.

Remarks

This shader cannot be used in conjunction with any other shaders.

10.89.2.2 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.

10.89.2.3 Shader∗ LLGL::ShaderProgramDescriptor::geometryShader = nullptr

Specifies an optional geometry shader.

Remarks

This shader may also have a stream output.

See also

ShaderDescriptor::streamOutput

10.89.2.4 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

10.89.2.5 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

10.89.2.6 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

10.89.2.7 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

10.90 LLGL::ShaderReflectionDescriptor Struct Reference

Shader reflection descriptor structure.

```
#include <ShaderProgramFlags.h>
```

Classes

- struct ResourceView
  Shader reflection resource view structure.

Public Attributes

- std::vector< VertexAttribute > vertexAttributes
  List of all vertex attributes.
- std::vector< StreamOutputAttribute > streamOutputAttributes
  List of all stream-output attributes.
- std::vector< ResourceView > resourceViews
  List of all shader reflection resource views.
- std::vector< UniformDescriptor > uniforms
  List of all uniforms.

10.90.1 Detailed Description

Shader reflection descriptor structure.

Remarks

Contains all information of resources and attributes that can be queried from a shader program.

See also

ShaderProgram::QueryReflectionDesc

10.90.2 Member Data Documentation

10.90.2.1 std::vector< ResourceView > LLGL::ShaderReflectionDescriptor::resourceViews

List of all shader reflection resource views.

10.90.2.2 std::vector< StreamOutputAttribute > LLGL::ShaderReflectionDescriptor::streamOutputAttributes

List of all stream-output attributes.

10.90.2.3 std::vector< UniformDescriptor > LLGL::ShaderReflectionDescriptor::uniforms

List of all uniforms.

Note

Only supported with: OpenGL, Vulkan.

10.90.2.4 std::vector< VertexAttribute > LLGL::ShaderReflectionDescriptor::vertexAttributes

List of all vertex attributes.

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

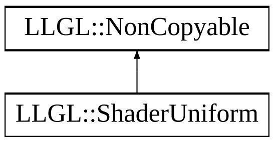
- ShaderProgramFlags.h

10.91 LLGL::ShaderUniform Class Reference

Shader uniform setter interface.

```
#include <ShaderUniform.h>
```

Inheritance diagram for LLGL::ShaderUniform:



Public Member Functions

- virtual void SetUniform1i (const UniformLocation location, int value0)=0

  Sets an integral scalar uniform.
- virtual void SetUniform2i (const UniformLocation location, int value0, int value1)=0
- virtual void SetUniform3i (const UniformLocation location, int value0, int value1, int value2)=0
- virtual void SetUniform4i (const UniformLocation location, int value0, int value1, int value2, int value3)=0
- virtual void SetUniform1f (const UniformLocation location, float value0)=0
- virtual void SetUniform2f (const UniformLocation location, float value0, float value1)=0
- virtual void SetUniform3f (const UniformLocation location, float value0, float value1, float value2)=0
- virtual void SetUniform4f (const UniformLocation location, float value0, float value1, float value2, float value3)=0
- virtual void SetUniform1iv (const UniformLocation location, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform2iv (const UniformLocation location, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform3iv (const UniformLocation location, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform4iv (const UniformLocation location, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform1fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform2fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform3fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform4fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform2x2fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform3x3fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform4x4fv (const UniformLocation location, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform1i (const char ∗name, int value0)=0
- virtual void SetUniform2i (const char ∗name, int value0, int value1)=0
- virtual void SetUniform3i (const char ∗name, int value0, int value1, int value2)=0
- virtual void SetUniform4i (const char ∗name, int value0, int value1, int value2, int value3)=0
- virtual void SetUniform1f (const char ∗name, float value0)=0
- virtual void SetUniform2f (const char ∗name, float value0, float value1)=0
- virtual void SetUniform3f (const char ∗name, float value0, float value1, float value2)=0
- virtual void SetUniform4f (const char ∗name, float value0, float value1, float value2, float value3)=0
- virtual void SetUniform1iv (const char ∗name, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform2iv (const char ∗name, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform3iv (const char ∗name, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform4iv (const char ∗name, const int ∗value, std::size_t count=1)=0
- virtual void SetUniform1fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform2fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform3fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform4fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform2x2fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform3x3fv (const char ∗name, const float ∗value, std::size_t count=1)=0
- virtual void SetUniform4x4fv (const char ∗name, const float ∗value, std::size_t count=1)=0

Additional Inherited Members

10.91.1 Detailed Description

Shader uniform setter interface.

Note

Only supported with: OpenGL.

See also

ShaderProgram::LockShaderUniform

Todo Complete documentation.

10.91.2 Member Function Documentation

10.91.2.1 virtual void LLGL::ShaderUniform::SetUniform1f (const UniformLocation *location*, float *value0*) [pure virtual]

10.91.2.2 virtual void LLGL::ShaderUniform::SetUniform1f (const char ∗ *name*, float *value0*) [pure virtual]

10.91.2.3 virtual void LLGL::ShaderUniform::SetUniform1fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.4 virtual void LLGL::ShaderUniform::SetUniform1fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.5 virtual void LLGL::ShaderUniform::SetUniform1i (const UniformLocation *location*, int *value0*) [pure virtual]

Sets an integral scalar uniform.

Remarks

This can be used to set the binding slot for samplers, like in the following GLSL example:

```
uniform sampler2D myColorSampler;
```

10.91.2.6 virtual void LLGL::ShaderUniform::SetUniform1i (const char ∗ *name*, int *value0*) [pure virtual]

10.91.2.7 virtual void LLGL::ShaderUniform::SetUniform1iv (const UniformLocation *location*, const int ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.8 virtual void LLGL::ShaderUniform::SetUniform1iv (const char ∗ *name*, const int ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.9 virtual void LLGL::ShaderUniform::SetUniform2f (const UniformLocation *location*, float *value0*, float *value1*) [pure virtual]

10.91.2.10 virtual void LLGL::ShaderUniform::SetUniform2f (const char ∗ *name*, float *value0*, float *value1*) [pure virtual]

10.91.2.11 virtual void LLGL::ShaderUniform::SetUniform2fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.12 virtual void LLGL::ShaderUniform::SetUniform2fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.13 virtual void LLGL::ShaderUniform::SetUniform2i (const UniformLocation *location*, int *value0*, int *value1*) [pure virtual]

10.91.2.14 virtual void LLGL::ShaderUniform::SetUniform2i (const char ∗ *name*, int *value0*, int *value1*) [pure virtual]

10.91.2.15 virtual void LLGL::ShaderUniform::SetUniform2iv (const UniformLocation *location*, const int ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.16 virtual void LLGL::ShaderUniform::SetUniform2iv (const char ∗ *name*, const int ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.17 virtual void LLGL::ShaderUniform::SetUniform2x2fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.18 virtual void LLGL::ShaderUniform::SetUniform2x2fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.19 virtual void LLGL::ShaderUniform::SetUniform3f (const UniformLocation *location*, float *value0*, float *value1*, float *value2*) `[pure virtual]`

10.91.2.20 virtual void LLGL::ShaderUniform::SetUniform3f (const char ∗ *name*, float *value0*, float *value1*, float *value2*) `[pure virtual]`

10.91.2.21 virtual void LLGL::ShaderUniform::SetUniform3fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.22 virtual void LLGL::ShaderUniform::SetUniform3fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.23 virtual void LLGL::ShaderUniform::SetUniform3i (const UniformLocation *location*, int *value0*, int *value1*, int *value2*) `[pure virtual]`

10.91.2.24 virtual void LLGL::ShaderUniform::SetUniform3i (const char ∗ *name*, int *value0*, int *value1*, int *value2*) `[pure virtual]`

10.91.2.25 virtual void LLGL::ShaderUniform::SetUniform3iv (const UniformLocation *location*, const int ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.26 virtual void LLGL::ShaderUniform::SetUniform3iv (const char ∗ *name*, const int ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.27 virtual void LLGL::ShaderUniform::SetUniform3x3fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.28 virtual void LLGL::ShaderUniform::SetUniform3x3fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = `1`) `[pure virtual]`

10.91.2.29 virtual void LLGL::ShaderUniform::SetUniform4f (const UniformLocation *location*, float *value0*, float *value1*, float *value2*, float *value3*) `[pure virtual]`

10.91.2.30 virtual void LLGL::ShaderUniform::SetUniform4f (const char ∗ *name*, float *value0*, float *value1*, float *value2*, float *value3*) `[pure virtual]`

10.91.2.31 virtual void LLGL::ShaderUniform::SetUniform4fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.32 virtual void LLGL::ShaderUniform::SetUniform4fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.33 virtual void LLGL::ShaderUniform::SetUniform4i (const UniformLocation *location*, int *value0*, int *value1*, int *value2*, int *value3*) [pure virtual]

10.91.2.34 virtual void LLGL::ShaderUniform::SetUniform4i (const char ∗ *name*, int *value0*, int *value1*, int *value2*, int *value3*) [pure virtual]

10.91.2.35 virtual void LLGL::ShaderUniform::SetUniform4iv (const UniformLocation *location*, const int ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.36 virtual void LLGL::ShaderUniform::SetUniform4iv (const char ∗ *name*, const int ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.37 virtual void LLGL::ShaderUniform::SetUniform4x4fv (const UniformLocation *location*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

10.91.2.38 virtual void LLGL::ShaderUniform::SetUniform4x4fv (const char ∗ *name*, const float ∗ *value*, std::size_t *count* = 1) [pure virtual]

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

- ShaderUniform.h

10.92 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.

10.92.1 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

10.92.2 Constructor & Destructor Documentation

10.92.2.1 LLGL::SrcImageDescriptor::SrcImageDescriptor ( ) [default]

10.92.2.2 LLGL::SrcImageDescriptor::SrcImageDescriptor (const SrcImageDescriptor &) [default]

10.92.2.3 LLGL::SrcImageDescriptor::SrcImageDescriptor (ImageFormat *format*, DataType *dataType*, const void ∗ *data*, std::size_t *dataSize*) [inline]

Constructor to initialize all attributes.

10.92.3 Member Data Documentation

10.92.3.1 const void∗ LLGL::SrcImageDescriptor::data = nullptr

Pointer to the read-only image data.

10.92.3.2 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.

10.92.3.3 DataType LLGL::SrcImageDescriptor::dataType = DataType::UInt8

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

10.92.3.4 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

10.93 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), StorageUsage = (1 << 6), AllTessStages = (Tess↩
  ControlStage | TessEvaluationStage),
  AllGraphicsStages = (VertexStage | AllTessStages | GeometryStage | FragmentStage), AllStages = (All↩
  GraphicsStages | ComputeStage) }

10.93.1 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 set. For the render systems, which do not support these flags, always all shader stages are affected.

10.93.2 Member Enumeration Documentation

10.93.2.1 anonymous enum

Enumerator

VertexStage Specifies the vertex shader stage.

TessControlStage Specifies the tessellation-control shader stage (also "Hull Shader").

TessEvaluationStage Specifies the tessellation-evaluation shader stage (also "Domain Shader").

GeometryStage Specifies the geometry shader stage.

FragmentStage Specifies the fragment shader stage (also "Pixel Shader").

ComputeStage Specifies the compute shader stage.

StorageUsage Specifies whether a resource is bound to the shader stages as unordered access view (UAV) instead of a read-only shader resource view (SRV).

Remarks

This can be used to bind a storage buffer (i.e. BufferType::Storage) that was created with read/write access (e.g. StorageBufferType::RWBuffer) as UAV instead of SRV.

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

10.94 LLGL::StencilDescriptor Struct Reference

Stencil state descriptor structure.

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

Public Attributes

- bool testEnabled = false

  Specifies whether the stencil test is enabled or disabled.
- StencilFaceDescriptor front

  Specifies the front face settings for the stencil test.
- StencilFaceDescriptor back

  Specifies the back face settings for the stencil test.

10.94.1 Detailed Description

Stencil state descriptor structure.

See also

GraphicsPipelineDescriptor::stencil

10.94.2 Member Data Documentation

10.94.2.1 StencilFaceDescriptor LLGL::StencilDescriptor::back

Specifies the back face settings for the stencil test.

10.94.2.2 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

10.94.2.3 bool LLGL::StencilDescriptor::testEnabled = false

Specifies whether the stencil test is enabled or disabled.

Remarks

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

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

- GraphicsPipelineFlags.h

10.95 LLGL::StencilFaceDescriptor Struct Reference

Stencil face descriptor structure.

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

Public Attributes

- StencilOp stencilFailOp = StencilOp::Keep

    Specifies the operation to take when the stencil test fails.
- StencilOp depthFailOp = StencilOp::Keep

    Specifies the operation to take when the stencil test passes but the depth test fails.
- StencilOp depthPassOp = StencilOp::Keep

    Specifies the operation to take when both the stencil test and the depth test pass.
- CompareOp compareOp = CompareOp::Less

    Specifies the stencil compare operation.
- std::uint32_t readMask = ∼0

    Specifies the portion of the depth-stencil buffer for reading stencil data. By default `0xFFFFFFFF`.
- std::uint32_t writeMask = ∼0

    Specifies the portion of the depth-stencil buffer for writing stencil data. By default `0xFFFFFFFF`.
- std::uint32_t reference = 0

    Specifies the stencil reference value.

10.95.1 Detailed Description

Stencil face descriptor structure.

See also

StencilDescriptor::front
StencilDescriptor::back

10.95.2 Member Data Documentation

10.95.2.1 CompareOp LLGL::StencilFaceDescriptor::compareOp = CompareOp::Less

Specifies the stencil compare operation.

10.95.2.2 StencilOp LLGL::StencilFaceDescriptor::depthFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test passes but the depth test fails.

10.95.2.3 StencilOp LLGL::StencilFaceDescriptor::depthPassOp = StencilOp::Keep

Specifies the operation to take when both the stencil test and the depth test pass.

10.95.2.4 std::uint32_t LLGL::StencilFaceDescriptor::readMask = ∼0

Specifies the portion of the depth-stencil buffer for reading stencil data. By default `0xFFFFFFFF`.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (i.e. `readMask & 0xFF`) of the read mask value of the front face will be used.

See also

StencilDescriptor::front

10.95.2.5 std::uint32_t LLGL::StencilFaceDescriptor::reference = 0

Specifies the stencil reference value.

Remarks

This value will be used when the stencil operation is StencilOp::Replace.

Note

For Direct3D 11 and Direct3D 12, only the stencil reference value of the front face will be used.

See also

StencilDescriptor::front

10.95.2.6 StencilOp LLGL::StencilFaceDescriptor::stencilFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test fails.

10.95.2.7 std::uint32_t LLGL::StencilFaceDescriptor::writeMask = ∼0

Specifies the portion of the depth-stencil buffer for writing stencil data. By default `0xFFFFFFFF`.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (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

10.96 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.

10.96.1 Detailed Description

Storage buffer specific descriptor structure.

10.96.2 Member Data Documentation

10.96.2.1 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

IsStorageBufferTyped

10.96.2.2 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.

10.96.2.3 std::uint32_t LLGL::BufferDescriptor::StorageBuffer::stride = 0

Specifies the stride (in bytes) of each element in a storage buffer.

Remarks

If this value is zero, the behavior of the buffer creation is undefined.

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

- BufferFlags.h

10.97 LLGL::ShaderDescriptor::StreamOutput Struct Reference

Additional descriptor for stream outputs.

```
#include <ShaderFlags.h>
```

Public Attributes

- StreamOutputFormat format
  Stream-output buffer format.

10.97.1 Detailed Description

Additional descriptor for stream outputs.

10.97.2 Member Data Documentation

10.97.2.1 StreamOutputFormat LLGL::ShaderDescriptor::StreamOutput::format

Stream-output buffer format.

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

- ShaderFlags.h

10.98 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.

10.98.1 Detailed Description

Stream-output attribute structure.

10.98.2 Constructor & Destructor Documentation

10.98.2.1 LLGL::StreamOutputAttribute::StreamOutputAttribute ( ) [default]

10.98.2.2 LLGL::StreamOutputAttribute::StreamOutputAttribute (const StreamOutputAttribute &) [default]

10.98.3 Member Function Documentation

10.98.3.1 StreamOutputAttribute& LLGL::StreamOutputAttribute::operator= (const StreamOutputAttribute &) [default]

10.98.4 Member Data Documentation

10.98.4.1 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

10.98.4.2 std::string LLGL::StreamOutputAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

10.98.4.3 std::uint8_t LLGL::StreamOutputAttribute::outputSlot = 0

Stream-output buffer output slot.

Remarks

This is used when multiple stream-output buffers are used simultaneously.

10.98.4.4 std::uint32_t LLGL::StreamOutputAttribute::semanticIndex = 0

Semantic index.

Note

Only supported with: Direct3D 11, Direct3D 12.

10.98.4.5 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.

10.98.4.6 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

10.99 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.

10.99.1 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.

10.99.2 Member Function Documentation

10.99.2.1 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.

10.99.2.2 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.

10.99.3 Member Data Documentation

10.99.3.1 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:

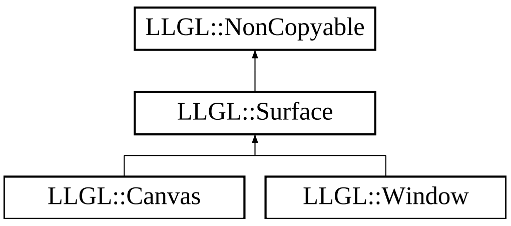
- StreamOutputFormat.h

10.100 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.

Additional Inherited Members

10.100.1 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

10.100.2 Member Function Documentation

10.100.2.1 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.

10.100.2.2 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

10.100.2.3 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';
}
```

10.100.2.4 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:

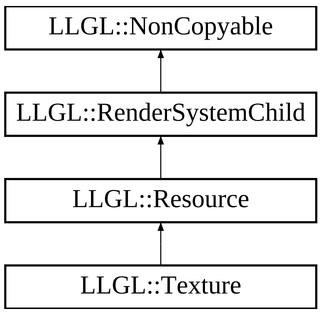
- Surface.h

10.101 LLGL::Texture Class Reference

Texture interface.

`#include <Texture.h>`

Inheritance diagram for LLGL::Texture:



Public Member Functions

- ResourceType QueryResourceType () const override

    Returns ResourceType::Texture.
- TextureType GetType () const

    Returns the type of this texture.
- virtual TextureDescriptor QueryDesc () const =0

    Queries a descriptor of this texture (including type, format, and size).
- virtual Extent3D QueryMipExtent (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)

10.101.1 Detailed Description

Texture interface.

See also

RenderSystem::CreateTexture

10.101.2 Constructor & Destructor Documentation

10.101.2.1 LLGL::Texture::Texture (const TextureType *type*) `[protected]`

10.101.3 Member Function Documentation

10.101.3.1 TextureType LLGL::Texture::GetType ( ) const `[inline]`

Returns the type of this texture.

10.101.3.2 virtual TextureDescriptor LLGL::Texture::QueryDesc ( ) const [pure virtual]

Queries a descriptor of this texture (including type, format, and size).

Remarks

This function is guaranteed to keep the currently bound textures, i.e. all previously boundd textures (e.g. using the CommandBufferExt::SetTexture function) will remain.

Note

The field TextureDescriptor::flags will always be 0, i.e. the texture flags cannot be retrived after texture creation.

See also

TextureDescriptor

10.101.3.3 virtual Extent3D LLGL::Texture::QueryMipExtent (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

This function is guaranteed to keep the currently bound textures, i.e. all previously bounds textures (e.g. using the CommandBufferExt::SetTexture function) will remain.

Note

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

RenderSystem::GenerateMips
CommandBufferExt::SetTexture

10.101.3.4 ResourceType LLGL::Texture::QueryResourceType ( ) const [override], [virtual]

Returns ResourceType::Texture.

Implements LLGL::Resource.

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

- Texture.h

10.102 LLGL::TextureDescriptor Struct Reference

Texture descriptor structure.

`#include <TextureFlags.h>`

Public Attributes

- TextureType type = TextureType::Texture1D

  Hardware texture type. By default TextureType::Texture1D.
- long flags = TextureFlags::Default

  Specifies the texture creation flags (e.g. if MIP-mapping is required). By default TextureFlags::Default.
- Format format = Format::RGBA8UNorm

  Hardware texture format. By default Format::RGBA8UNorm.
- Extent3D extent = { 1, 1, 1 }

  Texture extent. 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.

10.102.1 Detailed Description

Texture descriptor structure.

Remarks

This is used to specifiy the dimensions of a texture which is to be created.

See also

RenderSystem::CreateTexture

10.102.2 Member Data Documentation

10.102.2.1 std::uint32_t LLGL::TextureDescriptor::arrayLayers = 1

Number of array layers. By default 1.

Remarks

This can be greater than 1 for array textures and cube textures (i.e. TextureType::Texture1DArray, Texture↩ Type::Texture2DArray, TextureType::TextureCube, TextureType::TextureCubeArray, TextureType::Texture2D↩ MSArray). For cube textures, this must be a multiple of 6 (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

10.102.2.2 Extent3D LLGL::TextureDescriptor::extent = { 1, 1, 1 }

Texture extent. By default (1, 1, 1).

Remarks

The height component is only used for 2D, 3D, and Cube textures (i.e. TextureType::Texture2D, Texture↩ Type::Texture2DArray, TextureType::Texture3D, TextureType::TextureCube, TextureType::TextureCubeArray, TextureType::Texture2DMS, TextureType::Texture2DMSArray). The depth component is only used for 3D textures (i.e. TextureType::Texture3D). For cube textures, the width and height component must be equal.

See also

IsArrayTexture
IsCubeTexture

10.102.2.3 long LLGL::TextureDescriptor::flags = TextureFlags::Default

Specifies the texture creation flags (e.g. if MIP-mapping is required). By default TextureFlags::Default.

Remarks

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

See also

TextureFlags

10.102.2.4 Format LLGL::TextureDescriptor::format = Format::RGBA8UNorm

Hardware texture format. By default Format::RGBA8UNorm.

10.102.2.5 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 MIP level. This field is ignored for multi-sampled textures (i.e. TextureType::Texture2DMS, Texture↩ Type::Texture2DMSArray), since these texture types only have a single MIP-map level.

See also

NumMipLevels
RenderSystem::GenerateMips

10.102.2.6 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::Texture2DM↩ SArray). The equivalent member for graphics pipeline states is MultiSamplingDescriptor::samples.

See also

IsMultiSampleTexture

10.102.2.7 TextureType LLGL::TextureDescriptor::type = TextureType::Texture1D

Hardware texture type. By default TextureType::Texture1D.

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

- TextureFlags.h

10.103 LLGL::TextureFlags Struct Reference

Texture creation flags enumeration.

```
#include <TextureFlags.h>
```

Public Types

- enum {
  ColorAttachmentUsage = (1 << 3), DepthStencilAttachmentUsage = (1 << 4), SampleUsage = (1 << 5),
  StorageUsage = (1 << 6),
  FixedSamples = (1 << 7), Default = (ColorAttachmentUsage | SampleUsage | FixedSamples) }

10.103.1 Detailed Description

Texture creation flags enumeration.

See also

TextureDescriptor::flags

10.103.2 Member Enumeration Documentation

10.103.2.1 anonymous enum

Enumerator

ColorAttachmentUsage Texture can be used as render target color attachment.

Remarks

This is part of the default flags.

Note

This cannot be used together with the TextureFlags::DepthStencilAttachmentUsage flag.

See also

AttachmentDescriptor::texture
AttachmentType::Color

DepthStencilAttachmentUsage Texture can be used as render target depth-stencil attachment.

Note

This cannot be used together with the TextureFlags::ColorAttachmentUsage flag.

See also

AttachmentDescriptor::texture
AttachmentType::DepthStencil

SampleUsage Texture can be used for sampling (e.g. "sampler2D" in GLSL, or "Texture2D" in HLSL).

Remarks

This is part of the default flags.

StorageUsage Texture can be used as storage texture (e.g. "image2D" in GLSL, or "RWTexture2D" in HL↩
SL).

FixedSamples Multi-sampled texture has fixed sample locations.

Remarks

This can only be used with multi-sampled textures (i.e. TextureType::Texture2DMS, TextureType↩
::Texture2DMSArray).

See also

TextureType

Default Default texture flags are `(ColorAttachmentUsage | SampleUsage | Fixed↩ Samples)`.

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

- TextureFlags.h

10.104 LLGL::TextureRegion Struct Reference

Texture region structure.

```
#include <TextureFlags.h>
```

Public Attributes

- 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.
- Offset3D offset = { 0, 0, 0 }

  Sub-texture offset. By default (0, 0, 0).
- Extent3D extent = { 1, 1, 1 }

  Sub-texture extent. By default (1, 1, 1).

10.104.1 Detailed Description

Texture region structure.

Remarks

This is used to write (or partially write) the image data of a texture MIP-map level.

See also

RenderSystem::WriteTexture

10.104.2 Member Data Documentation

10.104.2.1 Extent3D LLGL::TextureRegion::extent = { 1, 1, 1 }

Sub-texture extent. By default (1, 1, 1).

Remarks

For array textures, the depth component specifies the number of array layers (for 1D-array textures it's the height component). For cube textures, the depthj component specifies the number of array layers and cube faces (where each cube has 6 faces).

10.104.2.2 std::uint32_t LLGL::TextureRegion::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.

10.104.2.3 Offset3D LLGL::TextureRegion::offset = { 0, 0, 0 }

Sub-texture offset. By default (0, 0, 0).

Remarks

For array textures, the Z component specifies the array layer. For cube textures, the Z component specifies the array layer and cube face offset (for 1D-array textures it's the Y component). The layer offset for the respective cube faces is described at the TextureDescriptor::arrayLayers member. Negative values of this member are not allowed and result in undefined behavior.

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

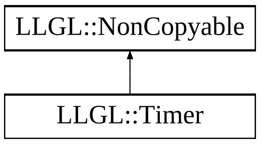
- TextureFlags.h

10.105 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

10.105.1 Detailed Description

Interface for a Timer class.

Remarks

This basic class is also designed as interface, since the native timer is platform specific.

10.105.2 Member Function Documentation

10.105.2.1 static std::unique_ptr<Timer> LLGL::Timer::Create ( ) [static]

Creates a platform specific timer object.

10.105.2.2 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

10.105.2.3 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).

10.105.2.4 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

10.105.2.5 void LLGL::Timer::MeasureTime ( )

Measures the time (elapsed time, and frame count) for each frame.

See also

GetDeltaTime

10.105.2.6 virtual void LLGL::Timer::Start ( ) `[pure virtual]`

Starts the timer.

10.105.2.7 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

10.106 LLGL::UniformDescriptor Struct Reference

Shader uniform descriptor structure.

`#include <ShaderUniformFlags.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.

10.106.1 Detailed Description

Shader uniform descriptor structure.

10.106.2 Member Data Documentation

10.106.2.1 UniformLocation LLGL::UniformDescriptor::location = 0

Internal location of the uniform within a shader program.

10.106.2.2 std::string LLGL::UniformDescriptor::name

Name of the uniform inside the shader.

10.106.2.3 std::uint32_t LLGL::UniformDescriptor::size = 0

Array size of the uniform.

10.106.2.4 UniformType LLGL::UniformDescriptor::type = UniformType::Undefined

Data type of the uniform. By default UniformType::Undefined.

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

- ShaderUniformFlags.h

10.107 LLGL::UninitializeTag Struct Reference

Common uninitialize tag.

```
#include <Tags.h>
```

10.107.1 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

10.108 LLGL::VertexAttribute Struct Reference

Vertex attribute structure.

```
#include <VertexAttribute.h>
```

Public Member Functions

- VertexAttribute ()=default
- VertexAttribute (const VertexAttribute &)=default
- VertexAttribute & operator= (const VertexAttribute &)=default
- VertexAttribute (const std::string &name, const 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.

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).

10.108.1 Detailed Description

Vertex attribute structure.

See also

VertexFormat

10.108.2 Constructor & Destructor Documentation

10.108.2.1 LLGL::VertexAttribute::VertexAttribute ( ) `[default]`

10.108.2.2 LLGL::VertexAttribute::VertexAttribute (const VertexAttribute &) `[default]`

10.108.2.3 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

10.108.2.4 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

10.108.3 Member Function Documentation

10.108.3.1 std::uint32_t LLGL::VertexAttribute::GetSize ( ) const

Returns the size (in bytes) which is required for this vertex attribute.

Returns

The format bit size converted to byte size: `FormatBitSize(format) / 8`.

10.108.3.2 VertexAttribute& LLGL::VertexAttribute::operator= (const VertexAttribute &) [default]

10.108.4 Member Data Documentation

10.108.4.1 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:

```
myVertexFormat.AppendAttribute({ "myMatrix", 0, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix", 1, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix", 2, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix", 3, LLGL::Format::RGBA32Float });
```

10.108.4.2 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.

10.108.4.3 std::string LLGL::VertexAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

10.108.4.4 std::uint32_t LLGL::VertexAttribute::offset = 0

Byte offset within each vertex and each buffer. By default 0.

10.108.4.5 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.

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

- VertexAttribute.h

10.109 LLGL::BufferDescriptor::VertexBuffer Struct Reference

Vertex buffer specific descriptor structure.

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

Public Attributes

- VertexFormat format

  Specifies the vertex format layout.

10.109.1 Detailed Description

Vertex buffer specific descriptor structure.

10.109.2 Member Data Documentation

10.109.2.1 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

10.110 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.

10.110.1 Detailed Description

Vertex format structure.

Remarks

A vertex format is required to describe how the vertex attributes are supported inside a vertex buffer.

See also

BufferDescriptor::VertexBuffer::format
ShaderProgram::BuildInputLayout

10.110.2 Member Function Documentation

10.110.2.1 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 `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 will always overwrite the `offset` and `inputSlot` members before the attribute is appended to this vertex format. The `inputSlot` member will be set to the input slot value of the previous vertex attribute and is increased by one, if the new offset of the new vertex attribute is less than the offset plus size of the previous vertex attribute.

Exceptions

std::invalid_argument	If `attrib.components` is neither 1, 2, 3, nor 4.

See also

VertexAttribute::offset
VertexAttribute::inputSlot
Constants::ignoreOffset

10.110.3 Member Data Documentation

10.110.3.1 std::vector< VertexAttribute > LLGL::VertexFormat::attributes

Specifies the list of vertex attributes.

Remarks

Use "AppendAttribute" or "AppendAttributes" to append new attributes.

10.110.3.2 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.

10.110.3.3 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

10.111 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.

10.111.1 Detailed Description

Video adapter descriptor structure.

Remarks

A video adapter determines the output capabilities of a GPU.

Todo Currently unused in the interface.

10.111.2 Member Data Documentation

10.111.2.1 std::wstring LLGL::VideoAdapterDescriptor::name

Hardware adapter name (name of the GPU).

10.111.2.2 std::vector< VideoOutputDescriptor > LLGL::VideoAdapterDescriptor::outputs

List of all adapter output descriptors.

10.111.2.3 std::string LLGL::VideoAdapterDescriptor::vendor

Vendor name (e.g. "NVIDIA Corporation", "Advanced Micro Devices, Inc." etc.).

10.111.2.4 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

10.112 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).

10.112.1 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

10.112.2 Member Data Documentation

10.112.2.1 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::QueryColorFormat function.

See also

RenderContext::QueryColorFormat

10.112.2.2 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::Query←DepthStencilFormat function.

See also

RenderContext::QueryDepthStencilFormat

10.112.2.3 bool LLGL::VideoModeDescriptor::fullscreen = false

Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

10.112.2.4 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

10.112.2.5 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::↩
QueryDepthStencilFormat function.

See also

RenderContext::QueryDepthStencilFormat

10.112.2.6 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

10.113 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.

10.113.1 Detailed Description

Video output structure.

See also

VideoAdapterDescriptor::outputs

Todo Currently unused in the interface.

10.113.2 Member Data Documentation

10.113.2.1 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

10.114 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.

10.114.1 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

10.114.2 Constructor & Destructor Documentation

10.114.2.1 LLGL::Viewport::Viewport ( ) `[default]`

10.114.2.2 LLGL::Viewport::Viewport (const Viewport &) `[default]`

10.114.2.3 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*) `[inline]`

Viewport constructor with default depth range of [0, 1].

10.114.2.4 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*, float *minDepth*, float *maxDepth*) `[inline]`

Viewport constructor with parameters for all attributes.

10.114.2.5 LLGL::Viewport::Viewport (const Extent2D & *extent*) `[inline]`

Viewport constructor with extent and default depth range of [0, 1].

10.114.2.6 LLGL::Viewport::Viewport (const Extent2D & *extent*, float *minDepth*, float *maxDepth*) `[inline]`

Viewport constructor with extent and explicit depth range.

10.114.2.7 LLGL::Viewport::Viewport (const Offset2D & *offset*, const Extent2D & *extent*) `[inline]`

Viewport constructor with offset, extent, and default depth range of [0, 1].

10.114.2.8 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.

10.114.3 Member Data Documentation

10.114.3.1 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.

10.114.3.2 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.

10.114.3.3 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.

10.114.3.4 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.

10.114.3.5 float LLGL::Viewport::x = 0.0f

Y coordinate of the left-top origin. By default 0.0.

10.114.3.6 float LLGL::Viewport::y = 0.0f

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

- GraphicsPipelineFlags.h

10.115 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.

10.115.1 Detailed Description

Vertical-synchronization (Vsync) descriptor structure.

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

10.115.2 Member Data Documentation

10.115.2.1 bool LLGL::VsyncDescriptor::enabled = false

Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.

10.115.2.2 std::uint32_t LLGL::VsyncDescriptor::interval = 1

Synchronisation interval. Can be 1, 2, 3, or 4.

Remarks

If Vsync is disabled, this value is implicitly zero.

10.115.2.3 std::uint32_t LLGL::VsyncDescriptor::refreshRate = 60

Refresh rate (in Hz). By default 60.

Note

Only supported with: Direct3D 11, Direct3D 12.

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

- RenderContextFlags.h

10.116 LLGL::VulkanRendererConfiguration Struct Reference

Structure for a Vulkan renderer specific configuration.

```
#include <RenderSystemFlags.h>
```

Public Attributes

- ApplicationDescriptor application

  Application descriptor used when a Vulkan debug or validation layer is enabled.
- 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.

10.116.1 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.

10.116.2 Member Data Documentation

10.116.2.1 ApplicationDescriptor LLGL::VulkanRendererConfiguration::application

Application descriptor used when a Vulkan debug or validation layer is enabled.

See also

ApplicationDescriptor

10.116.2.2 std::uint64_t LLGL::VulkanRendererConfiguration::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.

10.116.2.3 bool LLGL::VulkanRendererConfiguration::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.

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

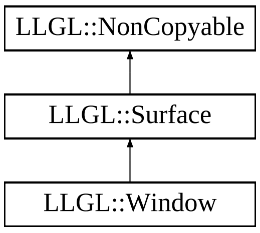
- RenderSystemFlags.h

10.117 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.
- 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.
- bool AdaptForVideoMode (VideoModeDescriptor &videoModeDesc) override

  Adapts the window for the specified video mode.
- bool ProcessEvents ()

  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 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 PostLoseFocus ()

  Posts a 'LoseFocus' event to all event listeners.
- void PostQuit ()

  Posts a 'Quit' 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.

10.117.1 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

10.117.2 Member Function Documentation

10.117.2.1 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.

10.117.2.2 void LLGL::Window::AddEventListener (const std::shared_ptr< **EventListener** > & *eventListener*)

Adds the specified event listener to this window.

10.117.2.3 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

10.117.2.4 const WindowBehavior& LLGL::Window::GetBehavior ( ) const [inline]

Returns the window heavior.

10.117.2.5 virtual WindowDescriptor LLGL::Window::GetDesc ( ) const [pure virtual]

Queries a window descriptor, which describes the attributes of this window.

10.117.2.6 virtual Offset2D LLGL::Window::GetPosition ( ) const [pure virtual]

Returns the window position relative to its parent.

10.117.2.7 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.

10.117.2.8 virtual std::wstring LLGL::Window::GetTitle ( ) const [pure virtual]

Returns the window title as UTF16 string.

10.117.2.9 virtual bool LLGL::Window::HasFocus ( ) const [virtual]

Returns true if this window has the keyboard focus.

10.117.2.10 virtual bool LLGL::Window::IsShown ( ) const [pure virtual]

Returns true if this window is visible.

10.117.2.11 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

10.117.2.12 void LLGL::Window::PostChar (wchar_t *chr*)

See also

PostKeyDown

10.117.2.13 void LLGL::Window::PostDoubleClick (Key *keyCode*)

See also

PostKeyDown

10.117.2.14 void LLGL::Window::PostGetFocus ( )

Posts a 'GetFocus' event to all event listeners.

10.117.2.15 void LLGL::Window::PostGlobalMotion (const Offset2D & *motion*)

See also

PostKeyDown

10.117.2.16 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

10.117.2.17 void LLGL::Window::PostKeyUp (Key *keyCode*)

See also

PostKeyDown

10.117.2.18 void LLGL::Window::PostLocalMotion (const Offset2D & *position*)

See also

PostKeyDown

10.117.2.19 void LLGL::Window::PostLoseFocus ( )

Posts a 'LoseFocus' event to all event listeners.

10.117.2.20 void LLGL::Window::PostQuit ( )

Posts a 'Quit' event to all event listeners.

Remarks

If at least one event listener returns false within the "OnQuit" callback, the window will not quit. If all event listener return true within the "OnQuit" callback, "ProcessEvents" will returns false from now on.

See also

EventListener::OnQuit
ProcessEvents

10.117.2.21 void LLGL::Window::PostResize (const Extent2D & *clientAreaSize*)

See also

PostKeyDown

10.117.2.22 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.

10.117.2.23 void LLGL::Window::PostWheelMotion (int *motion*)

See also

PostKeyDown

10.117.2.24 bool LLGL::Window::ProcessEvents ( )

Processes the events for this window (i.e. mouse movement, key presses etc.).

Returns

Once the "PostQuit" function was called on this window object, this function returns false. This will happend, when the user clicks on the close button.

10.117.2.25 void LLGL::Window::RemoveEventListener (const EventListener ∗ *eventListener*)

Removes the specified event listener from this window.

10.117.2.26 virtual void LLGL::Window::SetBehavior (const WindowBehavior & *behavior*) [virtual]

Sets the new window behavior.

10.117.2.27 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

10.117.2.28 virtual void LLGL::Window::SetPosition (const Offset2D & *position*) [pure virtual]

Sets the window position relative to its parent.

10.117.2.29 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.

10.117.2.30 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.

10.117.2.31 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

10.118 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.

10.118.1 Detailed Description

Window behavior structure.

See also

Window::SetBehavior

10.118.2 Member Data Documentation

10.118.2.1 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.

10.118.2.2 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

10.119 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.

10.119.1 Detailed Description

Window descriptor structure.

10.119.2 Member Data Documentation

10.119.2.1 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.

10.119.2.2 bool LLGL::WindowDescriptor::borderless = false

Specifies whether the window is borderless. This is required for a fullscreen render context. By default false.

10.119.2.3 bool LLGL::WindowDescriptor::centered = false

Specifies whether the window is centered within the desktop screen. By default false.

10.119.2.4 Offset2D LLGL::WindowDescriptor::position

Window position (relative to the client area).

10.119.2.5 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.

10.119.2.6 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

10.119.2.7 Extent2D LLGL::WindowDescriptor::size

Window size (this should be the client area size).

10.119.2.8 std::wstring LLGL::WindowDescriptor::title

Window title as unicode string.

10.119.2.9 bool LLGL::WindowDescriptor::visible = false

Specifies whether the window is visible at creation time. By default false.

10.119.2.10 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 "VertexFormat.h"
#include "IndexFormat.h"
#include "RenderSystemFlags.h"
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::BufferFlags

  Buffer creation flags enumeration.
- 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::BufferType {
  LLGL::BufferType::Vertex, LLGL::BufferType::Index, LLGL::BufferType::Constant, LLGL::BufferType::Storage,
  LLGL::BufferType::StreamOutput }

  Hardware buffer type enumeration.
- enum LLGL::StorageBufferType {
  LLGL::StorageBufferType::Undefined, LLGL::StorageBufferType::Buffer, LLGL::StorageBufferType::↩
  StructuredBuffer, LLGL::StorageBufferType::ByteAddressBuffer,
  LLGL::StorageBufferType::RWBuffer, LLGL::StorageBufferType::RWStructuredBuffer, LLGL::Storage↩
  BufferType::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.

Namespaces

- LLGL

11.10 CommandBufferExt.h File Reference

```
#include "CommandBuffer.h"
#include "ForwardDecls.h"
```

Classes

- class LLGL::CommandBufferExt

  Extended command buffer interface with dynamic state access for shader resources (i.e. Constant Buffers, Storage Buffers, Textures, and Samplers).

Namespaces

- LLGL

11.11 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.

11.12 CommandQueue.h File Reference

```
#include "RenderSystemChild.h"
#include "ForwardDecls.h"
#include <cstdint>
#include <cstddef>
```

Classes

- class LLGL::CommandQueue

  Command queue interface.

Namespaces

- LLGL

11.13 ComputePipeline.h File Reference

```
#include "RenderSystemChild.h"
#include "ComputePipelineFlags.h"
```

Classes

- class LLGL::ComputePipeline

  Compute pipeline interface.

Namespaces

- LLGL

11.14 ComputePipelineFlags.h File Reference

Classes

- struct LLGL::ComputePipelineDescriptor

  Compute pipeline descriptor structure.

Namespaces

- LLGL

11.15 Constants.h File Reference

```
#include <cstddef>
#include <cstdint>
```

Namespaces

- LLGL
- LLGL::Constants

  Namespace with all constants used as default arguments.

11.16 Display.h File Reference

```
#include "NonCopyable.h"
#include "DisplayFlags.h"
#include <vector>
#include <memory>
```

Classes

- class LLGL::Display

    Display interface to query the attributes of all connected displays/monitors.

Namespaces

- LLGL

11.17 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.

11.18 Export.h File Reference

Macros

- #define LLGL_EXPORT

11.18.1 Macro Definition Documentation

11.18.1.1 #define LLGL_EXPORT

11.19 Fence.h File Reference

```
#include "RenderSystemChild.h"
```

Classes

- class LLGL::Fence

  Fence interface for CPU/GPU synchronization.

Namespaces

- LLGL

11.20 Format.h File Reference

```
#include "Export.h"
#include <cstdint>
```

Namespaces

- LLGL

Enumerations

- enum LLGL::Format {
  LLGL::Format::Undefined, 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::RG8UNorm, LLGL::Format::RG8SNorm, LLGL::Format::RG8UInt,
  LLGL::Format::RG8SInt, LLGL::Format::RG16UNorm, LLGL::Format::RG16SNorm, LLGL::Format::RG16↩
  UInt,
  LLGL::Format::RG16SInt, LLGL::Format::RG16Float, LLGL::Format::RG32UInt, LLGL::Format::RG32SInt,
  LLGL::Format::RG32Float, LLGL::Format::RGB8UNorm, LLGL::Format::RGB8SNorm, LLGL::Format::RG↩
  B8UInt,
  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::RGBA8UNorm, LLGL::Format::RGBA8SNorm, LLGL::Format↩
  ::RGBA8UInt,
  LLGL::Format::RGBA8SInt, LLGL::Format::RGBA16UNorm, LLGL::Format::RGBA16SNorm, LLGL::↩

Format::RGBA16UInt,
LLGL::Format::RGBA16SInt, LLGL::Format::RGBA16Float, LLGL::Format::RGBA32UInt, LLGL::Format::↩
RGBA32SInt,
LLGL::Format::RGBA32Float, LLGL::Format::R64Float, LLGL::Format::RG64Float, LLGL::Format::RGB64↩
Float,
LLGL::Format::RGBA64Float, LLGL::Format::BGRA8UNorm, LLGL::Format::BGRA8SNorm, LLGL::↩
Format::BGRA8UInt,
LLGL::Format::BGRA8SInt, LLGL::Format::BGRA8sRGB, LLGL::Format::D16UNorm, LLGL::Format::D24↩
UNormS8UInt,
LLGL::Format::D32Float, LLGL::Format::D32FloatS8X24UInt, LLGL::Format::BC1RGB, LLGL::Format::B↩
C1RGBA,
LLGL::Format::BC2RGBA, LLGL::Format::BC3RGBA }

Hardware vector and pixel format enumeration.

- enum LLGL::DataType {
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 }

  Renderer data types enumeration.

Functions

- LLGL_EXPORT std::uint32_t LLGL::FormatBitSize (const Format format)

  Returns the bit size of the specified hardware format.
- LLGL_EXPORT bool LLGL::SplitFormat (const Format format, DataType &dataType, std::uint32_t &components)

  Splits the specified hardware format into a data type and the number of components.
- 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.

11.21 ForwardDecls.h File Reference

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::Dec↩ Wrap }

  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::Inv↵
DstColor,
LLGL::BlendOp::DstAlpha, LLGL::BlendOp::InvDstAlpha, LLGL::BlendOp::SrcAlphaSaturate, LLGL::Blend↵
Op::BlendFactor,
LLGL::BlendOp::InvBlendFactor, LLGL::BlendOp::Src1Color, LLGL::BlendOp::InvSrc1Color, LLGL::Blend↵
Op::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::Polygon↵
Mode::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 primitive↵
Topology)

  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.

Enumerations

- enum LLGL::ImageFormat {
  LLGL::ImageFormat::R, LLGL::ImageFormat::RG, LLGL::ImageFormat::RGB, LLGL::ImageFormat::BGR,
  LLGL::ImageFormat::RGBA, LLGL::ImageFormat::BGRA, LLGL::ImageFormat::ARGB, LLGL::Image↩
  Format::ABGR,
  LLGL::ImageFormat::Depth, LLGL::ImageFormat::DepthStencil, LLGL::ImageFormat::CompressedRGB,
  LLGL::ImageFormat::CompressedRGBA }

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

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::FindSuitableImageFormat (const Format format, ImageFormat &imageFormat, DataType &dataType)

  Finds a suitable image format for the specified texture hardware format.
- 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 IndexFormat.h File Reference

```
#include "Export.h"
#include "ImageFlags.h"
```

Classes

- class LLGL::IndexFormat

  Index buffer format class.

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 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

11.29 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.

11.30 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

11.31 LLGL.h File Reference

```
#include "Version.h"
#include "Window.h"
#include "Canvas.h"
#include "Display.h"
#include "Input.h"
#include "Timer.h"
#include "ColorRGB.h"
#include "ColorRGBA.h"
#include "RenderSystem.h"
#include "Log.h"
```

11.32 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 ∗user←Data=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.

11.33 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

11.34 NativeHandle.h File Reference

```
#include "Platform.h"
```

11.35 NonCopyable.h File Reference

```
#include "Export.h"
```

Classes

- class LLGL::NonCopyable

  Base class for all interfaces in LLGL.

Namespaces

- LLGL

11.36 PipelineLayout.h File Reference

```
#include "RenderSystemChild.h"
```

Classes

- class LLGL::PipelineLayout

  Pipeline layout interface.

Namespaces

- LLGL

11.37 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

11.38 Platform.h File Reference

11.39 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

11.40 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::Any↩
  SamplesPassedConservative, LLGL::QueryType::TimeElapsed,
  LLGL::QueryType::StreamOutPrimitivesWritten, LLGL::QueryType::StreamOutOverflow, LLGL::QueryType↩
  ::PipelineStatistics }

  Query type enumeration.

11.41 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 <map>
```

Classes

- class LLGL::RenderContext

    Render context interface.

Namespaces

- LLGL

11.42 RenderContextFlags.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "GraphicsPipelineFlags.h"
#include <functional>
#include <cstdint>
```

Classes

- struct LLGL::VsyncDescriptor

    Vertical-synchronization (Vsync) descriptor structure.
- struct LLGL::VideoModeDescriptor

    Video mode descriptor structure.
- struct LLGL::ProfileOpenGLDescriptor

    OpenGL profile descriptor structure.
- struct LLGL::RenderContextDescriptor

    Render context descriptor structure.

Namespaces

- LLGL

Typedefs

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

    Debug callback function interface.

Enumerations

- enum LLGL::OpenGLContextProfile { LLGL::OpenGLContextProfile::CompatibilityProfile, LLGL::OpenGL↩ ContextProfile::CoreProfile, LLGL::OpenGLContextProfile::ESProfile }

    OpenGL context profile enumeration.

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.

11.43 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 }

  Rendering debugger warning types enumeration.

11.44 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

11.45 RenderPass.h File Reference

```
#include "RenderSystemChild.h"
```

Classes

- class LLGL::RenderPass

    Render pass interface.

Namespaces

- LLGL

11.46 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.

11.47 RenderSystem.h File Reference

```
#include "NonCopyable.h"
#include "RenderContext.h"
#include "CommandQueue.h"
#include "CommandBufferExt.h"
#include "RenderSystemFlags.h"
#include "RenderingProfiler.h"
#include "RenderingDebugger.h"
#include "Buffer.h"
#include "BufferArray.h"
#include "Texture.h"
#include "Sampler.h"
#include "ResourceHeap.h"
#include "RenderPass.h"
#include "RenderPassFlags.h"
#include "RenderTarget.h"
#include "Shader.h"
#include "ShaderProgram.h"
#include "PipelineLayout.h"
#include "GraphicsPipeline.h"
#include "ComputePipeline.h"
#include "QueryHeap.h"
#include "Fence.h"
#include <string>
#include <memory>
#include <vector>
#include <cstdint>
```

Classes

- class LLGL::RenderSystem

  Render system interface.

Namespaces

- LLGL

11.48 RenderSystemChild.h File Reference

```
#include "NonCopyable.h"
```

Classes

- class LLGL::RenderSystemChild

  Base class for all interfaces whoes instances are owned by the RenderSystem.

Namespaces

- LLGL

11.49 RenderSystemFlags.h File Reference

```
#include "Export.h"
#include "CommandBufferFlags.h"
#include "TextureFlags.h"
#include "Constants.h"
#include <cstddef>
#include <cstdint>
#include <string>
#include <vector>
#include <functional>
```

Classes

- struct LLGL::ImageInitialization

  Structure of image initialization for textures without initial image data.
- struct LLGL::RenderSystemConfiguration

  Render system configuration structure.
- struct LLGL::RendererID

  Renderer identification number enumeration.
- struct LLGL::RendererInfo

  Renderer basic information structure.
- struct LLGL::ApplicationDescriptor

  Application descriptor structure.
- struct LLGL::VulkanRendererConfiguration

  Structure for a Vulkan renderer specific configuration.
- 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::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), L←
  LGL::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), L←
  LGL::ShadingLanguage::Metal_1_1 = (0x40000 | 110), LLGL::ShadingLanguage::Metal_1_2 = (0x40000 | 120),
  LLGL::ShadingLanguage::SPIRV = (0x50000), LLGL::ShadingLanguage::SPIRV_100 = (0x50000 | 100), L←
  LGL::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::CPU←
  Access::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.

11.50 RenderTarget.h File Reference

```
#include "RenderSystemChild.h"
#include "RenderTargetFlags.h"
#include "Types.h"
```

Classes

- class LLGL::RenderTarget

  Render target interface.

Namespaces

- LLGL

11.51 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.

11.52 Resource.h File Reference

```
#include "RenderSystemChild.h"
#include "ResourceFlags.h"
```

Classes

- class LLGL::Resource

  Base class for all hardware resource interfaces.

Namespaces

- LLGL

11.53 ResourceFlags.h File Reference

Namespaces

- LLGL

Enumerations

- enum LLGL::ResourceType {
  LLGL::ResourceType::Undefined, LLGL::ResourceType::VertexBuffer, LLGL::ResourceType::IndexBuffer, LLGL::ResourceType::ConstantBuffer,
  LLGL::ResourceType::StorageBuffer, LLGL::ResourceType::StreamOutputBuffer, LLGL::ResourceType::↩
  Texture, LLGL::ResourceType::Sampler }

  Hardware resource type enumeration.

11.54 ResourceHeap.h File Reference

```
#include "RenderSystemChild.h"
#include "ResourceHeapFlags.h"
```

Classes

- class LLGL::ResourceHeap

  Resource heap interface.

Namespaces

- LLGL

11.55 ResourceHeapFlags.h File Reference

```
#include "Export.h"
#include <vector>
```

Classes

- struct LLGL::ResourceViewDescriptor

  Resource view descriptor structure.
- struct LLGL::ResourceHeapDescriptor

  Resource heap descriptor structure.

Namespaces

- LLGL

11.56 Sampler.h File Reference

```
#include "Resource.h"
#include "SamplerFlags.h"
```

Classes

- class LLGL::Sampler

  Sampler interface.

Namespaces

- LLGL

11.57 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.

11.58 Shader.h File Reference

```
#include "RenderSystemChild.h"
#include "ShaderFlags.h"
```

Classes

- class LLGL::Shader

  Shader interface.

Namespaces

- LLGL

11.59 ShaderFlags.h File Reference

```
#include "Export.h"
#include "StreamOutputFormat.h"
#include <cstddef>
```

Classes

- struct LLGL::ShaderCompileFlags

  Shader compilation flags enumeration.
- struct LLGL::ShaderDisassembleFlags

  Shader disassemble flags enumeration.
- struct LLGL::StageFlags

  Shader stage flags enumeration.
- 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.

11.60 ShaderProgram.h File Reference

```
#include "RenderSystemChild.h"
#include "ShaderProgramFlags.h"
#include "ShaderUniform.h"
```

Classes

- class LLGL::ShaderProgram

    Shader program interface.

Namespaces

- LLGL

11.61 ShaderProgramFlags.h File Reference

```
#include "ForwardDecls.h"
#include "VertexFormat.h"
#include "StreamOutputFormat.h"
#include "ShaderUniformFlags.h"
#include "ResourceFlags.h"
#include "BufferFlags.h"
#include <vector>
#include <string>
```

Classes

- struct LLGL::ShaderProgramDescriptor

    Descriptor structure for shader programs.
- struct LLGL::ShaderReflectionDescriptor

    Shader reflection descriptor structure.
- struct LLGL::ShaderReflectionDescriptor::ResourceView

    Shader reflection resource view structure.

Namespaces

- LLGL

11.62 ShaderUniform.h File Reference

```
#include "NonCopyable.h"
#include "ShaderUniformFlags.h"
```

Classes

- class LLGL::ShaderUniform

    Shader uniform setter interface.

Namespaces

- LLGL

11.63 ShaderUniformFlags.h File Reference

```
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::UniformDescriptor

    Shader uniform descriptor 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::Float3x3, LLGL::Uniform↩Type::Float4x4,
  LLGL::UniformType::Float2x3, LLGL::UniformType::Float2x4, LLGL::UniformType::Float3x2, LLGL::↩UniformType::Float3x4,
  LLGL::UniformType::Float4x2, LLGL::UniformType::Float4x3, LLGL::UniformType::Double2x2, LLGL::↩UniformType::Double3x3,
  LLGL::UniformType::Double4x4, LLGL::UniformType::Double2x3, LLGL::UniformType::Double2x4, LLGL::↩UniformType::Double3x2,
  LLGL::UniformType::Double3x4, LLGL::UniformType::Double4x2, LLGL::UniformType::Double4x3, LLGL::↩UniformType::Sampler,
  LLGL::UniformType::Image, LLGL::UniformType::AtomicCounter }

  Shader uniform type enumeration.

11.64 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)

11.65 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

11.66 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.

11.67 Surface.h File Reference

```
#include "NonCopyable.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

11.68 Tags.h File Reference

Classes

- struct LLGL::UninitializeTag

  Common uninitialize tag.

Namespaces

- LLGL

11.69 Texture.h File Reference

```
#include "Resource.h"
#include "Types.h"
#include "TextureFlags.h"
#include <cstdint>
```

Classes

- class LLGL::Texture

  Texture interface.

Namespaces

- LLGL

11.70 TextureFlags.h File Reference

```
#include "Export.h"
#include "Types.h"
#include "Format.h"
#include <cstddef>
#include <cstdint>
```

Classes

- struct LLGL::TextureFlags

  Texture creation flags enumeration.
- struct LLGL::TextureDescriptor

  Texture descriptor structure.
- struct LLGL::TextureRegion

  Texture region 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.

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.

11.71 Timer.h File Reference

```
#include "NonCopyable.h"
#include <memory>
#include <cstdint>
```

Classes

- class LLGL::Timer

  Interface for a Timer class.

Namespaces

- LLGL

11.72 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'.

11.73 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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

  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 flags=TextureFlags::Default)

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

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t size, const IndexFormat &indexFormat, long flags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t size, long flags=BufferFlags::↩
DynamicUsage)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long flags=BufferFlags::MapReadAccess|BufferFlags::MapWriteAccess)

  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 ShaderReflectionDescriptor &reflectionDesc)

    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 &render←TargetDesc)

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

11.74 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)").

11.75 VertexAttribute.h File Reference

```
#include "Export.h"
#include "Format.h"
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::VertexAttribute

    Vertex 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).

11.76 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

11.77 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

11.78 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

11.79 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

11.80 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

∼EventListener
    LLGL::Canvas::EventListener, 170
    LLGL::Window::EventListener, 172
∼NonCopyable
    LLGL::NonCopyable, 213
∼RenderingDebugger
    LLGL::RenderingDebugger, 240

A
    LLGL, 75
a
    LLGL::Color< T, 4u >, 134
ABGR
    LLGL, 74
ANDInverted
    LLGL, 78
ANDReverse
    LLGL, 78
AND
    LLGL, 78
ARGB
    LLGL, 74
acceptDropFiles
    LLGL::WindowDescriptor, 361
Accumulate
    LLGL::FrameProfile, 179
    LLGL::RenderingProfiler, 251
AdaptForVideoMode
    LLGL::Canvas, 118
    LLGL::Surface, 325
    LLGL::Window, 355
Add
    LLGL, 67
AddEventListener
    LLGL::Canvas, 118
    LLGL::Window, 355
addressModeU
    LLGL::SamplerDescriptor, 291
addressModeV
    LLGL::SamplerDescriptor, 291
addressModeW
    LLGL::SamplerDescriptor, 291
All
    LLGL::ClearFlags, 120
AllGraphicsStages
    LLGL::StageFlags, 316
AllStages
    LLGL::StageFlags, 316
AllTessStages
    LLGL::StageFlags, 316
alphaArithmetic
    LLGL::BlendTargetDescriptor, 111
alphaToCoverageEnabled
    LLGL::BlendDescriptor, 108
AlwaysPass
    LLGL, 68
antiAliasedLineEnabled
    LLGL::RasterizerDescriptor, 224
Any
    LLGL, 77
AnySamplesPassed
    LLGL, 81
AnySamplesPassedConservative
    LLGL, 81
AppendAttribute
    LLGL::StreamOutputFormat, 323
    LLGL::VertexFormat, 342
AppendAttributes
    LLGL::StreamOutputFormat, 324
AppendStructuredBuffer
    LLGL, 88
application
    LLGL::VulkanRendererConfiguration, 352
applicationName
    LLGL::ApplicationDescriptor, 99
applicationVersion
    LLGL::ApplicationDescriptor, 99
Apps
    LLGL, 76
arrayLayer
    LLGL::AttachmentDescriptor, 103
arrayLayers
    LLGL::TextureDescriptor, 330
arraySize
    LLGL::BindingDescriptor, 107
    LLGL::ShaderReflectionDescriptor::ResourceView,
        286
AssertCreateBuffer
    LLGL::RenderSystem, 257
AssertCreateBufferArray
    LLGL::RenderSystem, 257
AssertCreateRenderPass
    LLGL::RenderSystem, 258
AssertCreateRenderTarget
    LLGL::RenderSystem, 258
AssertCreateShader
    LLGL::RenderSystem, 258
AssertCreateShaderProgram
    LLGL::RenderSystem, 258

AssertImageDataSize
LLGL::RenderSystem, 258
AtomicCounter
LLGL, 89
AttachmentClear
LLGL::AttachmentClear, 101
attachmentClears
LLGL::FrameProfile, 179
AttachmentDescriptor
LLGL::AttachmentDescriptor, 103
AttachmentFormatDescriptor
LLGL::AttachmentFormatDescriptor, 105
AttachmentLoadOp
LLGL, 65
AttachmentStoreOp
LLGL, 65
AttachmentType
LLGL, 66
attachments
LLGL::RenderTargetDescriptor, 281
Attn
LLGL, 77
attributes
LLGL::StreamOutputFormat, 324
LLGL::VertexFormat, 343

B
LLGL, 75
b
LLGL::Color< T, 3u >, 129
LLGL::Color< T, 4u >, 134
BC1RGBA
LLGL, 73
BC1RGB
LLGL, 73
BC2RGBA
LLGL, 73
BC3RGBA
LLGL, 73
BGRA8SInt
LLGL, 73
BGRA8SNorm
LLGL, 73
BGRA8UInt
LLGL, 73
BGRA8UNorm
LLGL, 73
BGRA8sRGB
LLGL, 73
BGRA
LLGL, 74
BGR
LLGL, 74
Back
LLGL, 69, 74
back
LLGL::StencilDescriptor, 316
Begin
LLGL::CommandBuffer, 137
BeginQuery
LLGL::CommandBuffer, 137
BeginRenderCondition
LLGL::CommandBuffer, 138
BeginRenderPass
LLGL::CommandBuffer, 138
BeginStreamOutput
LLGL::CommandBuffer, 139
BinaryBuffer
LLGL, 85
BinaryFile
LLGL, 85
BindConstantBuffer
LLGL::ShaderProgram, 302
BindStorageBuffer
LLGL::ShaderProgram, 303
BindingDescriptor
LLGL::BindingDescriptor, 107
bindings
LLGL::PipelineLayoutDescriptor, 217
blend
LLGL::GraphicsPipelineDescriptor, 187
BlendArithmetic
LLGL, 66
blendEnabled
LLGL::BlendTargetDescriptor, 111
BlendFactor
LLGL, 67
blendFactor
LLGL::BlendDescriptor, 108
BlendOp
LLGL, 67
Blit
LLGL::Image, 192
Block
LLGL::RenderingDebugger::Message, 207
BlockAfter
LLGL::RenderingDebugger::Message, 207
Bool1
LLGL, 89
Bool2
LLGL, 89
Bool3
LLGL, 89
Bool4
LLGL, 89
Border
LLGL, 84
borderColor
LLGL::SamplerDescriptor, 291
borderless
LLGL::CanvasDescriptor, 120
LLGL::WindowDescriptor, 361
BrowserBack
LLGL, 77
BrowserFavorits
LLGL, 77
BrowserForward

LLGL, 77
BrowserHome
LLGL, 77
BrowserRefresh
LLGL, 77
BrowserSearch
LLGL, 77
BrowserStop
LLGL, 77
Buffer
LLGL::Buffer, 112
LLGL, 88
Buffer utility functions to determine buffer types., 26
IsByteAddressBuffer, 26
IsRWBuffer, 26
IsStructuredBuffer, 26
IsTypedBuffer, 26
Buffer.h, 365
BufferArray
LLGL::BufferArray, 114
BufferArray.h, 365
bufferCopies
LLGL::FrameProfile, 179
BufferFlags.h, 366
bufferMappings
LLGL::FrameProfile, 180
bufferReads
LLGL::FrameProfile, 180
BufferType
LLGL, 67
bufferUpdates
LLGL::FrameProfile, 180
bufferWrites
LLGL::FrameProfile, 180
ByRegionNoWait
LLGL, 82
ByRegionNoWaitInverted
LLGL, 82
ByRegionWait
LLGL, 82
ByRegionWaitInverted
LLGL, 82
ByteAddressBuffer
LLGL, 88
ByteBuffer
LLGL, 64

C
LLGL, 75
CPUAccess
LLGL, 68
Cancel
LLGL, 74
Canvas
LLGL::Canvas::EventListener, 171
Canvas.h, 367
CanvasFlags.h, 367
Capital
LLGL, 74
Cast
LLGL::Color, 123
LLGL::Color< T, 3u >, 127
LLGL::Color< T, 4u >, 132
CastColorValue
LLGL, 90
CastColorValue< bool, bool >
LLGL, 90
CastColorValue< double, double >
LLGL, 90
CastColorValue< float, float >
LLGL, 90
CastColorValue< std::uint8_t, std::uint8_t >
LLGL, 90
centered
LLGL::WindowDescriptor, 361
Clamp
LLGL, 84
clamp
LLGL::DepthBiasDescriptor, 162
Clear
LLGL::CommandBuffer, 140
LLGL::FrameProfile, 179
LLGL, 65, 74, 78
ClearAttachments
LLGL::CommandBuffer, 140
clearValue
LLGL::AttachmentClear, 101
LLGL::ImageInitialization, 199
clippingInvocations
LLGL::QueryPipelineStatistics, 222
clippingPrimitives
LLGL::QueryPipelineStatistics, 222
ClippingRange
LLGL, 68
clippingRange
LLGL::RenderingCapabilities, 238
CodeFile
LLGL, 85
CodeString
LLGL, 85
Color
LLGL::ClearFlags, 120
LLGL::Color, 123
LLGL::Color< T, 3u >, 127
LLGL::Color< T, 4u >, 131, 132
LLGL, 66
color
LLGL::ClearValue, 121
Color.h, 367
colorArithmetic
LLGL::BlendTargetDescriptor, 111
colorAttachment
LLGL::AttachmentClear, 101
ColorAttachmentUsage
LLGL::TextureFlags, 332
colorAttachments
LLGL::RenderPassDescriptor, 253

colorBits
LLGL::VideoModeDescriptor, 346
ColorDepth
LLGL::ClearFlags, 120
colorMap
LLGL::NativeContextHandle, 210
colorMask
LLGL::BlendTargetDescriptor, 111
ColorRGB.h, 369
ColorRGBA.h, 369
ColorRGBAb
LLGL, 64
ColorRGBAd
LLGL, 65
ColorRGBAf
LLGL, 65
ColorRGBAT
LLGL, 65
ColorRGBAub
LLGL, 65
ColorRGBb
LLGL, 65
ColorRGBd
LLGL, 65
ColorRGBf
LLGL, 65
ColorRGBT
LLGL, 65
ColorRGBub
LLGL, 65
Comma
LLGL, 77
CommandBuffer
LLGL::CommandBuffer, 137
CommandBuffer.h, 370
commandBufferEncodings
LLGL::FrameProfile, 180
CommandBufferExt
LLGL::CommandBufferExt, 154
CommandBufferExt.h, 370
CommandBufferFlags.h, 370
commandBufferSubmittions
LLGL::FrameProfile, 180
CommandQueue
LLGL::CommandQueue, 157
CommandQueue.h, 371
compareEnabled
LLGL::SamplerDescriptor, 291
CompareOp
LLGL, 68
compareOp
LLGL::DepthDescriptor, 163
LLGL::SamplerDescriptor, 291
LLGL::StencilFaceDescriptor, 318
CompareSWO
Global functions for Strict-Weak-Order (SWO) comparisons., 21
CompatibilityProfile
LLGL, 78
components
LLGL::Color, 125
LLGL::Color< T, 3u >, 129
LLGL::Color< T, 4u >, 134
LLGL::StreamOutputAttribute, 322
CompressedRGBA
LLGL, 74
CompressedRGB
LLGL, 74
Compute
LLGL, 86
ComputePipeline.h, 372
computePipelineBindings
LLGL::FrameProfile, 181
ComputePipelineDescriptor
LLGL::ComputePipelineDescriptor, 160
ComputePipelineFlags.h, 372
computeResourceHeapBindings
LLGL::FrameProfile, 181
computeShader
LLGL::ShaderProgramDescriptor, 306
computeShaderInvocations
LLGL::QueryPipelineStatistics, 222
ComputeStage
LLGL::StageFlags, 315
conservativeRasterization
LLGL::RasterizerDescriptor, 224
Constant
LLGL, 68
ConstantBuffer
LLGL, 83
constantBufferBindings
LLGL::FrameProfile, 181
ConstantBufferDesc
Global utility functions, especially to fill descriptor structures., 42
constantBufferSize
LLGL::ShaderReflectionDescriptor::ResourceView, 286
constantFactor
LLGL::DepthBiasDescriptor, 162
Constants.h, 372
ConsumeStructuredBuffer
LLGL, 88
contextProfile
LLGL::ProfileOpenGLDescriptor, 218
Control
LLGL, 74
Convert
LLGL::Image, 193
ConvertImageBuffer
Image utility functions to classify and convert image data., 31, 32
Copy
LLGL, 78
CopyBuffer
LLGL::CommandBuffer, 141

CopyInverted
LLGL, 78
CoreProfile
LLGL, 78
CrSel
LLGL, 77
Create
LLGL::Canvas, 118
LLGL::Timer, 335
LLGL::Window, 355
CreateBuffer
LLGL::RenderSystem, 258
CreateBufferArray
LLGL::RenderSystem, 258
CreateCommandBuffer
LLGL::RenderSystem, 259
CreateCommandBufferExt
LLGL::RenderSystem, 259
CreateComputePipeline
LLGL::RenderSystem, 260
CreateFence
LLGL::RenderSystem, 260
CreateGraphicsPipeline
LLGL::RenderSystem, 260
CreatePipelineLayout
LLGL::RenderSystem, 260
CreateQueryHeap
LLGL::RenderSystem, 261
CreateRenderContext
LLGL::RenderSystem, 261
CreateRenderPass
LLGL::RenderSystem, 261
CreateRenderTarget
LLGL::RenderSystem, 262
CreateResourceHeap
LLGL::RenderSystem, 262
CreateSampler
LLGL::RenderSystem, 262
CreateShader
LLGL::RenderSystem, 264
CreateShaderProgram
LLGL::RenderSystem, 264
CreateTexture
LLGL::RenderSystem, 264
CullMode
LLGL, 69
cullMode
LLGL::RasterizerDescriptor, 224
customMultiSampling
LLGL::RenderTargetDescriptor, 281

D
LLGL, 75
D0
LLGL, 75
D1
LLGL, 75
D16UNorm
LLGL, 73
D2
LLGL, 75
D24UNormS8UInt
LLGL, 73
D3
LLGL, 75
D32Float
LLGL, 73
D32FloatS8X24UInt
LLGL, 73
D4
LLGL, 75
D5
LLGL, 75
D6
LLGL, 75
D7
LLGL, 75
D8
LLGL, 75
D9
LLGL, 75
data
LLGL::DstImageDescriptor, 169
LLGL::SrcImageDescriptor, 314
Data type utility functions., 30
DataTypeSize, 30
IsFloatDataType, 30
IsIntDataType, 30
IsUIntDataType, 30
dataSize
LLGL::DstImageDescriptor, 169
LLGL::SrcImageDescriptor, 314
DataType
LLGL, 69
dataType
LLGL::DstImageDescriptor, 169
LLGL::SrcImageDescriptor, 314
DataTypeSize
Data type utility functions., 30
Debug
LLGL::ShaderCompileFlags, 297
DebugCallback
Global type aliases to callback interfaces., 25
debugCallback
LLGL::RenderContextDescriptor, 233
DecClamp
LLGL, 87
DecWrap
LLGL, 87
Default
LLGL::TextureFlags, 332
DeferredSubmit
LLGL::CommandBufferFlags, 156
Delete
LLGL, 75
Depth
LLGL::ClearFlags, 120

LLGL, 66, 74
depth
LLGL::ClearValue, 121
LLGL::Extent3D, 176
LLGL::GraphicsPipelineDescriptor, 187
depthAttachment
LLGL::RenderPassDescriptor, 253
depthBias
LLGL::RasterizerDescriptor, 224
depthBits
LLGL::VideoModeDescriptor, 346
depthClampEnabled
LLGL::RasterizerDescriptor, 225
depthFailOp
LLGL::StencilFaceDescriptor, 318
depthPassOp
LLGL::StencilFaceDescriptor, 318
DepthStencil
LLGL::ClearFlags, 120
LLGL, 66, 74
DepthStencilAttachmentUsage
LLGL::TextureFlags, 332
deviceName
LLGL::RendererInfo, 237
Direct3D10
LLGL::RendererID, 235
Direct3D11
LLGL::RendererID, 235
Direct3D12
LLGL::RendererID, 235
Direct3D9
LLGL::RendererID, 235
disableClearOnResize
LLGL::WindowBehavior, 360
Disabled
LLGL, 69, 78
Disassemble
LLGL::Shader, 295
discardEnabled
LLGL::RasterizerDescriptor, 225
Dispatch
LLGL::CommandBuffer, 141
dispatchCommands
LLGL::FrameProfile, 181
display
LLGL::NativeContextHandle, 210
LLGL::NativeHandle, 211
Display.h, 372
DisplayFlags.h, 373
displayModes
LLGL::VideoOutputDescriptor, 348
Double1
LLGL, 89
Double2
LLGL, 89
Double2x2
LLGL, 89
Double2x3
LLGL, 89
Double2x4
LLGL, 89
Double3
LLGL, 89
Double3x2
LLGL, 89
Double3x3
LLGL, 89
Double3x4
LLGL, 89
Double4
LLGL, 89
Double4x2
LLGL, 89
Double4x3
LLGL, 89
Double4x4
LLGL, 89
Down
LLGL, 75
Draw
LLGL::CommandBuffer, 142
drawCommands
LLGL::FrameProfile, 181
DrawIndexed
LLGL::CommandBuffer, 142
DrawIndexedInstanced
LLGL::CommandBuffer, 142, 143
DrawInstanced
LLGL::CommandBuffer, 143
DstAlpha
LLGL, 67
dstAlpha
LLGL::BlendTargetDescriptor, 111
DstColor
LLGL, 67
dstColor
LLGL::BlendTargetDescriptor, 111
DstImageDescriptor
LLGL::DstImageDescriptor, 169
DynamicUsage
LLGL::BufferFlags, 116

E
LLGL, 75
ESProfile
LLGL, 78
ESSL_100
LLGL, 86
ESSL_300
LLGL, 87
ESSL_310
LLGL, 87
ESSL_320
LLGL, 87
ESSL
LLGL, 86
enabled

LLGL::ImageInitialization, 199
LLGL::MultiSamplingDescriptor, 209
LLGL::VsyncDescriptor, 351
End
LLGL::CommandBuffer, 144
LLGL, 75
EndQuery
LLGL::CommandBuffer, 144
EndRenderCondition
LLGL::CommandBuffer, 144
EndRenderPass
LLGL::CommandBuffer, 144
EndStreamOutput
LLGL::CommandBuffer, 144
engineName
LLGL::ApplicationDescriptor, 100
engineVersion
LLGL::ApplicationDescriptor, 100
entryPoint
LLGL::ShaderDescriptor, 299
Equal
LLGL, 68
Equiv
LLGL, 78
ErEOF
LLGL, 77
Error
LLGL::Log, 95
ErrorType
LLGL, 70
Escape
LLGL, 74
ExSel
LLGL, 77
Exe
LLGL, 75
Exponent
LLGL, 77
Export.h, 373
LLGL_EXPORT, 374
extent
LLGL::TextureDescriptor, 330
LLGL::TextureRegion, 333
Extent2D
LLGL::Extent2D, 175
Extent3D
LLGL::Extent3D, 176

F
LLGL, 75
F1
LLGL, 76
F10
LLGL, 76
F11
LLGL, 76
F12
LLGL, 76
F13
LLGL, 76
F14
LLGL, 76
F15
LLGL, 76
F16
LLGL, 76
F17
LLGL, 76
F18
LLGL, 76
F19
LLGL, 76
F2
LLGL, 76
F20
LLGL, 76
F21
LLGL, 76
F22
LLGL, 77
F23
LLGL, 77
F24
LLGL, 77
F3
LLGL, 76
F4
LLGL, 76
F5
LLGL, 76
F6
LLGL, 76
F7
LLGL, 76
F8
LLGL, 76
F9
LLGL, 76
features
LLGL::RenderingCapabilities, 238
Fence.h, 374
fenceSubmissions
LLGL::FrameProfile, 182
Fill
LLGL::Image, 193
LLGL, 79
FinalizeDisplayModes
LLGL::Display, 165
FinalizeShaderReflection
LLGL::ShaderProgram, 303
FindModules
LLGL::RenderSystem, 265
FindSuitableImageFormat
Image utility functions to classify and convert image data., 33
FixedSamples
LLGL::TextureFlags, 332

flags
    LLGL::AttachmentClear, 102
    LLGL::BufferDescriptor, 115
    LLGL::CommandBufferDescriptor, 152
    LLGL::ShaderDescriptor, 299
    LLGL::TextureDescriptor, 330
Float1
    LLGL, 89
Float16
    LLGL, 70
Float2
    LLGL, 89
Float2x2
    LLGL, 89
Float2x3
    LLGL, 89
Float2x4
    LLGL, 89
Float3
    LLGL, 89
Float32
    LLGL, 70
Float3x2
    LLGL, 89
Float3x3
    LLGL, 89
Float3x4
    LLGL, 89
Float4
    LLGL, 89
Float4x2
    LLGL, 89
Float4x3
    LLGL, 89
Float4x4
    LLGL, 89
Float64
    LLGL, 70
Format
    LLGL, 70
format
    LLGL::AttachmentFormatDescriptor, 105
    LLGL::BufferDescriptor::IndexBuffer, 200
    LLGL::BufferDescriptor::StorageBuffer, 320
    LLGL::BufferDescriptor::VertexBuffer, 341
    LLGL::DstImageDescriptor, 170
    LLGL::ShaderDescriptor::StreamOutput, 321
    LLGL::SrcImageDescriptor, 314
    LLGL::TextureDescriptor, 330
    LLGL::VertexAttribute, 340
Format.h, 374
FormatBitSize
    Hardware format utility functions., 27
ForwardDecls.h, 376
Fragment
    LLGL, 86
fragmentShader
    LLGL::ShaderProgramDescriptor, 306
fragmentShaderInvocations
    LLGL::QueryPipelineStatistics, 222
FragmentStage
    LLGL::StageFlags, 315
FrameProfile
    LLGL::FrameProfile, 179
frameProfile
    LLGL::RenderingProfiler, 252
Front
    LLGL, 69
front
    LLGL::StencilDescriptor, 316
frontCCW
    LLGL::RasterizerDescriptor, 225
fullscreen
    LLGL::VideoModeDescriptor, 346

G
    LLGL, 75
g
    LLGL::Color< T, 3u >, 129
    LLGL::Color< T, 4u >, 134
GLSL_110
    LLGL, 86
GLSL_120
    LLGL, 86
GLSL_130
    LLGL, 86
GLSL_140
    LLGL, 86
GLSL_150
    LLGL, 86
GLSL_330
    LLGL, 86
GLSL_400
    LLGL, 86
GLSL_410
    LLGL, 86
GLSL_420
    LLGL, 86
GLSL_430
    LLGL, 86
GLSL_440
    LLGL, 86
GLSL_450
    LLGL, 86
GLSL_460
    LLGL, 86
GLSL
    LLGL, 86
GenerateEmptyByteBuffer
    Image utility functions to classify and convert image data., 33
GenerateImageBuffer
    Image utility functions to classify and convert image data., 34
GenerateMips
    LLGL::RenderSystem, 265
Geometry

LLGL, 86
geometryShader
LLGL::ShaderProgramDescriptor, 307
geometryShaderInvocations
LLGL::QueryPipelineStatistics, 222
geometryShaderPrimitives
LLGL::QueryPipelineStatistics, 222
GeometryStage
LLGL::StageFlags, 315
GetAnyKeyCount
LLGL::Input, 203
GetBehavior
LLGL::Window, 355
GetBytesPerPixel
LLGL::Image, 193
GetCommandQueue
LLGL::RenderSystem, 266
GetConfiguration
LLGL::RenderSystem, 266
GetContentSize
LLGL::Surface, 325
GetData
LLGL::Image, 194
GetDataSize
LLGL::Image, 194
GetDataType
LLGL::Image, 194
LLGL::IndexFormat, 201
GetDeltaTime
LLGL::Timer, 335
GetDepthStride
LLGL::Image, 194
GetDesc
LLGL::Window, 356
GetDeviceName
LLGL::Display, 165
GetDisplayMode
LLGL::Display, 165
GetEnteredChars
LLGL::Input, 203
GetExtent
LLGL::Image, 194
GetExtentRatio
LLGL, 90
GetFormat
LLGL::Image, 194
GetFormatSize
LLGL::IndexFormat, 201
GetFrequency
LLGL::Timer, 335
GetID
LLGL::Version, 97
GetMajor
LLGL::Version, 97
GetMinor
LLGL::Version, 97
GetMouseMotion
LLGL::Input, 204
GetMousePosition
LLGL::Input, 204
GetName
LLGL::RenderSystem, 266
GetNativeHandle
LLGL::Surface, 326
GetNumColorAttachments
LLGL::RenderContext, 228
LLGL::RenderTarget, 278
GetNumPixels
LLGL::Image, 194
GetOccurrences
LLGL::RenderingDebugger::Message, 207
GetOffset
LLGL::Display, 165
GetPosition
LLGL::Window, 356
GetPrimitiveTopologyPatchSize
LLGL, 90
GetRenderPass
LLGL::RenderTarget, 278
GetRendererID
LLGL::RenderSystem, 266
GetRendererInfo
LLGL::RenderSystem, 267
GetRenderingCaps
LLGL::RenderSystem, 267
GetResolution
LLGL::RenderContext, 228
LLGL::RenderTarget, 278
GetRevision
LLGL::Version, 97
GetRowStride
LLGL::Image, 195
GetSize
LLGL::VertexAttribute, 339
LLGL::Window, 356
GetSource
LLGL::RenderingDebugger::Message, 207
GetStageFlags
LLGL::Shader, 295
GetStatus
LLGL::Version, 98
GetString
LLGL::Version, 98
GetSurface
LLGL::RenderContext, 228
GetText
LLGL::RenderingDebugger::Message, 207
GetTitle
LLGL::Canvas, 118
LLGL::Window, 356
GetType
LLGL::Buffer, 112
LLGL::BufferArray, 114
LLGL::QueryHeap, 219
LLGL::Shader, 296
LLGL::Texture, 327

GetVideoMode
LLGL::RenderContext, 228
GetVsync
LLGL::RenderContext, 228
GetWheelMotion
LLGL::Input, 204
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., 36
ToString, 36, 37
Global utility functions, especially to fill descriptor structures., 41
ConstantBufferDesc, 42
IndexBufferDesc, 42
PipelineLayoutDesc, 42
RenderPassDesc, 44
ShaderDescFromFile, 44
ShaderProgramDesc, 44, 45
StorageBufferDesc, 45
Texture1DArrayDesc, 45
Texture1DDesc, 45
Texture2DArrayDesc, 46
Texture2DDesc, 46
Texture2DMSArrayDesc, 46
Texture2DMSDesc, 46
Texture3DDesc, 46
TextureCubeArrayDesc, 47
TextureCubeDesc, 47
VertexBufferDesc, 47
GraphicsPipeline.h, 376
graphicsPipelineBindings
LLGL::FrameProfile, 182
GraphicsPipelineFlags.h, 376
graphicsResourceHeapBindings
LLGL::FrameProfile, 182
Greater
LLGL, 68
GreaterEqual
LLGL, 68

H
LLGL, 75
HLSL_2_0
LLGL, 87
HLSL_2_0a
LLGL, 87
HLSL_2_0b
LLGL, 87
HLSL_3_0
LLGL, 87
HLSL_4_0
LLGL, 87
HLSL_4_1
LLGL, 87
HLSL_5_0
LLGL, 87
HLSL_5_1
LLGL, 87
HLSL
LLGL, 87
Hardware format utility functions., 27
FormatBitSize, 27
IsCompressedFormat, 27
IsDepthFormat, 28
IsDepthStencilFormat, 28
IsFloatFormat, 28
IsIntegralFormat, 28
IsNormalizedFormat, 29
IsStencilFormat, 29
SplitFormat, 29
has3DTextures
LLGL::RenderingFeatures, 242
hasArrayTextures
LLGL::RenderingFeatures, 242
hasCommandBufferExt
LLGL::RenderingFeatures, 243
hasComputeShaders
LLGL::RenderingFeatures, 243
hasConservativeRasterization
LLGL::RenderingFeatures, 243
hasConstantBuffers
LLGL::RenderingFeatures, 243
hasCubeArrayTextures
LLGL::RenderingFeatures, 243
hasCubeTextures
LLGL::RenderingFeatures, 244
HasDepthAttachment
LLGL::RenderContext, 229
LLGL::RenderTarget, 279
HasErrors
LLGL::Shader, 296
LLGL::ShaderProgram, 303
HasFocus
LLGL::Window, 356
hasGeometryShaders
LLGL::RenderingFeatures, 244
hasInstancing
LLGL::RenderingFeatures, 244
hasLogicOp
LLGL::RenderingFeatures, 244
hasMultiSampleTextures
LLGL::RenderingFeatures, 244
hasOffsetInstancing
LLGL::RenderingFeatures, 245
hasPipelineStatistics
LLGL::RenderingFeatures, 245
hasRenderCondition
LLGL::RenderingFeatures, 245

hasRenderTargets
LLGL::RenderingFeatures, 245
hasSamplers
LLGL::RenderingFeatures, 245
HasStencilAttachment
LLGL::RenderContext, 229
LLGL::RenderTarget, 279
hasStorageBuffers
LLGL::RenderingFeatures, 245
hasStreamOutputs
LLGL::RenderingFeatures, 246
hasTessellationShaders
LLGL::RenderingFeatures, 246
hasUniforms
LLGL::RenderingFeatures, 246
hasViewportArrays
LLGL::RenderingFeatures, 246
height
LLGL::Extent2D, 175
LLGL::Extent3D, 176
LLGL::Scissor, 294
LLGL::Viewport, 350
Help
LLGL, 75
Home
LLGL, 75

I
LLGL, 75
IOSNativeHandle.h, 381
Image
LLGL::Image, 192
LLGL, 89
Image utility functions to classify and convert image data., 31
ConvertImageBuffer, 31, 32
FindSuitableImageFormat, 33
GenerateEmptyByteBuffer, 33
GenerateImageBuffer, 34
ImageDataSize, 34
ImageFormatSize, 35
IsCompressedFormat, 35
IsDepthStencilFormat, 35
Image.h, 378
ImageDataSize
Image utility functions to classify and convert image data., 34
ImageFlags.h, 379
ImageFormat
LLGL, 73
ImageFormatSize
Image utility functions to classify and convert image data., 35
imageInitialization
LLGL::RenderSystemConfiguration, 274
ImproperArgument
LLGL, 90
ImproperState
LLGL, 90
IncClamp
LLGL, 87
IncOccurrence
LLGL::RenderingDebugger::Message, 207
IncWrap
LLGL, 87
IncompleteAttachments
LLGL::ShaderProgram, 302
independentBlendEnabled
LLGL::BlendDescriptor, 109
Index
LLGL, 68
IndexBuffer
LLGL, 82
indexBuffer
LLGL::BufferDescriptor, 115
indexBufferBindings
LLGL::FrameProfile, 182
IndexBufferDesc
Global utility functions, especially to fill descriptor structures., 42
IndexFormat
LLGL::IndexFormat, 201
IndexFormat.h, 380
Information
LLGL::Log, 95
Input
LLGL::Input, 203
Input.h, 380
inputAssemblyPrimitives
LLGL::QueryPipelineStatistics, 222
inputAssemblyVertices
LLGL::QueryPipelineStatistics, 223
inputSlot
LLGL::VertexFormat, 343
Insert
LLGL, 75
instanceDivisor
LLGL::VertexAttribute, 340
InstructionOnly
LLGL::ShaderDisassembleFlags, 301
Int1
LLGL, 89
Int16
LLGL, 70
Int2
LLGL, 89
Int3
LLGL, 89
Int32
LLGL, 70
Int4
LLGL, 89
Int8
LLGL, 70
interval
LLGL::VsyncDescriptor, 351
InvBlendFactor

LLGL, 67
InvDstAlpha
LLGL, 67
InvDstColor
LLGL, 67
InvSrc1Alpha
LLGL, 67
InvSrc1Color
LLGL, 67
InvSrcAlpha
LLGL, 67
InvSrcColor
LLGL, 67
InvalidArgument
LLGL, 70
InvalidByteCode
LLGL::ShaderProgram, 302
InvalidComposition
LLGL::ShaderProgram, 302
InvalidState
LLGL, 70
Invert
LLGL, 78, 87
invertFrontFace
LLGL::OpenGLDependentStateDescriptor, 216
IsArrayTexture
Texture utility functions to determine texture dimension and buffer sizes., 38
IsBlocked
LLGL::RenderingDebugger::Message, 208
IsByteAddressBuffer
Buffer utility functions to determine buffer types., 26
IsCompressedFormat
Hardware format utility functions., 27
Image utility functions to classify and convert image data., 35
IsCubeTexture
Texture utility functions to determine texture dimension and buffer sizes., 38
IsCursorShown
LLGL::Display, 165
IsDepthFormat
Hardware format utility functions., 28
IsDepthStencilFormat
Hardware format utility functions., 28
Image utility functions to classify and convert image data., 35
IsFloatDataType
Data type utility functions., 30
IsFloatFormat
Hardware format utility functions., 28
IsIntDataType
Data type utility functions., 30
IsIntegralFormat
Hardware format utility functions., 28
IsMipMappedTexture
Texture utility functions to determine texture dimension and buffer sizes., 38
IsMultiSampleTexture
Texture utility functions to determine texture dimension and buffer sizes., 39
IsNormalizedFormat
Hardware format utility functions., 29
IsPrimary
LLGL::Display, 165
IsPrimitiveTopologyPatches
LLGL, 91
IsRWBuffer
Buffer utility functions to determine buffer types., 26
IsRegionInside
LLGL::Image, 195
IsRenderContext
LLGL::RenderContext, 229
LLGL::RenderTarget, 279
IsRunning
LLGL::Timer, 335
IsShaderSourceBinary
LLGL, 91
IsShaderSourceCode
LLGL, 91
IsShown
LLGL::Window, 356
IsStencilFormat
Hardware format utility functions., 29
IsStructuredBuffer
Buffer utility functions to determine buffer types., 26
IsTypedBuffer
Buffer utility functions to determine buffer types., 26
IsUIntDataType
Data type utility functions., 30

J
LLGL, 75

K
LLGL, 75
Keep
LLGL, 87
Key
LLGL, 74
Key.h, 381
KeyDoubleClick
LLGL::Input, 204
KeyDown
LLGL::Input, 204
KeyDownRepeated
LLGL::Input, 204
KeyPressed
LLGL::Input, 204
KeyUp
LLGL::Input, 204
Keypad0
LLGL, 76
Keypad1
LLGL, 76
Keypad2
LLGL, 76

Keypad3
LLGL, 76
Keypad4
LLGL, 76
Keypad5
LLGL, 76
Keypad6
LLGL, 76
Keypad7
LLGL, 76
Keypad8
LLGL, 76
Keypad9
LLGL, 76
KeypadDecimal
LLGL, 76
KeypadDivide
LLGL, 76
KeypadMinus
LLGL, 76
KeypadMultiply
LLGL, 76
KeypadPlus
LLGL, 76
KeypadSeparator
LLGL, 76

L
LLGL, 75
LButton
LLGL, 74
LControl
LLGL, 77
LLGL.h, 383
LLGL::ApplicationDescriptor, 99
applicationName, 99
applicationVersion, 99
engineName, 100
engineVersion, 100
LLGL::AttachmentClear, 100
AttachmentClear, 101
clearValue, 101
colorAttachment, 101
flags, 102
operator=, 101
LLGL::AttachmentDescriptor, 102
arrayLayer, 103
AttachmentDescriptor, 103
mipLevel, 103
texture, 104
type, 104
LLGL::AttachmentFormatDescriptor, 104
AttachmentFormatDescriptor, 105
format, 105
loadOp, 105
storeOp, 106
LLGL::BindingDescriptor, 106
arraySize, 107
BindingDescriptor, 107
slot, 107
stageFlags, 107
type, 107
LLGL::BlendDescriptor, 108
alphaToCoverageEnabled, 108
blendFactor, 108
independentBlendEnabled, 109
logicOp, 109
targets, 109
LLGL::BlendTargetDescriptor, 110
alphaArithmetic, 111
blendEnabled, 111
colorArithmetic, 111
colorMask, 111
dstAlpha, 111
dstColor, 111
srcAlpha, 111
srcColor, 111
LLGL::Buffer, 112
Buffer, 112
GetType, 112
QueryResourceType, 112
LLGL::BufferArray, 113
BufferArray, 114
GetType, 114
LLGL::BufferDescriptor, 114
flags, 115
indexBuffer, 115
size, 115
storageBuffer, 115
type, 115
vertexBuffer, 115
LLGL::BufferDescriptor::IndexBuffer, 200
format, 200
LLGL::BufferDescriptor::StorageBuffer, 319
format, 320
storageType, 320
stride, 320
LLGL::BufferDescriptor::VertexBuffer, 341
format, 341
LLGL::BufferFlags, 116
DynamicUsage, 116
MapReadAccess, 116
MapReadWriteAccess, 116
MapWriteAccess, 116
LLGL::Canvas, 117
AdaptForVideoMode, 118
AddEventListener, 118
Create, 118
GetTitle, 118
OnProcessEvents, 118
ProcessEvents, 119
RemoveEventListener, 119
SetTitle, 119
LLGL::Canvas::EventListener, 170
∼EventListener, 170
Canvas, 171
OnProcessEvents, 170

LLGL::CanvasDescriptor, 119
borderless, 120
title, 120
LLGL::ClearFlags, 120
All, 120
Color, 120
ColorDepth, 120
Depth, 120
DepthStencil, 120
Stencil, 120
LLGL::ClearValue, 121
color, 121
depth, 121
stencil, 121
LLGL::Color
Cast, 123
Color, 123
components, 125
operator∗=, 124
operator+=, 124
operator-, 124
operator-=, 124
operator/=, 124
operator[], 124, 125
Ptr, 125
LLGL::Color< T, 3u >, 125
b, 129
Cast, 127
Color, 127
components, 129
g, 129
operator∗=, 128
operator+=, 128
operator-, 128
operator-=, 128
operator/=, 128
operator[], 128, 129
Ptr, 129
r, 130
ToRGBA, 129
LLGL::Color< T, 4u >, 130
a, 134
b, 134
Cast, 132
Color, 131, 132
components, 134
g, 134
operator∗=, 133
operator+=, 133
operator-, 133
operator-=, 133
operator/=, 133
operator[], 133, 134
Ptr, 134
r, 135
ToRGB, 134
LLGL::Color< T, N >, 122
LLGL::CommandBuffer, 135
Begin, 137
BeginQuery, 137
BeginRenderCondition, 138
BeginRenderPass, 138
BeginStreamOutput, 139
Clear, 140
ClearAttachments, 140
CommandBuffer, 137
CopyBuffer, 141
Dispatch, 141
Draw, 142
DrawIndexed, 142
DrawIndexedInstanced, 142, 143
DrawInstanced, 143
End, 144
EndQuery, 144
EndRenderCondition, 144
EndRenderPass, 144
EndStreamOutput, 144
SetClearColor, 145
SetClearDepth, 145
SetClearStencil, 145
SetComputePipeline, 146
SetComputeResourceHeap, 146
SetGraphicsAPIDependentState, 146
SetGraphicsPipeline, 147
SetGraphicsResourceHeap, 147
SetIndexBuffer, 148
SetScissor, 148
SetScissors, 148
SetStreamOutputBuffer, 149
SetStreamOutputBufferArray, 149
SetVertexBuffer, 150
SetVertexBufferArray, 150
SetViewport, 150
SetViewports, 150
UpdateBuffer, 151
LLGL::CommandBufferDescriptor, 152
flags, 152
numNativeBuffers, 152
LLGL::CommandBufferExt, 153
CommandBufferExt, 154
ResetResourceSlots, 154
SetConstantBuffer, 154
SetSampler, 154
SetStorageBuffer, 155
SetTexture, 155
LLGL::CommandBufferFlags, 156
DeferredSubmit, 156
LLGL::CommandQueue, 156
CommandQueue, 157
QueryResult, 157
Submit, 158
WaitFence, 158
WaitIdle, 159
LLGL::ComputePipeline, 159
LLGL::ComputePipelineDescriptor, 160
ComputePipelineDescriptor, 160

pipelineLayout, 161
shaderProgram, 161
LLGL::Constants, 94
LLGL::DepthBiasDescriptor, 161
clamp, 162
constantFactor, 162
slopeFactor, 162
LLGL::DepthDescriptor, 162
compareOp, 163
testEnabled, 163
writeEnabled, 163
LLGL::Display, 163
FinalizeDisplayModes, 165
GetDeviceName, 165
GetDisplayMode, 165
GetOffset, 165
IsCursorShown, 165
IsPrimary, 165
QueryList, 166
QueryPrimary, 166
QuerySupportedDisplayModes, 166
ResetDisplayMode, 166
SetDisplayMode, 166
ShowCursor, 167
LLGL::DisplayModeDescriptor, 167
refreshRate, 168
resolution, 168
LLGL::DstImageDescriptor, 168
data, 169
dataSize, 169
dataType, 169
DstImageDescriptor, 169
format, 170
LLGL::Extent2D, 174
Extent2D, 175
height, 175
width, 175
LLGL::Extent3D, 175
depth, 176
Extent3D, 176
height, 176
width, 176
LLGL::Fence, 177
LLGL::FrameProfile, 177
Accumulate, 179
attachmentClears, 179
bufferCopies, 179
bufferMappings, 180
bufferReads, 180
bufferUpdates, 180
bufferWrites, 180
Clear, 179
commandBufferEncodings, 180
commandBufferSubmittions, 180
computePipelineBindings, 181
computeResourceHeapBindings, 181
constantBufferBindings, 181
dispatchCommands, 181
drawCommands, 181
fenceSubmissions, 182
FrameProfile, 179
graphicsPipelineBindings, 182
graphicsResourceHeapBindings, 182
indexBufferBindings, 182
mipMapsGenerations, 182
querySections, 183
renderConditionSections, 183
renderPassSections, 183
samplerBindings, 183
storageBufferBindings, 183
streamOutputBufferBindings, 184
streamOutputSections, 184
textureBindings, 184
textureCopies, 184
textureMappings, 184
textureReads, 185
textureWrites, 185
values, 185
vertexBufferBindings, 185
LLGL::GraphicsPipeline, 186
LLGL::GraphicsPipelineDescriptor, 186
blend, 187
depth, 187
pipelineLayout, 187
primitiveTopology, 188
rasterizer, 188
renderPass, 188
scissors, 188
shaderProgram, 188
stencil, 189
viewports, 189
LLGL::Image, 189
Blit, 192
Convert, 193
Fill, 193
GetBytesPerPixel, 193
GetData, 194
GetDataSize, 194
GetDataType, 194
GetDepthStride, 194
GetExtent, 194
GetFormat, 194
GetNumPixels, 194
GetRowStride, 195
Image, 192
IsRegionInside, 195
MirrorXYPlane, 195
MirrorXZPlane, 195
MirrorYZPlane, 195
operator=, 195, 196
QueryDstDesc, 196
QuerySrcDesc, 196
ReadPixels, 196
Release, 197
Reset, 197
Resize, 197, 198

Swap, 198
WritePixels, 198
LLGL::ImageInitialization, 199
clearValue, 199
enabled, 199
LLGL::IndexFormat, 200
GetDataType, 201
GetFormatSize, 201
IndexFormat, 201
operator=, 201
LLGL::Input, 202
GetAnyKeyCount, 203
GetEnteredChars, 203
GetMouseMotion, 204
GetMousePosition, 204
GetWheelMotion, 204
Input, 203
KeyDoubleClick, 204
KeyDown, 204
KeyDownRepeated, 204
KeyPressed, 204
KeyUp, 204
OnChar, 204
OnDoubleClick, 205
OnGlobalMotion, 205
OnKeyDown, 205
OnKeyUp, 205
OnLocalMotion, 205
OnLoseFocus, 205
OnProcessEvents, 205
OnWheelMotion, 206
LLGL::Log, 94
Error, 95
Information, 95
Performance, 96
PostReport, 96
ReportCallback, 95
ReportType, 95
SetReportCallback, 96
SetReportCallbackStd, 96
Warning, 95
LLGL::MultiSamplingDescriptor, 208
enabled, 209
MultiSamplingDescriptor, 208
SampleCount, 209
sampleMask, 209
samples, 209
LLGL::NativeContextHandle, 210
colorMap, 210
display, 210
parentView, 210
parentWindow, 210
screen, 210
visual, 210
LLGL::NativeHandle, 211
display, 211
view, 211
visual, 211
window, 211
LLGL::NonCopyable, 212
∼NonCopyable, 213
NonCopyable, 213
operator=, 213
LLGL::Offset2D, 213
Offset2D, 214
x, 214
y, 214
LLGL::Offset3D, 214
Offset3D, 215
x, 215
y, 215
z, 215
LLGL::OpenGLDependentStateDescriptor, 215
invertFrontFace, 216
originLowerLeft, 216
LLGL::PipelineLayout, 216
LLGL::PipelineLayoutDescriptor, 217
bindings, 217
LLGL::ProfileOpenGLDescriptor, 218
contextProfile, 218
majorVersion, 218
minorVersion, 218
LLGL::QueryHeap, 219
GetType, 219
QueryHeap, 219
LLGL::QueryHeapDescriptor, 220
numQueries, 220
renderCondition, 220
type, 220
LLGL::QueryPipelineStatistics, 221
clippingInvocations, 222
clippingPrimitives, 222
computeShaderInvocations, 222
fragmentShaderInvocations, 222
geometryShaderInvocations, 222
geometryShaderPrimitives, 222
inputAssemblyPrimitives, 222
inputAssemblyVertices, 223
tessControlShaderInvocations, 223
tessEvaluationShaderInvocations, 223
vertexShaderInvocations, 223
LLGL::RasterizerDescriptor, 223
antiAliasedLineEnabled, 224
conservativeRasterization, 224
cullMode, 224
depthBias, 224
depthClampEnabled, 225
discardEnabled, 225
frontCCW, 225
lineWidth, 225
multiSampling, 225
polygonMode, 225
scissorTestEnabled, 226
LLGL::RenderContext, 226
GetNumColorAttachments, 228
GetResolution, 228

GetSurface, 228
GetVideoMode, 228
GetVsync, 228
HasDepthAttachment, 229
HasStencilAttachment, 229
IsRenderContext, 229
OnSetVideoMode, 229
OnSetVsync, 230
Present, 230
QueryColorFormat, 230
QueryDepthStencilFormat, 230
RenderContext, 228
SetDisplayModeByVideoMode, 231
SetOrCreateSurface, 231
SetVideoMode, 231
SetVsync, 232
ShareSurfaceAndConfig, 232
SwitchFullscreenMode, 232
LLGL::RenderContextDescriptor, 233
debugCallback, 233
multiSampling, 233
profileOpenGL, 234
videoMode, 234
vsync, 234
LLGL::RenderPass, 252
LLGL::RenderPassDescriptor, 253
colorAttachments, 253
depthAttachment, 253
stencilAttachment, 254
LLGL::RenderSystem, 254
AssertCreateBuffer, 257
AssertCreateBufferArray, 257
AssertCreateRenderPass, 258
AssertCreateRenderTarget, 258
AssertCreateShader, 258
AssertCreateShaderProgram, 258
AssertImageDataSize, 258
CreateBuffer, 258
CreateBufferArray, 258
CreateCommandBuffer, 259
CreateCommandBufferExt, 259
CreateComputePipeline, 260
CreateFence, 260
CreateGraphicsPipeline, 260
CreatePipelineLayout, 260
CreateQueryHeap, 261
CreateRenderContext, 261
CreateRenderPass, 261
CreateRenderTarget, 262
CreateResourceHeap, 262
CreateSampler, 262
CreateShader, 264
CreateShaderProgram, 264
CreateTexture, 264
FindModules, 265
GenerateMips, 265
GetCommandQueue, 266
GetConfiguration, 266
GetName, 266
GetRendererID, 266
GetRendererInfo, 267
GetRenderingCaps, 267
Load, 267
MapBuffer, 268
ReadTexture, 268
Release, 269–271
RenderSystem, 257
SetConfiguration, 271
SetRendererInfo, 271
SetRenderingCaps, 271
Unload, 272
UnmapBuffer, 272
WriteBuffer, 272
WriteTexture, 272
LLGL::RenderSystemChild, 273
LLGL::RenderSystemConfiguration, 274
imageInitialization, 274
threadCount, 274
LLGL::RenderSystemDescriptor, 275
moduleName, 276
operator=, 276
RenderSystemDescriptor, 275
rendererConfig, 276
rendererConfigSize, 276
LLGL::RenderTarget, 277
GetNumColorAttachments, 278
GetRenderPass, 278
GetResolution, 278
HasDepthAttachment, 279
HasStencilAttachment, 279
IsRenderContext, 279
ValidateMipResolution, 280
ValidateResolution, 280
LLGL::RenderTargetDescriptor, 280
attachments, 281
customMultiSampling, 281
multiSampling, 282
renderPass, 282
resolution, 282
LLGL::RendererID, 234
Direct3D10, 235
Direct3D11, 235
Direct3D12, 235
Direct3D9, 235
Metal, 235
OpenGLES1, 235
OpenGLES2, 235
OpenGLES3, 236
OpenGL, 235
Reserved, 236
Undefined, 236
Vulkan, 236
LLGL::RendererInfo, 236
deviceName, 237
rendererName, 237
shadingLanguageName, 237

vendorName, 237
LLGL::RenderingCapabilities, 237
clippingRange, 238
features, 238
limits, 238
screenOrigin, 238
shadingLanguages, 238
textureFormats, 238
LLGL::RenderingDebugger, 239
∼RenderingDebugger, 240
OnError, 240
OnWarning, 240
PostError, 240
PostWarning, 241
SetSource, 241
LLGL::RenderingDebugger::Message, 206
Block, 207
BlockAfter, 207
GetOccurrences, 207
GetSource, 207
GetText, 207
IncOccurrence, 207
IsBlocked, 208
Message, 207
operator=, 208
RenderingDebugger, 208
LLGL::RenderingFeatures, 241
has3DTextures, 242
hasArrayTextures, 242
hasCommandBufferExt, 243
hasComputeShaders, 243
hasConservativeRasterization, 243
hasConstantBuffers, 243
hasCubeArrayTextures, 243
hasCubeTextures, 244
hasGeometryShaders, 244
hasInstancing, 244
hasLogicOp, 244
hasMultiSampleTextures, 244
hasOffsetInstancing, 245
hasPipelineStatistics, 245
hasRenderCondition, 245
hasRenderTargets, 245
hasSamplers, 245
hasStorageBuffers, 245
hasStreamOutputs, 246
hasTessellationShaders, 246
hasUniforms, 246
hasViewportArrays, 246
LLGL::RenderingLimits, 246
lineWidthRange, 247
max1DTextureSize, 247
max2DTextureSize, 248
max3DTextureSize, 248
maxAnisotropy, 248
maxBufferSize, 248
maxColorAttachments, 248
maxComputeShaderWorkGroupSize, 249
maxComputeShaderWorkGroups, 249
maxConstantBufferSize, 249
maxCubeTextureSize, 249
maxPatchVertices, 249
maxTextureArrayLayers, 250
maxViewportSize, 250
maxViewports, 250
LLGL::RenderingProfiler, 250
Accumulate, 251
frameProfile, 252
NextProfile, 251
LLGL::Resource, 283
QueryResourceType, 283
LLGL::ResourceHeap, 284
LLGL::ResourceHeapDescriptor, 284
pipelineLayout, 285
resourceViews, 285
LLGL::ResourceViewDescriptor, 288
resource, 289
ResourceViewDescriptor, 288
LLGL::Sampler, 289
QueryResourceType, 290
Sampler, 290
LLGL::SamplerDescriptor, 290
addressModeU, 291
addressModeV, 291
addressModeW, 291
borderColor, 291
compareEnabled, 291
compareOp, 291
magFilter, 291
maxAnisotropy, 291
maxLOD, 292
minFilter, 292
minLOD, 292
mipMapFilter, 292
mipMapLODBias, 292
mipMapping, 292
LLGL::Scissor, 293
height, 294
Scissor, 293
width, 294
x, 294
y, 294
LLGL::Shader, 294
Disassemble, 295
GetStageFlags, 295
GetType, 296
HasErrors, 296
QueryInfoLog, 296
Shader, 295
LLGL::ShaderCompileFlags, 296
Debug, 297
O1, 297
O2, 297
O3, 297
WarnError, 297
LLGL::ShaderDescriptor, 297

entryPoint, 299
flags, 299
operator=, 299
profile, 299
ShaderDescriptor, 298
source, 299
sourceSize, 299
sourceType, 300
streamOutput, 300
type, 300
LLGL::ShaderDescriptor::StreamOutput, 320
format, 321
LLGL::ShaderDisassembleFlags, 300
InstructionOnly, 301
LLGL::ShaderProgram, 301
BindConstantBuffer, 302
BindStorageBuffer, 303
FinalizeShaderReflection, 303
HasErrors, 303
IncompleteAttachments, 302
InvalidByteCode, 302
InvalidComposition, 302
LinkError, 302
LinkErrorToString, 304
LockShaderUniform, 304
NoError, 302
QueryInfoLog, 304
QueryReflectionDesc, 304
TooManyAttachments, 302
UnlockShaderUniform, 305
ValidateShaderComposition, 305
LLGL::ShaderProgramDescriptor, 306
computeShader, 306
fragmentShader, 306
geometryShader, 307
tessControlShader, 307
tessEvaluationShader, 307
vertexFormats, 307
vertexShader, 308
LLGL::ShaderReflectionDescriptor, 308
resourceViews, 309
streamOutputAttributes, 309
uniforms, 309
vertexAttributes, 309
LLGL::ShaderReflectionDescriptor::ResourceView, 286
arraySize, 286
constantBufferSize, 286
name, 287
slot, 287
stageFlags, 287
storageBufferType, 287
type, 287
LLGL::ShaderUniform, 309
SetUniform1f, 311
SetUniform1fv, 311
SetUniform1i, 311
SetUniform1iv, 311
SetUniform2f, 311
SetUniform2fv, 311
SetUniform2i, 311
SetUniform2iv, 311, 312
SetUniform2x2fv, 312
SetUniform3f, 312
SetUniform3fv, 312
SetUniform3i, 312
SetUniform3iv, 312
SetUniform3x3fv, 312
SetUniform4f, 312
SetUniform4fv, 312, 313
SetUniform4i, 313
SetUniform4iv, 313
SetUniform4x4fv, 313
LLGL::SrcImageDescriptor, 313
data, 314
dataSize, 314
dataType, 314
format, 314
SrcImageDescriptor, 314
LLGL::StageFlags, 315
AllGraphicsStages, 316
AllStages, 316
AllTessStages, 316
ComputeStage, 315
FragmentStage, 315
GeometryStage, 315
StorageUsage, 315
TessControlStage, 315
TessEvaluationStage, 315
VertexStage, 315
LLGL::StencilDescriptor, 316
back, 316
front, 316
testEnabled, 317
LLGL::StencilFaceDescriptor, 317
compareOp, 318
depthFailOp, 318
depthPassOp, 318
readMask, 318
reference, 318
stencilFailOp, 318
writeMask, 319
LLGL::StreamOutputAttribute, 321
components, 322
name, 322
operator=, 322
outputSlot, 322
semanticIndex, 322
startComponent, 322
stream, 323
StreamOutputAttribute, 322
LLGL::StreamOutputFormat, 323
AppendAttribute, 323
AppendAttributes, 324
attributes, 324
LLGL::Surface, 324
AdaptForVideoMode, 325

GetContentSize, 325
GetNativeHandle, 326
ResetPixelFormat, 326
LLGL::Texture, 327
GetType, 327
QueryDesc, 327
QueryMipExtent, 328
QueryResourceType, 328
Texture, 327
LLGL::TextureDescriptor, 329
arrayLayers, 330
extent, 330
flags, 330
format, 330
mipLevels, 331
samples, 331
type, 331
LLGL::TextureFlags, 331
ColorAttachmentUsage, 332
Default, 332
DepthStencilAttachmentUsage, 332
FixedSamples, 332
SampleUsage, 332
StorageUsage, 332
LLGL::TextureRegion, 333
extent, 333
mipLevel, 333
offset, 333
LLGL::Timer, 334
Create, 335
GetDeltaTime, 335
GetFrequency, 335
IsRunning, 335
MeasureTime, 335
Start, 336
Stop, 336
LLGL::UniformDescriptor, 336
location, 337
name, 337
size, 337
type, 337
LLGL::UninitializeTag, 337
LLGL::Version, 97
GetID, 97
GetMajor, 97
GetMinor, 97
GetRevision, 97
GetStatus, 98
GetString, 98
LLGL::VertexAttribute, 338
format, 340
GetSize, 339
instanceDivisor, 340
name, 340
offset, 340
operator=, 339
semanticIndex, 340
VertexAttribute, 338, 339
LLGL::VertexFormat, 341
AppendAttribute, 342
attributes, 343
inputSlot, 343
stride, 343
LLGL::VideoAdapterDescriptor, 344
name, 344
outputs, 344
vendor, 344
videoMemory, 344
LLGL::VideoModeDescriptor, 345
colorBits, 346
depthBits, 346
fullscreen, 346
resolution, 346
stencilBits, 346
swapChainSize, 347
LLGL::VideoOutputDescriptor, 347
displayModes, 348
LLGL::Viewport, 348
height, 350
maxDepth, 350
minDepth, 350
Viewport, 349
width, 350
x, 350
y, 350
LLGL::VsyncDescriptor, 351
enabled, 351
interval, 351
refreshRate, 351
LLGL::VulkanRendererConfiguration, 352
application, 352
minDeviceMemoryAllocationSize, 352
reduceDeviceMemoryFragmentation, 352
LLGL::Window, 353
AdaptForVideoMode, 355
AddEventListener, 355
Create, 355
GetBehavior, 355
GetDesc, 356
GetPosition, 356
GetSize, 356
GetTitle, 356
HasFocus, 356
IsShown, 356
OnProcessEvents, 356
PostChar, 356
PostDoubleClick, 356
PostGetFocus, 357
PostGlobalMotion, 357
PostKeyDown, 357
PostKeyUp, 357
PostLocalMotion, 357
PostLoseFocus, 357
PostQuit, 357
PostResize, 358
PostTimer, 358

PostWheelMotion, 358
ProcessEvents, 358
RemoveEventListener, 358
SetBehavior, 359
SetDesc, 359
SetPosition, 359
SetSize, 359
SetTitle, 359
Show, 359
LLGL::Window::EventListener, 171
∼EventListener, 172
OnChar, 172
OnDoubleClick, 172
OnGetFocus, 172
OnGlobalMotion, 172
OnKeyDown, 172
OnKeyUp, 173
OnLocalMotion, 173
OnLoseFocus, 173
OnProcessEvents, 173
OnQuit, 173
OnResize, 173
OnTimer, 174
OnWheelMotion, 174
Window, 174
LLGL::WindowBehavior, 360
disableClearOnResize, 360
moveAndResizeTimerID, 360
LLGL::WindowDescriptor, 361
acceptDropFiles, 361
borderless, 361
centered, 361
position, 362
preventForPowerSafe, 362
resizable, 362
size, 362
title, 363
visible, 363
windowContext, 363
LLGL_EXPORT
Export.h, 374
LLGL, 49
A, 75
ABGR, 74
ANDInverted, 78
ANDReverse, 78
AND, 78
ARGB, 74
Add, 67
AlwaysPass, 68
Any, 77
AnySamplesPassed, 81
AnySamplesPassedConservative, 81
AppendStructuredBuffer, 88
Apps, 76
AtomicCounter, 89
AttachmentLoadOp, 65
AttachmentStoreOp, 65
AttachmentType, 66
Attn, 77
B, 75
BC1RGBA, 73
BC1RGB, 73
BC2RGBA, 73
BC3RGBA, 73
BGRA8SInt, 73
BGRA8SNorm, 73
BGRA8UInt, 73
BGRA8UNorm, 73
BGRA8sRGB, 73
BGRA, 74
BGR, 74
Back, 69, 74
BinaryBuffer, 85
BinaryFile, 85
BlendArithmetic, 66
BlendFactor, 67
BlendOp, 67
Bool1, 89
Bool2, 89
Bool3, 89
Bool4, 89
Border, 84
BrowserBack, 77
BrowserFavorits, 77
BrowserForward, 77
BrowserHome, 77
BrowserRefresh, 77
BrowserSearch, 77
BrowserStop, 77
Buffer, 88
BufferType, 67
ByRegionNoWait, 82
ByRegionNoWaitInverted, 82
ByRegionWait, 82
ByRegionWaitInverted, 82
ByteAddressBuffer, 88
ByteBuffer, 64
C, 75
CPUAccess, 68
Cancel, 74
Capital, 74
CastColorValue, 90
CastColorValue< bool, bool >, 90
CastColorValue< double, double >, 90
CastColorValue< float, float >, 90
CastColorValue< std::uint8_t, std::uint8_t >, 90
Clamp, 84
Clear, 65, 74, 78
ClippingRange, 68
CodeFile, 85
CodeString, 85
Color, 66
ColorRGBAb, 64
ColorRGBAd, 65
ColorRGBAf, 65

ColorRGBAT, 65
ColorRGBAub, 65
ColorRGBb, 65
ColorRGBd, 65
ColorRGBf, 65
ColorRGBT, 65
ColorRGBub, 65
Comma, 77
CompareOp, 68
CompatibilityProfile, 78
CompressedRGBA, 74
CompressedRGB, 74
Compute, 86
Constant, 68
ConstantBuffer, 83
ConsumeStructuredBuffer, 88
Control, 74
Copy, 78
CopyInverted, 78
CoreProfile, 78
CrSel, 77
CullMode, 69
D, 75
D0, 75
D1, 75
D16UNorm, 73
D2, 75
D24UNormS8UInt, 73
D3, 75
D32Float, 73
D32FloatS8X24UInt, 73
D4, 75
D5, 75
D6, 75
D7, 75
D8, 75
D9, 75
DataType, 69
DecClamp, 87
DecWrap, 87
Delete, 75
Depth, 66, 74
DepthStencil, 66, 74
Disabled, 69, 78
Double1, 89
Double2, 89
Double2x2, 89
Double2x3, 89
Double2x4, 89
Double3, 89
Double3x2, 89
Double3x3, 89
Double3x4, 89
Double4, 89
Double4x2, 89
Double4x3, 89
Double4x4, 89
Down, 75
DstAlpha, 67
DstColor, 67
E, 75
ESProfile, 78
ESSL_100, 86
ESSL_300, 87
ESSL_310, 87
ESSL_320, 87
ESSL, 86
End, 75
Equal, 68
Equiv, 78
ErEOF, 77
ErrorType, 70
Escape, 74
ExSel, 77
Exe, 75
Exponent, 77
F, 75
F1, 76
F10, 76
F11, 76
F12, 76
F13, 76
F14, 76
F15, 76
F16, 76
F17, 76
F18, 76
F19, 76
F2, 76
F20, 76
F21, 76
F22, 77
F23, 77
F24, 77
F3, 76
F4, 76
F5, 76
F6, 76
F7, 76
F8, 76
F9, 76
Fill, 79
Float1, 89
Float16, 70
Float2, 89
Float2x2, 89
Float2x3, 89
Float2x4, 89
Float3, 89
Float32, 70
Float3x2, 89
Float3x3, 89
Float3x4, 89
Float4, 89
Float4x2, 89
Float4x3, 89

Float4x4, 89
Float64, 70
Format, 70
Fragment, 86
Front, 69
G, 75
GLSL_110, 86
GLSL_120, 86
GLSL_130, 86
GLSL_140, 86
GLSL_150, 86
GLSL_330, 86
GLSL_400, 86
GLSL_410, 86
GLSL_420, 86
GLSL_430, 86
GLSL_440, 86
GLSL_450, 86
GLSL_460, 86
GLSL, 86
Geometry, 86
GetExtentRatio, 90
GetPrimitiveTopologyPatchSize, 90
Greater, 68
GreaterEqual, 68
H, 75
HLSL_2_0, 87
HLSL_2_0a, 87
HLSL_2_0b, 87
HLSL_3_0, 87
HLSL_4_0, 87
HLSL_4_1, 87
HLSL_5_0, 87
HLSL_5_1, 87
HLSL, 87
Help, 75
Home, 75
I, 75
Image, 89
ImageFormat, 73
ImproperArgument, 90
ImproperState, 90
IncClamp, 87
IncWrap, 87
Index, 68
IndexBuffer, 82
Insert, 75
Int1, 89
Int16, 70
Int2, 89
Int3, 89
Int32, 70
Int4, 89
Int8, 70
InvBlendFactor, 67
InvDstAlpha, 67
InvDstColor, 67
InvSrc1Alpha, 67
InvSrc1Color, 67
InvSrcAlpha, 67
InvSrcColor, 67
InvalidArgument, 70
InvalidState, 70
Invert, 78, 87
IsPrimitiveTopologyPatches, 91
IsShaderSourceBinary, 91
IsShaderSourceCode, 91
J, 75
K, 75
Keep, 87
Key, 74
Keypad0, 76
Keypad1, 76
Keypad2, 76
Keypad3, 76
Keypad4, 76
Keypad5, 76
Keypad6, 76
Keypad7, 76
Keypad8, 76
Keypad9, 76
KeypadDecimal, 76
KeypadDivide, 76
KeypadMinus, 76
KeypadMultiply, 76
KeypadPlus, 76
KeypadSeparator, 76
L, 75
LButton, 74
LControl, 77
LMenu, 77
LShift, 77
LWin, 76
LaunchApp1, 77
LaunchApp2, 77
LaunchMail, 77
LaunchMediaSelect, 77
Left, 75
Less, 68
LessEqual, 68
LineList, 79
LineListAdjacency, 79
LineLoop, 79
LineStrip, 79
LineStripAdjacency, 79
Linear, 85
Lines, 81
Load, 65
LogicOp, 77
LowerLeft, 85
M, 75
MButton, 74
Max, 67
MaxColorValue, 91
MaxColorValue< bool >, 91
MaxColorValue< std::uint8_t >, 91

MediaNextTrack, 77
MediaPlayPause, 77
MediaPrevTrack, 77
MediaStop, 77
Menu, 74
Metal, 87
Metal_1_0, 87
Metal_1_1, 87
Metal_1_2, 87
Min, 67
Minus, 77
MinusOneToOne, 68
Mirror, 83
MirrorOnce, 84
N, 75
NAND, 78
NOR, 78
Nearest, 85
NeverPass, 68
NoName, 77
NoOp, 78
NoWait, 82
NoWaitInverted, 82
NotEqual, 68
NumLock, 77
O, 75
OEMClear, 77
ORInverted, 78
ORReverse, 78
One, 67
OpenGLContextProfile, 78
operator!=, 91, 92
operator∗, 92
operator+, 92
operator-, 92
operator/, 92
operator==, 92, 93
OR, 78
P, 75
PA1, 77
PageDown, 74
PageUp, 74
Patches1, 80
Patches10, 80
Patches11, 80
Patches12, 80
Patches13, 80
Patches14, 80
Patches15, 80
Patches16, 80
Patches17, 80
Patches18, 80
Patches19, 80
Patches2, 80
Patches20, 80
Patches21, 80
Patches22, 80
Patches23, 80
Patches24, 80
Patches25, 80
Patches26, 80
Patches27, 80
Patches28, 80
Patches29, 80
Patches3, 80
Patches30, 80
Patches31, 80
Patches32, 80
Patches4, 80
Patches5, 80
Patches6, 80
Patches7, 80
Patches8, 80
Patches9, 80
Pause, 74
Period, 77
PipelineStatistics, 81
Play, 77
Plus, 77
PointList, 79
PointlessOperation, 90
Points, 79, 81
PolygonMode, 78
PrimitiveTopology, 79
PrimitiveType, 80
Print, 75
Q, 75
QueryType, 81
R, 74, 75
R16Float, 71
R16SInt, 71
R16SNorm, 71
R16UInt, 71
R16UNorm, 71
R32Float, 71
R32SInt, 71
R32UInt, 71
R64Float, 72
R8SInt, 71
R8SNorm, 71
R8UInt, 71
R8UNorm, 71
RButton, 74
RControl, 77
RG16Float, 71
RG16SInt, 71
RG16SNorm, 71
RG16UInt, 71
RG16UNorm, 71
RG32Float, 71
RG32SInt, 71
RG32UInt, 71
RG64Float, 72
RG8SInt, 71
RG8SNorm, 71
RG8UInt, 71

RG8UNorm, 71
RGB16Float, 72
RGB16SInt, 72
RGB16SNorm, 72
RGB16UInt, 72
RGB16UNorm, 72
RGB32Float, 72
RGB32SInt, 72
RGB32UInt, 72
RGB64Float, 72
RGB8SInt, 72
RGB8SNorm, 71
RGB8UInt, 71
RGB8UNorm, 71
RGBA16Float, 72
RGBA16SInt, 72
RGBA16SNorm, 72
RGBA16UInt, 72
RGBA16UNorm, 72
RGBA32Float, 72
RGBA32SInt, 72
RGBA32UInt, 72
RGBA64Float, 73
RGBA8SInt, 72
RGBA8SNorm, 72
RGBA8UInt, 72
RGBA8UNorm, 72
RGBA, 74
RGB, 74
RMenu, 77
RShift, 77
RWBuffer, 88
RWByteAddressBuffer, 88
RWStructuredBuffer, 88
RWin, 76
ReadOnly, 69
ReadWrite, 69
RenderConditionMode, 81
Repeat, 83
Replace, 87
ResourceType, 82
Return, 74
RevSubtract, 67
RG, 74
Right, 75
S, 75
SPIRV_100, 87
SPIRV, 87
Sampler, 83, 89
SamplerAddressMode, 83
SamplerFilter, 84
SamplesPassed, 81
ScreenOrigin, 85
ScrollLock, 77
Select, 75
Set, 78
ShaderSourceType, 85
ShaderType, 85
ShadingLanguage, 86
Shift, 74
Sleep, 76
Snapshot, 75
Space, 74
Src1Alpha, 67
Src1Color, 67
SrcAlpha, 67
SrcAlphaSaturate, 67
SrcColor, 67
Stencil, 66
StencilOp, 87
Storage, 68
StorageBuffer, 83
StorageBufferType, 87
Store, 66
StreamOutOverflow, 81
StreamOutPrimitivesWritten, 81
StreamOutput, 68
StreamOutputBuffer, 83
StructuredBuffer, 88
Subtract, 67
T, 75
Tab, 74
TessControl, 86
TessEvaluation, 86
Texture, 83
Texture1DArray, 88
Texture1D, 88
Texture2DArray, 88
Texture2DMSArray, 88
Texture2DMS, 88
Texture2D, 88
Texture3D, 88
TextureCube, 88
TextureCubeArray, 88
TextureType, 88
TimeElapsed, 81
TriangleFan, 79
TriangleList, 79
TriangleListAdjacency, 80
TriangleStrip, 79
TriangleStripAdjacency, 80
Triangles, 81
U, 75
UInt1, 89
UInt16, 70
UInt2, 89
UInt3, 89
UInt32, 70
UInt4, 89
UInt8, 70
Undefined, 65, 66, 71, 82, 86, 88, 89
UndefinedBehavior, 70
UniformLocation, 65
UniformType, 88
UnsupportedFeature, 70
Up, 75

UpperLeft, 85
V, 75
ValidateRenderingCaps, 93
VersionBitmask, 87
Vertex, 68, 86
VertexBuffer, 82
VolumeDown, 77
VolumeMute, 77
VolumeUp, 77
W, 76
Wait, 82
WaitInverted, 82
WarningType, 89
Wireframe, 79
WriteDiscard, 69
WriteOnly, 69
X, 76
XButton1, 74
XButton2, 74
XOR, 78
Y, 76
Z, 76
Zero, 67, 87
ZeroToOne, 68
Zoom, 77

LMenu
LLGL, 77
LShift
LLGL, 77
LWin
LLGL, 76
LaunchApp1
LLGL, 77
LaunchApp2
LLGL, 77
LaunchMail
LLGL, 77
LaunchMediaSelect
LLGL, 77
Left
LLGL, 75
Less
LLGL, 68
LessEqual
LLGL, 68
limits
LLGL::RenderingCapabilities, 238
LineList
LLGL, 79
LineListAdjacency
LLGL, 79
LineLoop
LLGL, 79
LineStrip
LLGL, 79
LineStripAdjacency
LLGL, 79
lineWidth
LLGL::RasterizerDescriptor, 225
lineWidthRange
LLGL::RenderingLimits, 247
Linear
LLGL, 85
Lines
LLGL, 81
LinkError
LLGL::ShaderProgram, 302
LinkErrorToString
LLGL::ShaderProgram, 304
LinuxNativeHandle.h, 382
Load
LLGL::RenderSystem, 267
LLGL, 65
loadOp
LLGL::AttachmentFormatDescriptor, 105
location
LLGL::UniformDescriptor, 337
LockShaderUniform
LLGL::ShaderProgram, 304
Log.h, 383
LogicOp
LLGL, 77
logicOp
LLGL::BlendDescriptor, 109
LowerLeft
LLGL, 85

M
LLGL, 75
MButton
LLGL, 74
MacOSNativeHandle.h, 384
magFilter
LLGL::SamplerDescriptor, 291
majorVersion
LLGL::ProfileOpenGLDescriptor, 218
MapBuffer
LLGL::RenderSystem, 268
MapReadAccess
LLGL::BufferFlags, 116
MapReadWriteAccess
LLGL::BufferFlags, 116
MapWriteAccess
LLGL::BufferFlags, 116
Max
LLGL, 67
max1DTextureSize
LLGL::RenderingLimits, 247
max2DTextureSize
LLGL::RenderingLimits, 248
max3DTextureSize
LLGL::RenderingLimits, 248
maxAnisotropy
LLGL::RenderingLimits, 248
LLGL::SamplerDescriptor, 291
maxBufferSize
LLGL::RenderingLimits, 248

maxColorAttachments
LLGL::RenderingLimits, 248
MaxColorValue
LLGL, 91
MaxColorValue< bool >
LLGL, 91
MaxColorValue< std::uint8_t >
LLGL, 91
maxComputeShaderWorkGroupSize
LLGL::RenderingLimits, 249
maxComputeShaderWorkGroups
LLGL::RenderingLimits, 249
maxConstantBufferSize
LLGL::RenderingLimits, 249
maxCubeTextureSize
LLGL::RenderingLimits, 249
maxDepth
LLGL::Viewport, 350
maxLOD
LLGL::SamplerDescriptor, 292
maxPatchVertices
LLGL::RenderingLimits, 249
maxTextureArrayLayers
LLGL::RenderingLimits, 250
maxViewportSize
LLGL::RenderingLimits, 250
maxViewports
LLGL::RenderingLimits, 250
MeasureTime
LLGL::Timer, 335
MediaNextTrack
LLGL, 77
MediaPlayPause
LLGL, 77
MediaPrevTrack
LLGL, 77
MediaStop
LLGL, 77
Menu
LLGL, 74
Message
LLGL::RenderingDebugger::Message, 207
Metal
LLGL::RendererID, 235
LLGL, 87
Metal_1_0
LLGL, 87
Metal_1_1
LLGL, 87
Metal_1_2
LLGL, 87
Min
LLGL, 67
minDepth
LLGL::Viewport, 350
minDeviceMemoryAllocationSize
LLGL::VulkanRendererConfiguration, 352
minFilter
LLGL::SamplerDescriptor, 292
minLOD
LLGL::SamplerDescriptor, 292
minorVersion
LLGL::ProfileOpenGLDescriptor, 218
Minus
LLGL, 77
MinusOneToOne
LLGL, 68
mipLevel
LLGL::AttachmentDescriptor, 103
LLGL::TextureRegion, 333
mipLevels
LLGL::TextureDescriptor, 331
mipMapFilter
LLGL::SamplerDescriptor, 292
mipMapLODBias
LLGL::SamplerDescriptor, 292
mipMapping
LLGL::SamplerDescriptor, 292
mipMapsGenerations
LLGL::FrameProfile, 182
Mirror
LLGL, 83
MirrorOnce
LLGL, 84
MirrorXYPlane
LLGL::Image, 195
MirrorXZPlane
LLGL::Image, 195
MirrorYZPlane
LLGL::Image, 195
moduleName
LLGL::RenderSystemDescriptor, 276
moveAndResizeTimerID
LLGL::WindowBehavior, 360
multiSampling
LLGL::RasterizerDescriptor, 225
LLGL::RenderContextDescriptor, 233
LLGL::RenderTargetDescriptor, 282
MultiSamplingDescriptor
LLGL::MultiSamplingDescriptor, 208

N
LLGL, 75
NAND
LLGL, 78
NOR
LLGL, 78
name
LLGL::ShaderReflectionDescriptor::ResourceView, 287
LLGL::StreamOutputAttribute, 322
LLGL::UniformDescriptor, 337
LLGL::VertexAttribute, 340
LLGL::VideoAdapterDescriptor, 344
NativeHandle.h, 384
Nearest
LLGL, 85

NeverPass
LLGL, 68
NextProfile
LLGL::RenderingProfiler, 251
NoError
LLGL::ShaderProgram, 302
NoName
LLGL, 77
NoOp
LLGL, 78
NoWait
LLGL, 82
NoWaitInverted
LLGL, 82
NonCopyable
LLGL::NonCopyable, 213
NonCopyable.h, 384
NotEqual
LLGL, 68
NumLock
LLGL, 77
NumMipLevels
Texture utility functions to determine texture dimension and buffer sizes., 39
numNativeBuffers
LLGL::CommandBufferDescriptor, 152
numQueries
LLGL::QueryHeapDescriptor, 220

O
LLGL, 75
O1
LLGL::ShaderCompileFlags, 297
O2
LLGL::ShaderCompileFlags, 297
O3
LLGL::ShaderCompileFlags, 297
OEMClear
LLGL, 77
ORInverted
LLGL, 78
ORReverse
LLGL, 78
offset
LLGL::TextureRegion, 333
LLGL::VertexAttribute, 340
Offset2D
LLGL::Offset2D, 214
Offset3D
LLGL::Offset3D, 215
OnChar
LLGL::Input, 204
LLGL::Window::EventListener, 172
OnDoubleClick
LLGL::Input, 205
LLGL::Window::EventListener, 172
OnError
LLGL::RenderingDebugger, 240
OnGetFocus
LLGL::Window::EventListener, 172
OnGlobalMotion
LLGL::Input, 205
LLGL::Window::EventListener, 172
OnKeyDown
LLGL::Input, 205
LLGL::Window::EventListener, 172
OnKeyUp
LLGL::Input, 205
LLGL::Window::EventListener, 173
OnLocalMotion
LLGL::Input, 205
LLGL::Window::EventListener, 173
OnLoseFocus
LLGL::Input, 205
LLGL::Window::EventListener, 173
OnProcessEvents
LLGL::Canvas, 118
LLGL::Canvas::EventListener, 170
LLGL::Input, 205
LLGL::Window, 356
LLGL::Window::EventListener, 173
OnQuit
LLGL::Window::EventListener, 173
OnResize
LLGL::Window::EventListener, 173
OnSetVideoMode
LLGL::RenderContext, 229
OnSetVsync
LLGL::RenderContext, 230
OnTimer
LLGL::Window::EventListener, 174
OnWarning
LLGL::RenderingDebugger, 240
OnWheelMotion
LLGL::Input, 206
LLGL::Window::EventListener, 174
One
LLGL, 67
OpenGLContextProfile
LLGL, 78
OpenGLES1
LLGL::RendererID, 235
OpenGLES2
LLGL::RendererID, 235
OpenGLES3
LLGL::RendererID, 236
OpenGL
LLGL::RendererID, 235
operator!=
Global operators for basic data structures., 22, 23
LLGL, 91, 92
operator∗
LLGL, 92
operator∗=
LLGL::Color, 124
LLGL::Color< T, 3u >, 128
LLGL::Color< T, 4u >, 133

operator+
Global operators for basic data structures., 23
LLGL, 92
operator+=
LLGL::Color, 124
LLGL::Color< T, 3u >, 128
LLGL::Color< T, 4u >, 133
operator-
Global operators for basic data structures., 23, 24
LLGL::Color, 124
LLGL::Color< T, 3u >, 128
LLGL::Color< T, 4u >, 133
LLGL, 92
operator-=
LLGL::Color, 124
LLGL::Color< T, 3u >, 128
LLGL::Color< T, 4u >, 133
operator/
LLGL, 92
operator/=
LLGL::Color, 124
LLGL::Color< T, 3u >, 128
LLGL::Color< T, 4u >, 133
operator=
LLGL::AttachmentClear, 101
LLGL::Image, 195, 196
LLGL::IndexFormat, 201
LLGL::NonCopyable, 213
LLGL::RenderSystemDescriptor, 276
LLGL::RenderingDebugger::Message, 208
LLGL::ShaderDescriptor, 299
LLGL::StreamOutputAttribute, 322
LLGL::VertexAttribute, 339
operator==
Global operators for basic data structures., 24
LLGL, 92, 93
operator[]
LLGL::Color, 124, 125
LLGL::Color< T, 3u >, 128, 129
LLGL::Color< T, 4u >, 133, 134
OR
LLGL, 78
originLowerLeft
LLGL::OpenGLDependentStateDescriptor, 216
outputSlot
LLGL::StreamOutputAttribute, 322
outputs
LLGL::VideoAdapterDescriptor, 344

P
LLGL, 75
PA1
LLGL, 77
PageDown
LLGL, 74
PageUp
LLGL, 74
parentView
LLGL::NativeContextHandle, 210
parentWindow
LLGL::NativeContextHandle, 210
Patches1
LLGL, 80
Patches10
LLGL, 80
Patches11
LLGL, 80
Patches12
LLGL, 80
Patches13
LLGL, 80
Patches14
LLGL, 80
Patches15
LLGL, 80
Patches16
LLGL, 80
Patches17
LLGL, 80
Patches18
LLGL, 80
Patches19
LLGL, 80
Patches2
LLGL, 80
Patches20
LLGL, 80
Patches21
LLGL, 80
Patches22
LLGL, 80
Patches23
LLGL, 80
Patches24
LLGL, 80
Patches25
LLGL, 80
Patches26
LLGL, 80
Patches27
LLGL, 80
Patches28
LLGL, 80
Patches29
LLGL, 80
Patches3
LLGL, 80
Patches30
LLGL, 80
Patches31
LLGL, 80
Patches32
LLGL, 80
Patches4
LLGL, 80
Patches5
LLGL, 80

Patches6
LLGL, 80
Patches7
LLGL, 80
Patches8
LLGL, 80
Patches9
LLGL, 80
Pause
LLGL, 74
Performance
LLGL::Log, 96
Period
LLGL, 77
pipelineLayout
LLGL::ComputePipelineDescriptor, 161
LLGL::GraphicsPipelineDescriptor, 187
LLGL::ResourceHeapDescriptor, 285
PipelineLayout.h, 385
PipelineLayoutDesc
Global utility functions, especially to fill descriptor structures., 42
PipelineLayoutFlags.h, 385
PipelineStatistics
LLGL, 81
Platform.h, 385
Play
LLGL, 77
Plus
LLGL, 77
PointList
LLGL, 79
PointlessOperation
LLGL, 90
Points
LLGL, 79, 81
PolygonMode
LLGL, 78
polygonMode
LLGL::RasterizerDescriptor, 225
position
LLGL::WindowDescriptor, 362
PostChar
LLGL::Window, 356
PostDoubleClick
LLGL::Window, 356
PostError
LLGL::RenderingDebugger, 240
PostGetFocus
LLGL::Window, 357
PostGlobalMotion
LLGL::Window, 357
PostKeyDown
LLGL::Window, 357
PostKeyUp
LLGL::Window, 357
PostLocalMotion
LLGL::Window, 357
PostLoseFocus
LLGL::Window, 357
PostQuit
LLGL::Window, 357
PostReport
LLGL::Log, 96
PostResize
LLGL::Window, 358
PostTimer
LLGL::Window, 358
PostWarning
LLGL::RenderingDebugger, 241
PostWheelMotion
LLGL::Window, 358
Present
LLGL::RenderContext, 230
preventForPowerSafe
LLGL::WindowDescriptor, 362
PrimitiveTopology
LLGL, 79
primitiveTopology
LLGL::GraphicsPipelineDescriptor, 188
PrimitiveType
LLGL, 80
Print
LLGL, 75
ProcessEvents
LLGL::Canvas, 119
LLGL::Window, 358
profile
LLGL::ShaderDescriptor, 299
profileOpenGL
LLGL::RenderContextDescriptor, 234
Ptr
LLGL::Color, 125
LLGL::Color< T, 3u >, 129
LLGL::Color< T, 4u >, 134

Q
LLGL, 75
QueryColorFormat
LLGL::RenderContext, 230
QueryDepthStencilFormat
LLGL::RenderContext, 230
QueryDesc
LLGL::Texture, 327
QueryDstDesc
LLGL::Image, 196
QueryHeap
LLGL::QueryHeap, 219
QueryHeap.h, 385
QueryHeapFlags.h, 386
QueryInfoLog
LLGL::Shader, 296
LLGL::ShaderProgram, 304
QueryList
LLGL::Display, 166
QueryMipExtent
LLGL::Texture, 328

QueryPrimary
LLGL::Display, 166
QueryReflectionDesc
LLGL::ShaderProgram, 304
QueryResourceType
LLGL::Buffer, 112
LLGL::Resource, 283
LLGL::Sampler, 290
LLGL::Texture, 328
QueryResult
LLGL::CommandQueue, 157
querySections
LLGL::FrameProfile, 183
QuerySrcDesc
LLGL::Image, 196
QuerySupportedDisplayModes
LLGL::Display, 166
QueryType
LLGL, 81

R
LLGL, 74, 75
r
LLGL::Color< T, 3u >, 130
LLGL::Color< T, 4u >, 135
R16Float
LLGL, 71
R16SInt
LLGL, 71
R16SNorm
LLGL, 71
R16UInt
LLGL, 71
R16UNorm
LLGL, 71
R32Float
LLGL, 71
R32SInt
LLGL, 71
R32UInt
LLGL, 71
R64Float
LLGL, 72
R8SInt
LLGL, 71
R8SNorm
LLGL, 71
R8UInt
LLGL, 71
R8UNorm
LLGL, 71
RButton
LLGL, 74
RControl
LLGL, 77
RG16Float
LLGL, 71
RG16SInt
LLGL, 71
RG16SNorm
LLGL, 71
RG16UInt
LLGL, 71
RG16UNorm
LLGL, 71
RG32Float
LLGL, 71
RG32SInt
LLGL, 71
RG32UInt
LLGL, 71
RG64Float
LLGL, 72
RG8SInt
LLGL, 71
RG8SNorm
LLGL, 71
RG8UInt
LLGL, 71
RG8UNorm
LLGL, 71
RGB16Float
LLGL, 72
RGB16SInt
LLGL, 72
RGB16SNorm
LLGL, 72
RGB16UInt
LLGL, 72
RGB16UNorm
LLGL, 72
RGB32Float
LLGL, 72
RGB32SInt
LLGL, 72
RGB32UInt
LLGL, 72
RGB64Float
LLGL, 72
RGB8SInt
LLGL, 72
RGB8SNorm
LLGL, 71
RGB8UInt
LLGL, 71
RGB8UNorm
LLGL, 71
RGBA16Float
LLGL, 72
RGBA16SInt
LLGL, 72
RGBA16SNorm
LLGL, 72
RGBA16UInt
LLGL, 72
RGBA16UNorm
LLGL, 72

RGBA32Float
LLGL, 72
RGBA32SInt
LLGL, 72
RGBA32UInt
LLGL, 72
RGBA64Float
LLGL, 73
RGBA8SInt
LLGL, 72
RGBA8SNorm
LLGL, 72
RGBA8UInt
LLGL, 72
RGBA8UNorm
LLGL, 72
RGBA
LLGL, 74
RGB
LLGL, 74
RMenu
LLGL, 77
RShift
LLGL, 77
RWBuffer
LLGL, 88
RWByteAddressBuffer
LLGL, 88
RWStructuredBuffer
LLGL, 88
RWin
LLGL, 76
rasterizer
LLGL::GraphicsPipelineDescriptor, 188
readMask
LLGL::StencilFaceDescriptor, 318
ReadOnly
LLGL, 69
ReadPixels
LLGL::Image, 196
ReadTexture
LLGL::RenderSystem, 268
ReadWrite
LLGL, 69
reduceDeviceMemoryFragmentation
LLGL::VulkanRendererConfiguration, 352
reference
LLGL::StencilFaceDescriptor, 318
refreshRate
LLGL::DisplayModeDescriptor, 168
LLGL::VsyncDescriptor, 351
Release
LLGL::Image, 197
LLGL::RenderSystem, 269–271
RemoveEventListener
LLGL::Canvas, 119
LLGL::Window, 358
renderCondition
LLGL::QueryHeapDescriptor, 220
RenderConditionMode
LLGL, 81
renderConditionSections
LLGL::FrameProfile, 183
RenderContext
LLGL::RenderContext, 228
RenderContext.h, 386
RenderContextFlags.h, 387
renderPass
LLGL::GraphicsPipelineDescriptor, 188
LLGL::RenderTargetDescriptor, 282
RenderPass.h, 389
RenderPassDesc
Global utility functions, especially to fill descriptor structures., 44
RenderPassFlags.h, 389
renderPassSections
LLGL::FrameProfile, 183
RenderSystem
LLGL::RenderSystem, 257
RenderSystem.h, 390
RenderSystemChild.h, 390
RenderSystemDescriptor
LLGL::RenderSystemDescriptor, 275
RenderSystemFlags.h, 391
RenderTarget.h, 392
RenderTargetFlags.h, 393
rendererConfig
LLGL::RenderSystemDescriptor, 276
rendererConfigSize
LLGL::RenderSystemDescriptor, 276
rendererName
LLGL::RendererInfo, 237
RenderingDebugger
LLGL::RenderingDebugger::Message, 208
RenderingDebugger.h, 388
RenderingProfiler.h, 388
Repeat
LLGL, 83
Replace
LLGL, 87
ReportCallback
LLGL::Log, 95
ReportType
LLGL::Log, 95
Reserved
LLGL::RendererID, 236
Reset
LLGL::Image, 197
ResetDisplayMode
LLGL::Display, 166
ResetPixelFormat
LLGL::Surface, 326
ResetResourceSlots
LLGL::CommandBufferExt, 154
resizable
LLGL::WindowDescriptor, 362

Resize
    LLGL::Image, 197, 198
resolution
    LLGL::DisplayModeDescriptor, 168
    LLGL::RenderTargetDescriptor, 282
    LLGL::VideoModeDescriptor, 346
resource
    LLGL::ResourceViewDescriptor, 289
Resource.h, 393
ResourceFlags.h, 394
ResourceHeap.h, 394
ResourceHeapFlags.h, 394
ResourceType
    LLGL, 82
ResourceViewDescriptor
    LLGL::ResourceViewDescriptor, 288
resourceViews
    LLGL::ResourceHeapDescriptor, 285
    LLGL::ShaderReflectionDescriptor, 309
Return
    LLGL, 74
RevSubtract
    LLGL, 67
RG
    LLGL, 74
Right
    LLGL, 75

S
    LLGL, 75
SPIRV_100
    LLGL, 87
SPIRV
    LLGL, 87
SampleCount
    LLGL::MultiSamplingDescriptor, 209
sampleMask
    LLGL::MultiSamplingDescriptor, 209
SampleUsage
    LLGL::TextureFlags, 332
Sampler
    LLGL::Sampler, 290
    LLGL, 83, 89
Sampler.h, 395
SamplerAddressMode
    LLGL, 83
samplerBindings
    LLGL::FrameProfile, 183
SamplerFilter
    LLGL, 84
SamplerFlags.h, 395
samples
    LLGL::MultiSamplingDescriptor, 209
    LLGL::TextureDescriptor, 331
SamplesPassed
    LLGL, 81
Scissor
    LLGL::Scissor, 293
scissorTestEnabled
    LLGL::RasterizerDescriptor, 226
scissors
    LLGL::GraphicsPipelineDescriptor, 188
screen
    LLGL::NativeContextHandle, 210
ScreenOrigin
    LLGL, 85
screenOrigin
    LLGL::RenderingCapabilities, 238
ScrollLock
    LLGL, 77
Select
    LLGL, 75
semanticIndex
    LLGL::StreamOutputAttribute, 322
    LLGL::VertexAttribute, 340
Set
    LLGL, 78
SetBehavior
    LLGL::Window, 359
SetClearColor
    LLGL::CommandBuffer, 145
SetClearDepth
    LLGL::CommandBuffer, 145
SetClearStencil
    LLGL::CommandBuffer, 145
SetComputePipeline
    LLGL::CommandBuffer, 146
SetComputeResourceHeap
    LLGL::CommandBuffer, 146
SetConfiguration
    LLGL::RenderSystem, 271
SetConstantBuffer
    LLGL::CommandBufferExt, 154
SetDesc
    LLGL::Window, 359
SetDisplayMode
    LLGL::Display, 166
SetDisplayModeByVideoMode
    LLGL::RenderContext, 231
SetGraphicsAPIDependentState
    LLGL::CommandBuffer, 146
SetGraphicsPipeline
    LLGL::CommandBuffer, 147
SetGraphicsResourceHeap
    LLGL::CommandBuffer, 147
SetIndexBuffer
    LLGL::CommandBuffer, 148
SetOrCreateSurface
    LLGL::RenderContext, 231
SetPosition
    LLGL::Window, 359
SetRendererInfo
    LLGL::RenderSystem, 271
SetRenderingCaps
    LLGL::RenderSystem, 271
SetReportCallback
    LLGL::Log, 96

SetReportCallbackStd
LLGL::Log, 96
SetSampler
LLGL::CommandBufferExt, 154
SetScissor
LLGL::CommandBuffer, 148
SetScissors
LLGL::CommandBuffer, 148
SetSize
LLGL::Window, 359
SetSource
LLGL::RenderingDebugger, 241
SetStorageBuffer
LLGL::CommandBufferExt, 155
SetStreamOutputBuffer
LLGL::CommandBuffer, 149
SetStreamOutputBufferArray
LLGL::CommandBuffer, 149
SetTexture
LLGL::CommandBufferExt, 155
SetTitle
LLGL::Canvas, 119
LLGL::Window, 359
SetUniform1f
LLGL::ShaderUniform, 311
SetUniform1fv
LLGL::ShaderUniform, 311
SetUniform1i
LLGL::ShaderUniform, 311
SetUniform1iv
LLGL::ShaderUniform, 311
SetUniform2f
LLGL::ShaderUniform, 311
SetUniform2fv
LLGL::ShaderUniform, 311
SetUniform2i
LLGL::ShaderUniform, 311
SetUniform2iv
LLGL::ShaderUniform, 311, 312
SetUniform2x2fv
LLGL::ShaderUniform, 312
SetUniform3f
LLGL::ShaderUniform, 312
SetUniform3fv
LLGL::ShaderUniform, 312
SetUniform3i
LLGL::ShaderUniform, 312
SetUniform3iv
LLGL::ShaderUniform, 312
SetUniform3x3fv
LLGL::ShaderUniform, 312
SetUniform4f
LLGL::ShaderUniform, 312
SetUniform4fv
LLGL::ShaderUniform, 312, 313
SetUniform4i
LLGL::ShaderUniform, 313
SetUniform4iv
LLGL::ShaderUniform, 313
SetUniform4x4fv
LLGL::ShaderUniform, 313
SetVertexBuffer
LLGL::CommandBuffer, 150
SetVertexBufferArray
LLGL::CommandBuffer, 150
SetVideoMode
LLGL::RenderContext, 231
SetViewport
LLGL::CommandBuffer, 150
SetViewports
LLGL::CommandBuffer, 150
SetVsync
LLGL::RenderContext, 232
Shader
LLGL::Shader, 295
Shader.h, 396
ShaderDescFromFile
Global utility functions, especially to fill descriptor structures., 44
ShaderDescriptor
LLGL::ShaderDescriptor, 298
ShaderFlags.h, 396
shaderProgram
LLGL::ComputePipelineDescriptor, 161
LLGL::GraphicsPipelineDescriptor, 188
ShaderProgram.h, 397
ShaderProgramDesc
Global utility functions, especially to fill descriptor structures., 44, 45
ShaderProgramFlags.h, 397
ShaderSourceType
LLGL, 85
ShaderType
LLGL, 85
ShaderUniform.h, 398
ShaderUniformFlags.h, 398
ShadingLanguage
LLGL, 86
shadingLanguageName
LLGL::RendererInfo, 237
shadingLanguages
LLGL::RenderingCapabilities, 238
ShareSurfaceAndConfig
LLGL::RenderContext, 232
Shift
LLGL, 74
Show
LLGL::Window, 359
ShowCursor
LLGL::Display, 167
size
LLGL::BufferDescriptor, 115
LLGL::UniformDescriptor, 337
LLGL::WindowDescriptor, 362
Sleep
LLGL, 76

slopeFactor
LLGL::DepthBiasDescriptor, 162
slot
LLGL::BindingDescriptor, 107
LLGL::ShaderReflectionDescriptor::ResourceView, 287
Snapshot
LLGL, 75
source
LLGL::ShaderDescriptor, 299
sourceSize
LLGL::ShaderDescriptor, 299
sourceType
LLGL::ShaderDescriptor, 300
Space
LLGL, 74
SplitFormat
Hardware format utility functions., 29
Src1Alpha
LLGL, 67
Src1Color
LLGL, 67
SrcAlpha
LLGL, 67
srcAlpha
LLGL::BlendTargetDescriptor, 111
SrcAlphaSaturate
LLGL, 67
SrcColor
LLGL, 67
srcColor
LLGL::BlendTargetDescriptor, 111
SrcImageDescriptor
LLGL::SrcImageDescriptor, 314
stageFlags
LLGL::BindingDescriptor, 107
LLGL::ShaderReflectionDescriptor::ResourceView, 287
Start
LLGL::Timer, 336
startComponent
LLGL::StreamOutputAttribute, 322
Stencil
LLGL::ClearFlags, 120
LLGL, 66
stencil
LLGL::ClearValue, 121
LLGL::GraphicsPipelineDescriptor, 189
stencilAttachment
LLGL::RenderPassDescriptor, 254
stencilBits
LLGL::VideoModeDescriptor, 346
stencilFailOp
LLGL::StencilFaceDescriptor, 318
StencilOp
LLGL, 87
Stop
LLGL::Timer, 336
Storage
LLGL, 68
StorageBuffer
LLGL, 83
storageBuffer
LLGL::BufferDescriptor, 115
storageBufferBindings
LLGL::FrameProfile, 183
StorageBufferDesc
Global utility functions, especially to fill descriptor structures., 45
StorageBufferType
LLGL, 87
storageBufferType
LLGL::ShaderReflectionDescriptor::ResourceView, 287
storageType
LLGL::BufferDescriptor::StorageBuffer, 320
StorageUsage
LLGL::StageFlags, 315
LLGL::TextureFlags, 332
Store
LLGL, 66
storeOp
LLGL::AttachmentFormatDescriptor, 106
stream
LLGL::StreamOutputAttribute, 323
StreamOutOverflow
LLGL, 81
StreamOutPrimitivesWritten
LLGL, 81
StreamOutput
LLGL, 68
streamOutput
LLGL::ShaderDescriptor, 300
StreamOutputAttribute
LLGL::StreamOutputAttribute, 322
StreamOutputAttribute.h, 399
streamOutputAttributes
LLGL::ShaderReflectionDescriptor, 309
StreamOutputBuffer
LLGL, 83
streamOutputBufferBindings
LLGL::FrameProfile, 184
StreamOutputFormat.h, 400
streamOutputSections
LLGL::FrameProfile, 184
stride
LLGL::BufferDescriptor::StorageBuffer, 320
LLGL::VertexFormat, 343
Strings.h, 400
StructuredBuffer
LLGL, 88
Submit
LLGL::CommandQueue, 158
Subtract
LLGL, 67
Surface.h, 401

Swap
LLGL::Image, 198
swapChainSize
LLGL::VideoModeDescriptor, 347
SwitchFullscreenMode
LLGL::RenderContext, 232

T
LLGL, 75
Tab
LLGL, 74
Tags.h, 401
targets
LLGL::BlendDescriptor, 109
TessControl
LLGL, 86
tessControlShader
LLGL::ShaderProgramDescriptor, 307
tessControlShaderInvocations
LLGL::QueryPipelineStatistics, 223
TessControlStage
LLGL::StageFlags, 315
TessEvaluation
LLGL, 86
tessEvaluationShader
LLGL::ShaderProgramDescriptor, 307
tessEvaluationShaderInvocations
LLGL::QueryPipelineStatistics, 223
TessEvaluationStage
LLGL::StageFlags, 315
testEnabled
LLGL::DepthDescriptor, 163
LLGL::StencilDescriptor, 317
Texture
LLGL::Texture, 327
LLGL, 83
texture
LLGL::AttachmentDescriptor, 104
Texture utility functions to determine texture dimension and buffer sizes., 38
IsArrayTexture, 38
IsCubeTexture, 38
IsMipMappedTexture, 38
IsMultiSampleTexture, 39
NumMipLevels, 39
TextureBufferSize, 40
TextureSize, 40
Texture.h, 401
Texture1DArray
LLGL, 88
Texture1DArrayDesc
Global utility functions, especially to fill descriptor structures., 45
Texture1DDesc
Global utility functions, especially to fill descriptor structures., 45
Texture1D
LLGL, 88
Texture2DArray
LLGL, 88
Texture2DArrayDesc
Global utility functions, especially to fill descriptor structures., 46
Texture2DDesc
Global utility functions, especially to fill descriptor structures., 46
Texture2DMSArray
LLGL, 88
Texture2DMSArrayDesc
Global utility functions, especially to fill descriptor structures., 46
Texture2DMSDesc
Global utility functions, especially to fill descriptor structures., 46
Texture2DMS
LLGL, 88
Texture2D
LLGL, 88
Texture3DDesc
Global utility functions, especially to fill descriptor structures., 46
Texture3D
LLGL, 88
textureBindings
LLGL::FrameProfile, 184
TextureBufferSize
Texture utility functions to determine texture dimension and buffer sizes., 40
textureCopies
LLGL::FrameProfile, 184
TextureCube
LLGL, 88
TextureCubeArray
LLGL, 88
TextureCubeArrayDesc
Global utility functions, especially to fill descriptor structures., 47
TextureCubeDesc
Global utility functions, especially to fill descriptor structures., 47
TextureFlags.h, 402
textureFormats
LLGL::RenderingCapabilities, 238
textureMappings
LLGL::FrameProfile, 184
textureReads
LLGL::FrameProfile, 185
TextureSize
Texture utility functions to determine texture dimension and buffer sizes., 40
TextureType
LLGL, 88
textureWrites
LLGL::FrameProfile, 185
threadCount
LLGL::RenderSystemConfiguration, 274
TimeElapsed

LLGL, 81
Timer.h, 403
title
LLGL::CanvasDescriptor, 120
LLGL::WindowDescriptor, 363
ToRGBA
LLGL::Color< T, 3u >, 129
ToRGB
LLGL::Color< T, 4u >, 134
ToString
Global type-to-string conversion functions., 36, 37
TooManyAttachments
LLGL::ShaderProgram, 302
TriangleFan
LLGL, 79
TriangleList
LLGL, 79
TriangleListAdjacency
LLGL, 80
TriangleStrip
LLGL, 79
TriangleStripAdjacency
LLGL, 80
Triangles
LLGL, 81
type
LLGL::AttachmentDescriptor, 104
LLGL::BindingDescriptor, 107
LLGL::BufferDescriptor, 115
LLGL::QueryHeapDescriptor, 220
LLGL::ShaderDescriptor, 300
LLGL::ShaderReflectionDescriptor::ResourceView, 287
LLGL::TextureDescriptor, 331
LLGL::UniformDescriptor, 337
Types.h, 403

U
LLGL, 75
UInt1
LLGL, 89
UInt16
LLGL, 70
UInt2
LLGL, 89
UInt3
LLGL, 89
UInt32
LLGL, 70
UInt4
LLGL, 89
UInt8
LLGL, 70
Undefined
LLGL::RendererID, 236
LLGL, 65, 66, 71, 82, 86, 88, 89
UndefinedBehavior
LLGL, 70
UniformLocation
LLGL, 65
UniformType
LLGL, 88
uniforms
LLGL::ShaderReflectionDescriptor, 309
Unload
LLGL::RenderSystem, 272
UnlockShaderUniform
LLGL::ShaderProgram, 305
UnmapBuffer
LLGL::RenderSystem, 272
UnsupportedFeature
LLGL, 70
Up
LLGL, 75
UpdateBuffer
LLGL::CommandBuffer, 151
UpperLeft
LLGL, 85
Utility.h, 404

V
LLGL, 75
ValidateMipResolution
LLGL::RenderTarget, 280
ValidateRenderingCaps
LLGL, 93
ValidateRenderingCapsFunc
Global type aliases to callback interfaces., 25
ValidateResolution
LLGL::RenderTarget, 280
ValidateShaderComposition
LLGL::ShaderProgram, 305
values
LLGL::FrameProfile, 185
vendor
LLGL::VideoAdapterDescriptor, 344
vendorName
LLGL::RendererInfo, 237
Version.h, 406
VersionBitmask
LLGL, 87
Vertex
LLGL, 68, 86
VertexAttribute
LLGL::VertexAttribute, 338, 339
VertexAttribute.h, 406
vertexAttributes
LLGL::ShaderReflectionDescriptor, 309
VertexBuffer
LLGL, 82
vertexBuffer
LLGL::BufferDescriptor, 115
vertexBufferBindings
LLGL::FrameProfile, 185
VertexBufferDesc
Global utility functions, especially to fill descriptor structures., 47
VertexFormat.h, 407

vertexFormats
LLGL::ShaderProgramDescriptor, 307
vertexShader
LLGL::ShaderProgramDescriptor, 308
vertexShaderInvocations
LLGL::QueryPipelineStatistics, 223
VertexStage
LLGL::StageFlags, 315
VideoAdapter.h, 407
videoMemory
LLGL::VideoAdapterDescriptor, 344
videoMode
LLGL::RenderContextDescriptor, 234
view
LLGL::NativeHandle, 211
Viewport
LLGL::Viewport, 349
viewports
LLGL::GraphicsPipelineDescriptor, 189
visible
LLGL::WindowDescriptor, 363
visual
LLGL::NativeContextHandle, 210
LLGL::NativeHandle, 211
VolumeDown
LLGL, 77
VolumeMute
LLGL, 77
VolumeUp
LLGL, 77
vsync
LLGL::RenderContextDescriptor, 234
Vulkan
LLGL::RendererID, 236

W
LLGL, 76
Wait
LLGL, 82
WaitFence
LLGL::CommandQueue, 158
WaitIdle
LLGL::CommandQueue, 159
WaitInverted
LLGL, 82
WarnError
LLGL::ShaderCompileFlags, 297
Warning
LLGL::Log, 95
WarningType
LLGL, 89
width
LLGL::Extent2D, 175
LLGL::Extent3D, 176
LLGL::Scissor, 294
LLGL::Viewport, 350
Win32NativeHandle.h, 408
Window
LLGL::Window::EventListener, 174
window
LLGL::NativeHandle, 211
Window.h, 408
windowContext
LLGL::WindowDescriptor, 363
WindowFlags.h, 409
Wireframe
LLGL, 79
WriteBuffer
LLGL::RenderSystem, 272
WriteDiscard
LLGL, 69
writeEnabled
LLGL::DepthDescriptor, 163
writeMask
LLGL::StencilFaceDescriptor, 319
WriteOnly
LLGL, 69
WritePixels
LLGL::Image, 198
WriteTexture
LLGL::RenderSystem, 272

X
LLGL, 76
x
LLGL::Offset2D, 214
LLGL::Offset3D, 215
LLGL::Scissor, 294
LLGL::Viewport, 350
XButton1
LLGL, 74
XButton2
LLGL, 74
XOR
LLGL, 78

Y
LLGL, 76
y
LLGL::Offset2D, 214
LLGL::Offset3D, 215
LLGL::Scissor, 294
LLGL::Viewport, 350

Z
LLGL, 76
z
LLGL::Offset3D, 215
Zero
LLGL, 67, 87
ZeroToOne
LLGL, 68
Zoom
LLGL, 77