LLGL

0.03 Beta

Generated by Doxygen 1.8.11

Contents

1 LLGL 0.03 Beta Documentation **1**

2 Todo List **3**

3 Module Index **5**

3.1 Modules 5

4 Namespace Index **7**

4.1 Namespace List 7

5 Hierarchical Index **9**

5.1 Class Hierarchy 9

6 Class Index **13**

6.1 Class List 13

7 File Index **19**

7.1 File List 19

8 Module Documentation **21**

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

8.1.1 Detailed Description 21

8.1.2 Function Documentation 21

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

8.2 Global operators for basic data structures. 22

8.2.1 Detailed Description 22

8.2.2 Function Documentation 22

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

8.5.2.1 GetFormatAttribs(const Format format) 28

8.5.2.2 GetMemoryFootprint(const Format format, std::uint32_t count) 29

8.5.2.3 GetMemoryFootprint(const ImageFormat imageFormat, const DataType data↩ Type, std::uint32_t count) 29

8.5.2.4 ImageFormatSize(const ImageFormat imageFormat) 29

8.5.2.5 IsCompressedFormat(const Format format) 30

8.5.2.6 IsCompressedFormat(const ImageFormat imageFormat) 30

8.5.2.7 IsDepthFormat(const Format format) 30

8.5.2.8 IsDepthStencilFormat(const Format format) 30

8.5.2.9 IsDepthStencilFormat(const ImageFormat imageFormat) 30

8.5.2.10 IsFloatFormat(const Format format) 31

8.5.2.11 IsIntegralFormat(const Format format) 31

8.5.2.12 IsNormalizedFormat(const Format format) 31

8.5.2.13 IsStencilFormat(const Format format) 31

8.6 Data type utility functions. 32

8.6.1 Detailed Description 32

8.6.2 Function Documentation 32

8.6.2.1 DataTypeSize(const DataType dataType) 32

8.6.2.2 IsFloatDataType(const DataType dataType) 32

8.6.2.3 IsIntDataType(const DataType dataType) 32

8.6.2.4 IsUIntDataType(const DataType dataType) 32

8.7 Image utility functions to classify and convert image data. 33

8.7.1 Detailed Description 33

8.7.2 Function Documentation 33

8.7.2.1 AllocateByteBuffer(std::size_t bufferSize) 33

8.7.2.2 AllocateByteBuffer(std::size_t bufferSize, UninitializeTag) 34

8.7.2.3 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, const Dst↩ ImageDescriptor &dstImageDesc, unsigned threadCount=0) 34

8.7.2.4 ConvertImageBuffer(const SrcImageDescriptor &srcImageDesc, ImageFormat dstFormat, DataType dstDataType, unsigned threadCount=0) 35

8.7.2.5 CopyImageBufferRegion(const DstImageDescriptor &dstImageDesc, const Offset3D &dstOffset, std::uint32_t dstRowStride, std::uint32_t dstSliceStride, const SrcImageDescriptor &srcImageDesc, const Offset3D &srcOffset, std::uint32_t srcRowStride, std::uint32_t srcSliceStride, const Extent3D &extent) . . 36

8.7.2.6 DecompressImageBufferToRGBA8UNorm(const SrcImageDescriptor &srcImageDesc, const Extent2D &extent, unsigned threadCount=0) 36

8.7.2.7 GenerateImageBuffer(ImageFormat format, DataType dataType, std::size_t imageSize, const ColorRGBAd &fillColor) 37

8.8 Buffer argument structures for indirect draw and compute commands. 38

8.8.1 Detailed Description 38

8.9 Global type-to-string conversion functions. 39

8.9.1 Detailed Description 39

8.9.2 Function Documentation 39

8.9.2.1 ToString(const ShaderType t) 39

8.9.2.2 ToString(const ErrorType t) 39

8.9.2.3 ToString(const WarningType t) 39

8.9.2.4 ToString(const ShadingLanguage t) 40

8.9.2.5 ToString(const Format t) 40

8.9.2.6 ToString(const TextureType t) 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 cpuAccessFlags=0) 42

8.10.2.2 IndexBufferDesc(uint64_t size, const Format format, long cpuAccessFlags=0) . . 42

8.10.2.3 PipelineLayoutDesc(const ShaderReflection &reflection) 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 StorageBufferDesc(uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite) 45

8.10.2.8 Texture1DArrayDesc(Format format, std::uint32_t width, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . 45

8.10.2.9 Texture1DDesc(Format format, std::uint32_t width, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 45

8.10.2.10 Texture2DArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 45

8.10.2.11 Texture2DDesc(Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 45

8.10.2.12 Texture2DMSArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 46

8.10.2.13 Texture2DMSDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 46

8.10.2.14 Texture3DDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 46

8.10.2.15 TextureCubeArrayDesc(Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 46

8.10.2.16 TextureCubeDesc(Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment" | BindFlags::Sampled)) . . . 46

8.10.2.17 VertexBufferDesc(uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0) . . . 46

8.11 Texture utility functions to determine texture dimension and buffer sizes. . . . 47

8.11.1 Detailed Description . . . 48

8.11.2 Function Documentation . . . 48

8.11.2.1 GetMemoryFootprint(const TextureType type, const Format format, const Extent3D &extent, const TextureSubresource &subresource) . . . 48

8.11.2.2 GetMipExtent(const TextureType type, const Extent3D &extent, std::uint32_t mipLevel) . . . 49

8.11.2.3 GetMipExtent(const TextureDescriptor &textureDesc, std::uint32_t mipLevel=0) . 49

8.11.2.4 IsArrayTexture(const TextureType type) . . . 50

8.11.2.5 IsCubeTexture(const TextureType type) . . . 50

8.11.2.6 IsMipMappedTexture(const TextureDescriptor &textureDesc) . . . 50

8.11.2.7 IsMultiSampleTexture(const TextureType type) . . . 50

8.11.2.8 IsTextureSwizzleIdentity(const TextureSwizzleRGBA &swizzle) . . . 51

8.11.2.9 NumMipDimensions(const TextureType type) . . . 51

8.11.2.10 NumMipLevels(std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1) 51

8.11.2.11 NumMipLevels(const TextureType type, const Extent3D &extent) . . . 52

8.11.2.12 NumMipLevels(const TextureDescriptor &textureDesc) . . . 52

8.11.2.13 NumMipTexels(const TextureType type, const Extent3D &extent, std::uint32_t mipLevel) . . . 52

8.11.2.14 NumMipTexels(const TextureType type, const Extent3D &extent, const TextureSubresource &subresource) . . . 53

8.11.2.15 NumMipTexels(const TextureDescriptor &textureDesc, std::uint32_t mipLevel=~0u) 53

8.11.2.16 NumTextureDimensions(const TextureType type) . . . 53

9 Namespace Documentation **55**

9.1 LLGL Namespace Reference 55

9.1.1 Typedef Documentation 72

9.1.1.1 ByteBuffer 72

9.1.1.2 ColorRGBAb 73

9.1.1.3 ColorRGBAd 73

9.1.1.4 ColorRGBAf 73

9.1.1.5 ColorRGBAT 73

9.1.1.6 ColorRGBAub 73

9.1.1.7 ColorRGBb 73

9.1.1.8 ColorRGBd 73

9.1.1.9 ColorRGBf 73

9.1.1.10 ColorRGBT 73

9.1.1.11 ColorRGBub 73

9.1.1.12 RenderSystemPtr 73

9.1.1.13 StringView 73

9.1.1.14 UniformLocation 73

9.1.1.15 WStringView 73

9.1.2 Enumeration Type Documentation 74

9.1.2.1 AttachmentLoadOp 74

9.1.2.2 AttachmentStoreOp 74

9.1.2.3 AttachmentType 74

9.1.2.4 BlendArithmetic 75

9.1.2.5 BlendOp 75

9.1.2.6 ClippingRange 76

9.1.2.7 CompareOp 76

9.1.2.8 CPUAccess 77

9.1.2.9 CullMode 78

9.1.2.10 DataType 78

9.1.2.11 ErrorType 79

9.1.2.12 Format . . . 79
9.1.2.13 ImageFormat . . . 84
9.1.2.14 Key . . . 85
9.1.2.15 LogicOp . . . 88
9.1.2.16 OpenGLContextProfile . . . 89
9.1.2.17 PipelineBindPoint . . . 89
9.1.2.18 PolygonMode . . . 90
9.1.2.19 PrimitiveTopology . . . 90
9.1.2.20 QueryType . . . 92
9.1.2.21 RenderConditionMode . . . 92
9.1.2.22 ResourceType . . . 93
9.1.2.23 SamplerAddressMode . . . 93
9.1.2.24 SamplerFilter . . . 94
9.1.2.25 ScreenOrigin . . . 95
9.1.2.26 ShaderSourceType . . . 95
9.1.2.27 ShaderType . . . 96
9.1.2.28 ShadingLanguage . . . 96
9.1.2.29 StencilFace . . . 97
9.1.2.30 StencilOp . . . 98
9.1.2.31 StorageBufferType . . . 98
9.1.2.32 SystemValue . . . 99
9.1.2.33 TessellationPartition . . . 101
9.1.2.34 TextureSwizzle . . . 101
9.1.2.35 TextureType . . . 102
9.1.2.36 UniformType . . . 103
9.1.2.37 WarningType . . . 104
9.1.3 Function Documentation . . . 104
9.1.3.1 CastColorValue(const Src &value) . . . 104
9.1.3.2 CastColorValue< bool, bool >(const bool &value) . . . 104
9.1.3.3 CastColorValue< double, double >(const double &value) . . . 104

9.1.3.4 CastColorValue< float, float >(const float &value) 104
9.1.3.5 CastColorValue< std::uint8_t, std::uint8_t >(const std::uint8_t &value) 104
9.1.3.6 CastTo(const Interface *obj) 105
9.1.3.7 CastTo(Interface *obj) 105
9.1.3.8 CastTo(const Interface &obj) 105
9.1.3.9 CastTo(Interface &obj) 105
9.1.3.10 GetExtentRatio(const Extent2D &extent) 106
9.1.3.11 GetPrimitiveTopologyPatchSize(const PrimitiveTopology primitiveTopology) . . . 106
9.1.3.12 IsInstanceOf(const Interface *obj) 106
9.1.3.13 IsInstanceOf(const Interface &obj) 106
9.1.3.14 IsPrimitiveTopologyPatches(const PrimitiveTopology primitiveTopology) 106
9.1.3.15 IsPrimitiveTopologyStrip(const PrimitiveTopology primitiveTopology) 107
9.1.3.16 IsShaderSourceBinary(const ShaderSourceType type) 107
9.1.3.17 IsShaderSourceCode(const ShaderSourceType type) 107
9.1.3.18 MaxColorValue() 107
9.1.3.19 MaxColorValue< bool >() 107
9.1.3.20 MaxColorValue< std::uint8_t >() 107
9.1.3.21 operator"!=(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)108
9.1.3.22 operator"!=(const UTF8String &lhs, const UTF8String &rhs) 108
9.1.3.23 operator"!=(const UTF8String &lhs, const StringView &rhs) 108
9.1.3.24 operator"!=(const UTF8String &lhs, const WStringView &rhs) 108
9.1.3.25 operator"!=(const UTF8String &lhs, const char *rhs) 108
9.1.3.26 operator"!=(const UTF8String &lhs, const wchar_t *rhs) 108
9.1.3.27 operator"!=(const StringView &lhs, const StringView &rhs) 108
9.1.3.28 operator"!=(const StringView &lhs, const UTF8String &rhs) 108
9.1.3.29 operator"!=(const VertexAttribute &lhs, const VertexAttribute &rhs) 108
9.1.3.30 operator"!=(const StringView &lhs, const char *rhs) 108
9.1.3.31 operator"!=(const WStringView &lhs, const WStringView &rhs) 108
9.1.3.32 operator"!=(const WStringView &lhs, const UTF8String &rhs) 108
9.1.3.33 operator"!=(const WStringView &lhs, const wchar_t *rhs) 108

9.1.3.34 operator"!=(const char *lhs, const UTF8String &rhs) 108
9.1.3.35 operator"!=(const char *lhs, const StringView &rhs) 108
9.1.3.36 operator"!=(const wchar_t *lhs, const UTF8String &rhs) 108
9.1.3.37 operator"!=(const wchar_t *lhs, const WStringView &rhs) 108
9.1.3.38 operator"!=(const Color< T, N > &lhs, const Color< T, N > &rhs) 108
9.1.3.39 operator*(const Color< T, N > &lhs, const Color< T, N > &rhs) 109
9.1.3.40 operator*(const Color< T, N > &lhs, const T &rhs) 109
9.1.3.41 operator*(const T &lhs, const Color< T, N > &rhs) 109
9.1.3.42 operator+(const UTF8String &lhs, const UTF8String &rhs) 109
9.1.3.43 operator+(const UTF8String &lhs, const StringView &rhs) 109
9.1.3.44 operator+(const UTF8String &lhs, const WStringView &rhs) 109
9.1.3.45 operator+(const UTF8String &lhs, const char *rhs) 109
9.1.3.46 operator+(const UTF8String &lhs, const wchar_t *rhs) 109
9.1.3.47 operator+(const StringView &lhs, const UTF8String &rhs) 109
9.1.3.48 operator+(const WStringView &lhs, const UTF8String &rhs) 109
9.1.3.49 operator+(const char *lhs, const UTF8String &rhs) 109
9.1.3.50 operator+(const wchar_t *lhs, const UTF8String &rhs) 109
9.1.3.51 operator+(const Color< T, N > &lhs, const Color< T, N > &rhs) 109
9.1.3.52 operator-(const Color< T, N > &lhs, const Color< T, N > &rhs) 109
9.1.3.53 operator/(const Color< T, N > &lhs, const Color< T, N > &rhs) 109
9.1.3.54 operator/(const Color< T, N > &lhs, const T &rhs) 109
9.1.3.55 operator/(const T &lhs, const Color< T, N > &rhs) 109
9.1.3.56 operator==(const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)109
9.1.3.57 operator==(const UTF8String &lhs, const UTF8String &rhs) 110
9.1.3.58 operator==(const UTF8String &lhs, const StringView &rhs) 110
9.1.3.59 operator==(const UTF8String &lhs, const WStringView &rhs) 110
9.1.3.60 operator==(const UTF8String &lhs, const char *rhs) 110
9.1.3.61 operator==(const UTF8String &lhs, const wchar_t *rhs) 110
9.1.3.62 operator==(const StringView &lhs, const StringView &rhs) 110
9.1.3.63 operator==(const StringView &lhs, const UTF8String &rhs) 110

9.1.3.64 operator==(const StringView &lhs, const char *rhs) . . . 110
9.1.3.65 operator==(const WStringView &lhs, const WStringView &rhs) . . . 110
9.1.3.66 operator==(const WStringView &lhs, const UTF8String &rhs) . . . 110
9.1.3.67 operator==(const WStringView &lhs, const wchar_t *rhs) . . . 110
9.1.3.68 operator==(const char *lhs, const UTF8String &rhs) . . . 110
9.1.3.69 operator==(const char *lhs, const StringView &rhs) . . . 110
9.1.3.70 operator==(const wchar_t *lhs, const UTF8String &rhs) . . . 110
9.1.3.71 operator==(const wchar_t *lhs, const WStringView &rhs) . . . 110
9.1.3.72 operator==(const VertexAttribute &lhs, const VertexAttribute &rhs) . . . 110
9.1.3.73 operator==(const Color< T, N > &lhs, const Color< T, N > &rhs) . . . 110
9.1.3.74 ValidateRenderingCaps(const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={}) . . . 110
9.2 LLGL::Constants Namespace Reference . . . 111
9.2.1 Detailed Description . . . 112
9.2.2 Variable Documentation . . . 112
9.2.2.1 ignoreOffset . . . 112
9.2.2.2 invalidLocation . . . 112
9.2.2.3 invalidSlot . . . 112
9.2.2.4 invalidTimerID . . . 113
9.2.2.5 maxThreadCount . . . 113
9.2.2.6 wholeSize . . . 113
9.3 LLGL::Log Namespace Reference . . . 113
9.3.1 Typedef Documentation . . . 114
9.3.1.1 ReportCallback . . . 114
9.3.2 Enumeration Type Documentation . . . 114
9.3.2.1 ReportType . . . 114
9.3.3 Function Documentation . . . 115
9.3.3.1 PostReport(ReportType type, const StringView &message, const StringView &contextInfo={}) . . . 115
9.3.3.2 SetReportCallback(const ReportCallback &callback, void *userData=nullptr) . . . 115
9.3.3.3 SetReportCallbackStd(std::ostream *stream) . . . 115

9.3.3.4 SetReportLimit(std::size_t maxCount) . . . 116
9.4 LLGL::Timer Namespace Reference . . . 116
9.4.1 Function Documentation . . . 116
9.4.1.1 Frequency() . . . 116
9.4.1.2 Tick() . . . 116
9.5 LLGL::Version Namespace Reference . . . 117
9.5.1 Detailed Description . . . 117
9.5.2 Function Documentation . . . 117
9.5.2.1 GetID() . . . 117
9.5.2.2 GetMajor() . . . 117
9.5.2.3 GetMinor() . . . 118
9.5.2.4 GetRevision() . . . 118
9.5.2.5 GetStatus() . . . 118
9.5.2.6 GetString() . . . 118

10 Class Documentation **119**

10.1 LLGL::ApplicationDescriptor Struct Reference . . . 119
10.1.1 Detailed Description . . . 119
10.1.2 Member Data Documentation . . . 119
10.1.2.1 applicationName . . . 119
10.1.2.2 applicationVersion . . . 120
10.1.2.3 engineName . . . 120
10.1.2.4 engineVersion . . . 120
10.2 LLGL::ArrayView< T > Class Template Reference . . . 120
10.2.1 Detailed Description . . . 121
10.2.2 Member Typedef Documentation . . . 121
10.2.2.1 const_iterator . . . 121
10.2.2.2 const_pointer . . . 121
10.2.2.3 const_reference . . . 122
10.2.2.4 const_reverse_iterator . . . 122
10.2.2.5 difference_type . . . 122

10.2.2.6 iterator 122
10.2.2.7 pointer 122
10.2.2.8 reference 122
10.2.2.9 reverse_iterator 122
10.2.2.10 size_type 122
10.2.2.11 value_type 122
10.2.3 Constructor & Destructor Documentation 122
10.2.3.1 ArrayView()=default 122
10.2.3.2 ArrayView(const ArrayView &)=default 122
10.2.3.3 ArrayView(ArrayView &&)=default 122
10.2.3.4 ArrayView(const std::initializer_list< T > &list) 122
10.2.3.5 ArrayView(const_pointer data, size_type size) 122
10.2.3.6 ArrayView(const T(&data)[N]) 122
10.2.3.7 ArrayView(const OtherContainer< T, OtherAllocator > &other) 123
10.2.4 Member Function Documentation 123
10.2.4.1 at(size_type pos) const 123
10.2.4.2 back() const 123
10.2.4.3 begin() const 123
10.2.4.4 cbegin() const 123
10.2.4.5 cend() const 123
10.2.4.6 crbegin() const 123
10.2.4.7 crend() const 123
10.2.4.8 data() const 123
10.2.4.9 empty() const 123
10.2.4.10 end() const 123
10.2.4.11 front() const 123
10.2.4.12 operator=(const ArrayView &)=default 123
10.2.4.13 operator=(ArrayView &&)=default 123
10.2.4.14 operator[](size_type pos) const 123
10.2.4.15 rbegin() const 123

10.2.4.16 rend() const . . . 123
10.2.4.17 size() const . . . 123
10.3 LLGL::AttachmentClear Struct Reference . . . 124
10.3.1 Detailed Description . . . 124
10.3.2 Constructor & Destructor Documentation . . . 124
10.3.2.1 AttachmentClear()=default . . . 124
10.3.2.2 AttachmentClear(const AttachmentClear &)=default . . . 124
10.3.2.3 AttachmentClear(const ColorRGBAf &color, std::uint32_t colorAttachment) . . . 124
10.3.2.4 AttachmentClear(float depth) . . . 125
10.3.2.5 AttachmentClear(std::uint32_t stencil) . . . 125
10.3.2.6 AttachmentClear(float depth, std::uint32_t stencil) . . . 125
10.3.3 Member Function Documentation . . . 125
10.3.3.1 operator=(const AttachmentClear &)=default . . . 125
10.3.4 Member Data Documentation . . . 125
10.3.4.1 clearValue . . . 125
10.3.4.2 colorAttachment . . . 125
10.3.4.3 flags . . . 125
10.4 LLGL::AttachmentDescriptor Struct Reference . . . 126
10.4.1 Detailed Description . . . 126
10.4.2 Constructor & Destructor Documentation . . . 126
10.4.2.1 AttachmentDescriptor()=default . . . 126
10.4.2.2 AttachmentDescriptor(const AttachmentDescriptor &)=default . . . 126
10.4.2.3 AttachmentDescriptor(AttachmentType type) . . . 126
10.4.2.4 AttachmentDescriptor(AttachmentType type, Texture *texture, std::uint32_t mip↩ Level=0, std::uint32_t arrayLayer=0) . . . 127
10.4.3 Member Data Documentation . . . 127
10.4.3.1 arrayLayer . . . 127
10.4.3.2 mipLevel . . . 127
10.4.3.3 texture . . . 127
10.4.3.4 type . . . 128
10.5 LLGL::AttachmentFormatDescriptor Struct Reference . . . 128

10.5.1 Detailed Description . . . 128

10.5.2 Constructor & Destructor Documentation . . . 129

10.5.2.1 AttachmentFormatDescriptor()=default . . . 129

10.5.2.2 AttachmentFormatDescriptor(const AttachmentFormatDescriptor &)=default . . . 129

10.5.2.3 AttachmentFormatDescriptor(const Format format, const AttachmentLoadOp loadOp=AttachmentLoadOp::Load, const AttachmentStoreOp storeOp=AttachmentStoreOp::Store) . . . 129

10.5.3 Member Data Documentation . . . 129

10.5.3.1 format . . . 129

10.5.3.2 loadOp . . . 129

10.5.3.3 storeOp . . . 130

10.6 LLGL::BarrierFlags Struct Reference . . . 130

10.6.1 Detailed Description . . . 130

10.6.2 Member Enumeration Documentation . . . 130

10.6.2.1 anonymous enum . . . 130

10.7 LLGL::BasicStringView< TChar, Traits > Class Template Reference . . . 131

10.7.1 Detailed Description . . . 132

10.7.2 Member Typedef Documentation . . . 132

10.7.2.1 const_iterator . . . 133

10.7.2.2 const_pointer . . . 133

10.7.2.3 const_reference . . . 133

10.7.2.4 const_reverse_iterator . . . 133

10.7.2.5 difference_type . . . 133

10.7.2.6 iterator . . . 133

10.7.2.7 pointer . . . 133

10.7.2.8 reference . . . 133

10.7.2.9 reverse_iterator . . . 133

10.7.2.10 size_type . . . 133

10.7.2.11 value_type . . . 133

10.7.3 Constructor & Destructor Documentation . . . 133

10.7.3.1 BasicStringView()=default . . . 133

10.7.3.2 BasicStringView(const BasicStringView &)=default . . . 133
10.7.3.3 BasicStringView(BasicStringView &&)=default . . . 133
10.7.3.4 BasicStringView(const_pointer str) . . . 133
10.7.3.5 BasicStringView(const_pointer str, size_type len) . . . 134
10.7.3.6 BasicStringView(const TString< TChar, Traits, Allocator > &str) . . . 134
10.7.4 Member Function Documentation . . . 134
10.7.4.1 at(size_type pos) const . . . 134
10.7.4.2 back() const . . . 134
10.7.4.3 begin() const . . . 134
10.7.4.4 cbegin() const . . . 134
10.7.4.5 cend() const . . . 134
10.7.4.6 compare(const BasicStringView &str) const . . . 134
10.7.4.7 compare(size_type pos1, size_type count1, const BasicStringView &str) const . 134
10.7.4.8 compare(size_type pos1, size_type count1, const BasicStringView &str, size_type pos2, size_type count2=npos) const . . . 134
10.7.4.9 crbegin() const . . . 134
10.7.4.10 crend() const . . . 134
10.7.4.11 data() const . . . 134
10.7.4.12 empty() const . . . 135
10.7.4.13 end() const . . . 135
10.7.4.14 front() const . . . 135
10.7.4.15 length() const . . . 135
10.7.4.16 operator=(const BasicStringView &)=default . . . 135
10.7.4.17 operator=(BasicStringView &&)=default . . . 135
10.7.4.18 operator[](size_type pos) const . . . 135
10.7.4.19 rbegin() const . . . 135
10.7.4.20 rend() const . . . 135
10.7.4.21 size() const . . . 135
10.7.4.22 substr(size_type pos=0, size_type count=npos) const . . . 135
10.7.5 Member Data Documentation . . . 135
10.7.5.1 npos . . . 135

10.8 LLGL::BindFlags Struct Reference . . . 135
10.8.1 Detailed Description . . . 136
10.8.2 Member Enumeration Documentation . . . 136
10.8.2.1 anonymous enum . . . 136
10.9 LLGL::BindingDescriptor Struct Reference . . . 138
10.9.1 Detailed Description . . . 139
10.9.2 Constructor & Destructor Documentation . . . 139
10.9.2.1 BindingDescriptor()=default . . . 139
10.9.2.2 BindingDescriptor(const BindingDescriptor &)=default . . . 139
10.9.2.3 BindingDescriptor(ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1) . . . 139
10.9.2.4 BindingDescriptor(const std::string &name, ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1) . . . 139
10.9.3 Member Data Documentation . . . 139
10.9.3.1 arraySize . . . 139
10.9.3.2 bindFlags . . . 139
10.9.3.3 name . . . 140
10.9.3.4 slot . . . 140
10.9.3.5 stageFlags . . . 140
10.9.3.6 type . . . 140
10.10 LLGL::BlendDescriptor Struct Reference . . . 140
10.10.1 Detailed Description . . . 141
10.10.2 Member Data Documentation . . . 141
10.10.2.1 alphaToCoverageEnabled . . . 141
10.10.2.2 blendFactor . . . 142
10.10.2.3 blendFactorDynamic . . . 142
10.10.2.4 independentBlendEnabled . . . 142
10.10.2.5 logicOp . . . 143
10.10.2.6 sampleMask . . . 143
10.10.2.7 targets . . . 143
10.11 LLGL::BlendTargetDescriptor Struct Reference . . . 144

10.11.1 Detailed Description . . . 144
10.11.2 Member Data Documentation . . . 144
10.11.2.1 alphaArithmetic . . . 144
10.11.2.2 blendEnabled . . . 144
10.11.2.3 colorArithmetic . . . 144
10.11.2.4 colorMask . . . 145
10.11.2.5 dstAlpha . . . 145
10.11.2.6 dstColor . . . 145
10.11.2.7 srcAlpha . . . 145
10.11.2.8 srcColor . . . 145
10.12 LLGL::Blob Class Reference . . . 145
10.12.1 Detailed Description . . . 146
10.12.2 Member Function Documentation . . . 146
10.12.2.1 CreateCopy(const void *data, std::size_t size) . . . 146
10.12.2.2 CreateFromFile(const char *filename) . . . 147
10.12.2.3 CreateFromFile(const std::string &filename) . . . 147
10.12.2.4 CreateStrongRef(std::vector< std::int8_t > &&container) . . . 147
10.12.2.5 CreateStrongRef(std::string &&container) . . . 147
10.12.2.6 CreateWeakRef(const void *data, std::size_t size) . . . 148
10.12.2.7 GetData() const =0 . . . 148
10.12.2.8 GetSize() const =0 . . . 148
10.13 LLGL::Buffer Class Reference . . . 148
10.13.1 Detailed Description . . . 149
10.13.2 Constructor & Destructor Documentation . . . 149
10.13.2.1 Buffer(long bindFlags) . . . 149
10.13.3 Member Function Documentation . . . 149
10.13.3.1 GetBindFlags() const . . . 149
10.13.3.2 GetDesc() const =0 . . . 149
10.13.3.3 GetResourceType() const override final . . . 150
10.14 LLGL::BufferArray Class Reference . . . 150

10.14.1 Detailed Description . . . 150
10.14.2 Constructor & Destructor Documentation . . . 151
10.14.2.1 BufferArray(long bindFlags) . . . 151
10.14.3 Member Function Documentation . . . 151
10.14.3.1 GetBindFlags() const . . . 151
10.15 LLGL::BufferDescriptor Struct Reference . . . 151
10.15.1 Detailed Description . . . 151
10.15.2 Member Data Documentation . . . 152
10.15.2.1 bindFlags . . . 152
10.15.2.2 cpuAccessFlags . . . 152
10.15.2.3 format . . . 152
10.15.2.4 miscFlags . . . 153
10.15.2.5 size . . . 153
10.15.2.6 stride . . . 153
10.15.2.7 vertexAttribs . . . 154
10.16 LLGL::BufferViewDescriptor Struct Reference . . . 154
10.16.1 Detailed Description . . . 154
10.16.2 Constructor & Destructor Documentation . . . 155
10.16.2.1 BufferViewDescriptor()=default . . . 155
10.16.2.2 BufferViewDescriptor(Format format, std::uint64_t offset=0, std::uint64_t size=↩ Constants::wholeSize) . . . 155
10.16.3 Member Data Documentation . . . 155
10.16.3.1 format . . . 155
10.16.3.2 offset . . . 155
10.16.3.3 size . . . 156
10.17 LLGL::Canvas Class Reference . . . 156
10.17.1 Detailed Description . . . 157
10.17.2 Constructor & Destructor Documentation . . . 158
10.17.2.1 ∼Canvas() . . . 158
10.17.2.2 Canvas() . . . 158
10.17.3 Member Function Documentation . . . 158

10.17.3.1 AdaptForVideoMode(Extent2D *resolution, bool *fullscreen) override 158
10.17.3.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 158
10.17.3.3 Create(const CanvasDescriptor &desc) 158
10.17.3.4 FindResidentDisplay() const override final 158
10.17.3.5 GetTitle() const =0 159
10.17.3.6 HasQuit() const 159
10.17.3.7 OnProcessEvents()=0 159
10.17.3.8 PostQuit() 159
10.17.3.9 ProcessEvents() override final 159
10.17.3.10 RemoveEventListener(const EventListener *eventListener) 160
10.17.3.11 SetTitle(const UTF8String &title)=0 160
10.18 LLGL::CanvasDescriptor Struct Reference 160
10.18.1 Detailed Description 160
10.18.2 Member Data Documentation 160
10.18.2.1 borderless 160
10.18.2.2 title 160
10.19 LLGL::ClearFlags Struct Reference 161
10.19.1 Detailed Description 161
10.19.2 Member Enumeration Documentation 161
10.19.2.1 anonymous enum 161
10.20 LLGL::ClearValue Struct Reference 161
10.20.1 Detailed Description 162
10.20.2 Constructor & Destructor Documentation 162
10.20.2.1 ClearValue()=default 162
10.20.2.2 ClearValue(const ClearValue &)=default 162
10.20.2.3 ClearValue(const ColorRGBAf &color, float depth=1.0f, std::uint32_t stencil=0) . 162
10.20.2.4 ClearValue(float depth) 162
10.20.2.5 ClearValue(std::uint32_t stencil) 163
10.20.2.6 ClearValue(float depth, std::uint32_t stencil) 163
10.20.3 Member Function Documentation 163

10.20.3.1 operator=(const ClearValue &)=default . . . 163
10.20.4 Member Data Documentation . . . 163
10.20.4.1 color . . . 163
10.20.4.2 depth . . . 163
10.20.4.3 stencil . . . 163
10.21 LLGL::Color< T, N > Class Template Reference . . . 163
10.21.1 Detailed Description . . . 164
10.21.2 Constructor & Destructor Documentation . . . 165
10.21.2.1 Color() . . . 165
10.21.2.2 Color(const Color< T, N > &rhs) . . . 165
10.21.2.3 Color(const T &scalar) . . . 165
10.21.2.4 Color(UninitializeTag) . . . 165
10.21.3 Member Function Documentation . . . 165
10.21.3.1 Cast() const . . . 165
10.21.3.2 operator*=(const Color< T, N > &rhs) . . . 166
10.21.3.3 operator*=(const T rhs) . . . 166
10.21.3.4 operator+=(const Color< T, N > &rhs) . . . 166
10.21.3.5 operator-() const . . . 166
10.21.3.6 operator-=(const Color< T, N > &rhs) . . . 166
10.21.3.7 operator/=(const Color< T, N > &rhs) . . . 166
10.21.3.8 operator/=(const T rhs) . . . 166
10.21.3.9 operator[](std::size_t component) . . . 166
10.21.3.10 operator[](std::size_t component) const . . . 167
10.21.3.11 Ptr() . . . 167
10.21.3.12 Ptr() const . . . 167
10.21.4 Member Data Documentation . . . 167
10.21.4.1 components . . . 167
10.22 LLGL::Color< T, 3u > Class Template Reference . . . 167
10.22.1 Detailed Description . . . 169
10.22.2 Constructor & Destructor Documentation . . . 169

10.22.2.1 Color() . . . 169
10.22.2.2 Color(const Color< T, 3 > &rhs) . . . 169
10.22.2.3 Color(const T &scalar) . . . 169
10.22.2.4 Color(const T &r, const T &g, const T &b) . . . 169
10.22.2.5 Color(UninitializeTag) . . . 169
10.22.3 Member Function Documentation . . . 169
10.22.3.1 Cast() const . . . 169
10.22.3.2 operator*=(const Color< T, 3 > &rhs) . . . 170
10.22.3.3 operator*=(const T rhs) . . . 170
10.22.3.4 operator+=(const Color< T, 3 > &rhs) . . . 170
10.22.3.5 operator-() const . . . 170
10.22.3.6 operator-=(const Color< T, 3 > &rhs) . . . 170
10.22.3.7 operator/=(const Color< T, 3 > &rhs) . . . 170
10.22.3.8 operator/=(const T rhs) . . . 170
10.22.3.9 operator[](std::size_t component) . . . 170
10.22.3.10 operator[](std::size_t component) const . . . 171
10.22.3.11 Ptr() . . . 171
10.22.3.12 Ptr() const . . . 171
10.22.3.13 ToRGBA() const . . . 171
10.22.4 Member Data Documentation . . . 171
10.22.4.1 b . . . 171
10.22.4.2 components . . . 171
10.22.4.3 g . . . 172
10.22.4.4 r . . . 172
10.23 LLGL::Color< T, 4u > Class Template Reference . . . 172
10.23.1 Detailed Description . . . 173
10.23.2 Constructor & Destructor Documentation . . . 173
10.23.2.1 Color() . . . 173
10.23.2.2 Color(const Color< T, 4 > &rhs) . . . 174
10.23.2.3 Color(const T &scalar) . . . 174

10.23.2.4 Color(const Color< T, 3u > &rhs) . . . 174
10.23.2.5 Color(const T &r, const T &g, const T &b) . . . 174
10.23.2.6 Color(const T &r, const T &g, const T &b, const T &a) . . . 174
10.23.2.7 Color(UninitializeTag) . . . 174
10.23.3 Member Function Documentation . . . 174
10.23.3.1 Cast() const . . . 174
10.23.3.2 operator*=(const Color< T, 4 > &rhs) . . . 175
10.23.3.3 operator*=(const T rhs) . . . 175
10.23.3.4 operator+=(const Color< T, 4 > &rhs) . . . 175
10.23.3.5 operator-() const . . . 175
10.23.3.6 operator-=(const Color< T, 4 > &rhs) . . . 175
10.23.3.7 operator/=(const Color< T, 4 > &rhs) . . . 175
10.23.3.8 operator/=(const T rhs) . . . 175
10.23.3.9 operator[](std::size_t component) . . . 175
10.23.3.10 operator[](std::size_t component) const . . . 176
10.23.3.11 Ptr() . . . 176
10.23.3.12 Ptr() const . . . 176
10.23.3.13 ToRGB() const . . . 176
10.23.4 Member Data Documentation . . . 176
10.23.4.1 a . . . 176
10.23.4.2 b . . . 176
10.23.4.3 components . . . 176
10.23.4.4 g . . . 177
10.23.4.5 r . . . 177
10.24 LLGL::CommandBuffer Class Reference . . . 177
10.24.1 Detailed Description . . . 180
10.24.2 Constructor & Destructor Documentation . . . 180
10.24.2.1 CommandBuffer()=default . . . 180
10.24.3 Member Function Documentation . . . 180
10.24.3.1 Begin()=0 . . . 180

10.24.3.2 BeginQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 180

10.24.3.3 BeginRenderCondition(QueryHeap &queryHeap, std::uint32_t query=0, const RenderConditionMode mode=RenderConditionMode::Wait)=0 181

10.24.3.4 BeginRenderPass(RenderTarget &renderTarget, const RenderPass *renderPass=nullptr, std::uint32_t numClearValues=0, const ClearValue *clearValues=nullptr)=0 181

10.24.3.5 BeginStreamOutput(std::uint32_t numBuffers, Buffer *const *buffers)=0 183

10.24.3.6 Clear(long flags, const ClearValue &clearValue={})=0 183

10.24.3.7 ClearAttachments(std::uint32_t numAttachments, const AttachmentClear *attachments)=0 184

10.24.3.8 CopyBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, Buffer &srcBuffer, std::uint64_t srcOffset, std::uint64_t size)=0 184

10.24.3.9 CopyBufferFromTexture(Buffer &dstBuffer, std::uint64_t dstOffset, Texture &srcTexture, const TextureRegion &srcRegion, std::uint32_t rowStride=0, std::uint32_t layerStride=0)=0 184

10.24.3.10 CopyTexture(Texture &dstTexture, const TextureLocation &dstLocation, Texture &srcTexture, const TextureLocation &srcLocation, const Extent3D &extent)=0 . . 185

10.24.3.11 CopyTextureFromBuffer(Texture &dstTexture, const TextureRegion &dstRegion, Buffer &srcBuffer, std::uint64_t srcOffset, std::uint32_t rowStride=0, std::uint32_t layerStride=0)=0 186

10.24.3.12 Dispatch(std::uint32_t numWorkGroupsX, std::uint32_t numWorkGroupsY, std::uint32_t numWorkGroupsZ)=0 187

10.24.3.13 DispatchIndirect(Buffer &buffer, std::uint64_t offset)=0 187

10.24.3.14 Draw(std::uint32_t numVertices, std::uint32_t firstVertex)=0 187

10.24.3.15 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex)=0 188

10.24.3.16 DrawIndexed(std::uint32_t numIndices, std::uint32_t firstIndex, std::int32_t vertexOffset)=0 188

10.24.3.17 DrawIndexedIndirect(Buffer &buffer, std::uint64_t offset)=0 188

10.24.3.18 DrawIndexedIndirect(Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0 189

10.24.3.19 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex)=0 189

10.24.3.20 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset)=0 189

10.24.3.21 DrawIndexedInstanced(std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset, std::uint32_t firstInstance)=0 . . 189

10.24.3.22 DrawIndirect(Buffer &buffer, std::uint64_t offset)=0 190

10.24.3.23 DrawIndirect(Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0 . . . 190

10.24.3.24 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances)=0 . . . 191

10.24.3.25 DrawInstanced(std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances, std::uint32_t firstInstance)=0 . . . 191

10.24.3.26 End()=0 . . . 192

10.24.3.27 EndQuery(QueryHeap &queryHeap, std::uint32_t query=0)=0 . . . 192

10.24.3.28 EndRenderCondition()=0 . . . 192

10.24.3.29 EndRenderPass()=0 . . . 193

10.24.3.30 EndStreamOutput()=0 . . . 193

10.24.3.31 Execute(CommandBuffer &deferredCommandBuffer)=0 . . . 193

10.24.3.32 FillBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, std::uint32_t value, std::uint64_t fillSize=Constants::wholeSize)=0 . . . 193

10.24.3.33 GenerateMips(Texture &texture)=0 . . . 194

10.24.3.34 GenerateMips(Texture &texture, const TextureSubresource &subresource)=0 . . . 194

10.24.3.35 PopDebugGroup()=0 . . . 195

10.24.3.36 PushDebugGroup(const char *name)=0 . . . 195

10.24.3.37 ResetResourceSlots(const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_t numSlots, long bindFlags, long stageFlags=StageFlags::AllStages)=0 196

10.24.3.38 SetBlendFactor(const ColorRGBAf &color)=0 . . . 196

10.24.3.39 SetGraphicsAPIDependentState(const void *stateDesc, std::size_t stateDescSize)=0 . . . 197

10.24.3.40 SetIndexBuffer(Buffer &buffer)=0 . . . 197

10.24.3.41 SetIndexBuffer(Buffer &buffer, const Format format, std::uint64_t offset=0)=0 . . . 198

10.24.3.42 SetPipelineState(PipelineState &pipelineState)=0 . . . 198

10.24.3.43 SetResource(Resource &resource, std::uint32_t slot, long bindFlags, long stageFlags=StageFlags::AllStages)=0 . . . 199

10.24.3.44 SetResourceHeap(ResourceHeap &resourceHeap, std::uint32_t descriptorSet=0, const PipelineBindPoint bindPoint=PipelineBindPoint::Undefined)=0 . . . 200

10.24.3.45 SetScissor(const Scissor &scissor)=0 . . . 200

10.24.3.46 SetScissors(std::uint32_t numScissors, const Scissor *scissors)=0 . . . 200

10.24.3.47 SetStencilReference(std::uint32_t reference, const StencilFace stencilFace=StencilFace::FrontAndBack)=0 . . . 201

10.24.3.48 SetUniform(UniformLocation location, const void *data, std::uint32_t dataSize)=0 201

10.24.3.49 SetUniforms(UniformLocation location, std::uint32_t count, const void *data, std::uint32_t dataSize)=0 . . . 202

10.24.3.50 SetVertexBuffer(Buffer &buffer)=0 . . . 203

10.24.3.51 SetVertexBufferArray(BufferArray &bufferArray)=0 . . . 203

10.24.3.52 SetViewport(const Viewport &viewport)=0 . . . 203

10.24.3.53 SetViewports(std::uint32_t numViewports, const Viewport *viewports)=0 . . . 203

10.24.3.54 UpdateBuffer(Buffer &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint16_t dataSize)=0 . . . 204

10.25 LLGL::CommandBufferDescriptor Struct Reference . . . 204

10.25.1 Detailed Description . . . 205

10.25.2 Constructor & Destructor Documentation . . . 205

10.25.2.1 CommandBufferDescriptor()=default . . . 205

10.25.2.2 CommandBufferDescriptor(const CommandBufferDescriptor &)=default . . . 205

10.25.2.3 CommandBufferDescriptor(long flags) . . . 205

10.25.2.4 CommandBufferDescriptor(long flags, std::uint32_t numNativeBuffers) . . . 205

10.25.3 Member Function Documentation . . . 206

10.25.3.1 operator=(const CommandBufferDescriptor &)=default . . . 206

10.25.4 Member Data Documentation . . . 206

10.25.4.1 flags . . . 206

10.25.4.2 numNativeBuffers . . . 206

10.26 LLGL::CommandBufferFlags Struct Reference . . . 206

10.26.1 Detailed Description . . . 207

10.26.2 Member Enumeration Documentation . . . 207

10.26.2.1 anonymous enum . . . 207

10.27 LLGL::CommandQueue Class Reference . . . 208

10.27.1 Detailed Description . . . 208

10.27.2 Constructor & Destructor Documentation . . . 208

10.27.2.1 CommandQueue()=default . . . 208

10.27.3 Member Function Documentation . . . 208

10.27.3.1 QueryResult(QueryHeap &queryHeap, std::uint32_t firstQuery, std::uint32_t numQueries, void *data, std::size_t dataSize)=0 . . . 208

10.27.3.2 Submit(CommandBuffer &commandBuffer)=0 . . . 209
10.27.3.3 Submit(Fence &fence)=0 . . . 210
10.27.3.4 WaitFence(Fence &fence, std::uint64_t timeout)=0 . . . 210
10.27.3.5 WaitIdle()=0 . . . 210
10.28 LLGL::ComputePipelineDescriptor Struct Reference . . . 211
10.28.1 Detailed Description . . . 211
10.28.2 Member Data Documentation . . . 211
10.28.2.1 computeShader . . . 211
10.28.2.2 pipelineLayout . . . 211
10.29 LLGL::ComputeShaderAttributes Struct Reference . . . 212
10.29.1 Detailed Description . . . 212
10.29.2 Member Data Documentation . . . 212
10.29.2.1 workGroupSize . . . 212
10.30 LLGL::CPUAccessFlags Struct Reference . . . 212
10.30.1 Detailed Description . . . 213
10.30.2 Member Enumeration Documentation . . . 213
10.30.2.1 anonymous enum . . . 213
10.31 LLGL::DepthBiasDescriptor Struct Reference . . . 213
10.31.1 Detailed Description . . . 214
10.31.2 Member Data Documentation . . . 214
10.31.2.1 clamp . . . 214
10.31.2.2 constantFactor . . . 214
10.31.2.3 slopeFactor . . . 214
10.32 LLGL::DepthDescriptor Struct Reference . . . 214
10.32.1 Detailed Description . . . 215
10.32.2 Member Data Documentation . . . 215
10.32.2.1 compareOp . . . 215
10.32.2.2 testEnabled . . . 215
10.32.2.3 writeEnabled . . . 215
10.33 LLGL::DispatchIndirectArguments Struct Reference . . . 215

10.33.1 Detailed Description . . . 216
10.33.2 Member Data Documentation . . . 216
10.33.2.1 numThreadGroups . . . 216
10.34 LLGL::Display Class Reference . . . 216
10.34.1 Detailed Description . . . 218
10.34.2 Member Function Documentation . . . 218
10.34.2.1 Count() . . . 218
10.34.2.2 FinalizeDisplayModes(std::vector< DisplayModeDescriptor > &displayModeDescs) 218
10.34.2.3 Get(std::size_t index) . . . 218
10.34.2.4 GetCursorPosition() . . . 219
10.34.2.5 GetDeviceName() const =0 . . . 219
10.34.2.6 GetDisplayMode() const =0 . . . 219
10.34.2.7 GetList() . . . 219
10.34.2.8 GetOffset() const =0 . . . 220
10.34.2.9 GetPrimary() . . . 220
10.34.2.10 GetSupportedDisplayModes() const =0 . . . 220
10.34.2.11 IsCursorShown() . . . 220
10.34.2.12 IsPrimary() const =0 . . . 220
10.34.2.13 ResetDisplayMode()=0 . . . 221
10.34.2.14 SetCursorPosition(const Offset2D &position) . . . 221
10.34.2.15 SetDisplayMode(const DisplayModeDescriptor &displayModeDesc)=0 . . . 221
10.34.2.16 ShowCursor(bool show) . . . 221
10.35 LLGL::DisplayModeDescriptor Struct Reference . . . 222
10.35.1 Detailed Description . . . 222
10.35.2 Member Data Documentation . . . 222
10.35.2.1 refreshRate . . . 222
10.35.2.2 resolution . . . 223
10.36 LLGL::DrawIndexedIndirectArguments Struct Reference . . . 223
10.36.1 Detailed Description . . . 223
10.36.2 Member Data Documentation . . . 224

10.36.2.1 firstIndex . . . 224
10.36.2.2 firstInstance . . . 224
10.36.2.3 numIndices . . . 224
10.36.2.4 numInstances . . . 224
10.36.2.5 vertexOffset . . . 224
10.37 LLGL::DrawIndirectArguments Struct Reference . . . 224
10.37.1 Detailed Description . . . 225
10.37.2 Member Data Documentation . . . 225
10.37.2.1 firstInstance . . . 225
10.37.2.2 firstVertex . . . 226
10.37.2.3 numInstances . . . 226
10.37.2.4 numVertices . . . 226
10.38 LLGL::DrawPatchIndirectArguments Struct Reference . . . 226
10.38.1 Detailed Description . . . 227
10.38.2 Member Data Documentation . . . 227
10.38.2.1 firstInstance . . . 227
10.38.2.2 firstPatch . . . 227
10.38.2.3 numInstances . . . 227
10.38.2.4 numPatches . . . 227
10.39 LLGL::DstImageDescriptor Struct Reference . . . 227
10.39.1 Detailed Description . . . 228
10.39.2 Constructor & Destructor Documentation . . . 228
10.39.2.1 DstImageDescriptor()=default . . . 228
10.39.2.2 DstImageDescriptor(const DstImageDescriptor &)=default . . . 228
10.39.2.3 DstImageDescriptor(ImageFormat format, DataType dataType, void *data, std↩ ::size_t dataSize) . . . 228
10.39.3 Member Data Documentation . . . 228
10.39.3.1 data . . . 228
10.39.3.2 dataSize . . . 229
10.39.3.3 dataType . . . 229
10.39.3.4 format . . . 229

10.40 LLGL::Window::EventListener Class Reference . . . 229
10.40.1 Detailed Description . . . 230
10.40.2 Member Function Documentation . . . 230
10.40.2.1 OnChar(Window &sender, wchar_t chr) . . . 230
10.40.2.2 OnDoubleClick(Window &sender, Key keyCode) . . . 230
10.40.2.3 OnGetFocus(Window &sender) . . . 231
10.40.2.4 OnGlobalMotion(Window &sender, const Offset2D &motion) . . . 231
10.40.2.5 OnKeyDown(Window &sender, Key keyCode) . . . 231
10.40.2.6 OnKeyUp(Window &sender, Key keyCode) . . . 231
10.40.2.7 OnLocalMotion(Window &sender, const Offset2D &position) . . . 231
10.40.2.8 OnLostFocus(Window &sender) . . . 231
10.40.2.9 OnProcessEvents(Window &sender) . . . 231
10.40.2.10 OnQuit(Window &sender, bool &veto) . . . 231
10.40.2.11 OnResize(Window &sender, const Extent2D &clientAreaSize) . . . 232
10.40.2.12 OnTimer(Window &sender, std::uint32_t timerID) . . . 232
10.40.2.13 OnWheelMotion(Window &sender, int motion) . . . 232
10.40.3 Friends And Related Function Documentation . . . 232
10.40.3.1 Window . . . 232
10.41 LLGL::Canvas::EventListener Class Reference . . . 233
10.41.1 Detailed Description . . . 233
10.41.2 Member Function Documentation . . . 233
10.41.2.1 OnProcessEvents(Canvas &sender) . . . 233
10.41.2.2 OnQuit(Canvas &sender, bool &veto) . . . 234
10.41.3 Friends And Related Function Documentation . . . 234
10.41.3.1 Canvas . . . 234
10.42 LLGL::Extent2D Struct Reference . . . 234
10.42.1 Detailed Description . . . 235
10.42.2 Constructor & Destructor Documentation . . . 235
10.42.2.1 Extent2D()=default . . . 235
10.42.2.2 Extent2D(const Extent2D &)=default . . . 235

10.42.2.3 Extent2D(std::uint32_t width, std::uint32_t height) . . . 235
10.42.3 Member Data Documentation . . . 235
10.42.3.1 height . . . 235
10.42.3.2 width . . . 235
10.43 LLGL::Extent3D Struct Reference . . . 235
10.43.1 Detailed Description . . . 236
10.43.2 Constructor & Destructor Documentation . . . 236
10.43.2.1 Extent3D()=default . . . 236
10.43.2.2 Extent3D(const Extent3D &)=default . . . 236
10.43.2.3 Extent3D(std::uint32_t width, std::uint32_t height, std::uint32_t depth) . . . 236
10.43.3 Member Data Documentation . . . 236
10.43.3.1 depth . . . 236
10.43.3.2 height . . . 236
10.43.3.3 width . . . 236
10.44 LLGL::Fence Class Reference . . . 237
10.44.1 Detailed Description . . . 237
10.45 LLGL::FormatAttributes Struct Reference . . . 237
10.45.1 Detailed Description . . . 238
10.45.2 Member Data Documentation . . . 238
10.45.2.1 bitSize . . . 238
10.45.2.2 blockHeight . . . 238
10.45.2.3 blockWidth . . . 238
10.45.2.4 components . . . 239
10.45.2.5 dataType . . . 239
10.45.2.6 flags . . . 239
10.45.2.7 format . . . 239
10.46 LLGL::FormatFlags Struct Reference . . . 239
10.46.1 Detailed Description . . . 240
10.46.2 Member Enumeration Documentation . . . 240
10.46.2.1 anonymous enum . . . 240

10.47 LLGL::FragmentAttribute Struct Reference . . . 241
10.47.1 Detailed Description . . . 242
10.47.2 Constructor & Destructor Documentation . . . 242
10.47.2.1 FragmentAttribute()=default . . . 242
10.47.2.2 FragmentAttribute(const FragmentAttribute &)=default . . . 242
10.47.2.3 FragmentAttribute(const StringView &name, std::uint32_t location=0) . . . 242
10.47.2.4 FragmentAttribute(const StringView &name, const Format format, std::uint32_t location=0, const SystemValue systemValue=SystemValue::Undefined) . . . 242
10.47.3 Member Function Documentation . . . 242
10.47.3.1 operator=(const FragmentAttribute &)=default . . . 242
10.47.4 Member Data Documentation . . . 242
10.47.4.1 format . . . 242
10.47.4.2 location . . . 243
10.47.4.3 name . . . 243
10.47.4.4 systemValue . . . 243
10.48 LLGL::FragmentShaderAttributes Struct Reference . . . 244
10.48.1 Detailed Description . . . 244
10.48.2 Member Data Documentation . . . 244
10.48.2.1 outputAttribs . . . 244
10.49 LLGL::FrameProfile Struct Reference . . . 244
10.49.1 Detailed Description . . . 246
10.49.2 Constructor & Destructor Documentation . . . 246
10.49.2.1 FrameProfile() . . . 246
10.49.3 Member Function Documentation . . . 246
10.49.3.1 Accumulate(const FrameProfile &rhs) . . . 246
10.49.3.2 Clear() . . . 246
10.49.4 Member Data Documentation . . . 246
10.49.4.1 "@5 . . . 246
10.49.4.2 attachmentClears . . . 246
10.49.4.3 bufferCopies . . . 247
10.49.4.4 bufferFills . . . 247

10.49.4.5 bufferMappings . . . 247
10.49.4.6 bufferReads . . . 247
10.49.4.7 bufferUpdates . . . 247
10.49.4.8 bufferWrites . . . 247
10.49.4.9 commandBufferEncodings . . . 248
10.49.4.10 commandBufferSubmittions . . . 248
10.49.4.11 computePipelineBindings . . . 248
10.49.4.12 constantBufferBindings . . . 248
10.49.4.13 dispatchCommands . . . 248
10.49.4.14 drawCommands . . . 249
10.49.4.15 fenceSubmissions . . . 249
10.49.4.16 graphicsPipelineBindings . . . 249
10.49.4.17 indexBufferBindings . . . 249
10.49.4.18 mipMapsGenerations . . . 249
10.49.4.19 querySections . . . 250
10.49.4.20 renderConditionSections . . . 250
10.49.4.21 renderPassSections . . . 250
10.49.4.22 resourceHeapBindings . . . 250
10.49.4.23 sampledBufferBindings . . . 250
10.49.4.24 sampledTextureBindings . . . 251
10.49.4.25 samplerBindings . . . 251
10.49.4.26 storageBufferBindings . . . 251
10.49.4.27 storageTextureBindings . . . 251
10.49.4.28 streamOutputSections . . . 251
10.49.4.29 textureCopies . . . 252
10.49.4.30 textureReads . . . 252
10.49.4.31 textureWrites . . . 252
10.49.4.32 timeRecords . . . 252
10.49.4.33 values . . . 252
10.49.4.34 vertexBufferBindings . . . 252

10.50 LLGL::GraphicsPipelineDescriptor Struct Reference . . . 253
10.50.1 Detailed Description . . . 253
10.50.2 Member Data Documentation . . . 254
10.50.2.1 blend . . . 254
10.50.2.2 depth . . . 254
10.50.2.3 fragmentShader . . . 254
10.50.2.4 geometryShader . . . 254
10.50.2.5 pipelineLayout . . . 254
10.50.2.6 primitiveTopology . . . 254
10.50.2.7 rasterizer . . . 255
10.50.2.8 renderPass . . . 255
10.50.2.9 scissors . . . 255
10.50.2.10 stencil . . . 255
10.50.2.11 tessControlShader . . . 255
10.50.2.12 tessellation . . . 256
10.50.2.13 tessEvaluationShader . . . 256
10.50.2.14 vertexShader . . . 256
10.50.2.15 viewports . . . 256
10.51 LLGL::GrowStrategyAddHalf Struct Reference . . . 257
10.51.1 Detailed Description . . . 257
10.51.2 Member Function Documentation . . . 257
10.51.2.1 IncreaseToFit(std::size_t size) . . . 257
10.52 LLGL::GrowStrategyDouble Struct Reference . . . 257
10.52.1 Detailed Description . . . 258
10.52.2 Member Function Documentation . . . 258
10.52.2.1 IncreaseToFit(std::size_t size) . . . 258
10.53 LLGL::GrowStrategyRoundUpPow2 Struct Reference . . . 258
10.53.1 Detailed Description . . . 258
10.53.2 Member Function Documentation . . . 258
10.53.2.1 IncreaseToFit(std::size_t size) . . . 258

10.53.2.2 Round(unsigned int v) . . . 259

10.53.2.3 Round(unsigned long v) . . . 259

10.53.2.4 Round(unsigned long long v) . . . 259

10.54 LLGL::Image Class Reference . . . 259

10.54.1 Detailed Description . . . 261

10.54.2 Constructor & Destructor Documentation . . . 261

10.54.2.1 Image()=default . . . 261

10.54.2.2 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType) . . . 261

10.54.2.3 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, const ColorRGBAd &fillColor) . . . 261

10.54.2.4 Image(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) . . . 262

10.54.2.5 Image(const Image &rhs) . . . 262

10.54.2.6 Image(Image &&rhs) . . . 262

10.54.3 Member Function Documentation . . . 262

10.54.3.1 Blit(Offset3D dstRegionOffset, const Image &srcImage, Offset3D srcRegion↩ Offset, Extent3D srcRegionExtent) . . . 262

10.54.3.2 Convert(const ImageFormat format, const DataType dataType, unsigned thread↩ Count=0) . . . 263

10.54.3.3 Fill(Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor) . . . 263

10.54.3.4 GetBytesPerPixel() const . . . 263

10.54.3.5 GetData() const . . . 263

10.54.3.6 GetData() . . . 263

10.54.3.7 GetDataSize() const . . . 264

10.54.3.8 GetDataType() const . . . 264

10.54.3.9 GetDepthStride() const . . . 264

10.54.3.10 GetDstDesc() . . . 264

10.54.3.11 GetExtent() const . . . 264

10.54.3.12 GetFormat() const . . . 264

10.54.3.13 GetNumPixels() const . . . 264

10.54.3.14 GetRowStride() const . . . 265

10.54.3.15 GetSrcDesc() const . . . 265

10.54.3.16 IsRegionInside(const Offset3D &offset, const Extent3D &extent) const 265

10.54.3.17 MirrorXYPlane() 265

10.54.3.18 MirrorXZPlane() 265

10.54.3.19 MirrorYZPlane() 265

10.54.3.20 operator=(const Image &rhs) 265

10.54.3.21 operator=(Image &&rhs) 265

10.54.3.22 ReadPixels(const Offset3D &offset, const Extent3D &extent, const DstImage↩ Descriptor &imageDesc, unsigned threadCount=0) const 265

10.54.3.23 Release() 266

10.54.3.24 Reset() 266

10.54.3.25 Reset(const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data) 266

10.54.3.26 Resize(const Extent3D &extent) 267

10.54.3.27 Resize(const Extent3D &extent, const ColorRGBAd &fillColor) 267

10.54.3.28 Resize(const Extent3D &extent, const ColorRGBAd &fillColor, const Offset3↩ D &offset) 267

10.54.3.29 Resize(const Extent3D &extent, const SamplerFilter filter) 267

10.54.3.30 Swap(Image &rhs) 268

10.54.3.31 WritePixels(const Offset3D &offset, const Extent3D &extent, const SrcImage↩ Descriptor &imageDesc, unsigned threadCount=0) 268

10.55 LLGL::Input Class Reference 268

10.55.1 Detailed Description 269

10.55.2 Constructor & Destructor Documentation 270

10.55.2.1 Input() 270

10.55.2.2 Input(Surface &surface) 270

10.55.2.3 ~Input() 270

10.55.3 Member Function Documentation 270

10.55.3.1 Drop(Surface &surface) 270

10.55.3.2 GetAnyKeyCount() const 270

10.55.3.3 GetEnteredChars() const 270

10.55.3.4 GetMouseMotion() const 270

10.55.3.5 GetMousePosition() const 270

10.55.3.6 GetWheelMotion() const . . . 270
10.55.3.7 KeyDoubleClick(Key keyCode) const . . . 271
10.55.3.8 KeyDown(Key keyCode) const . . . 271
10.55.3.9 KeyDownRepeated(Key keyCode) const . . . 271
10.55.3.10 KeyPressed(Key keyCode) const . . . 271
10.55.3.11 KeyUp(Key keyCode) const . . . 271
10.55.3.12 Listen(Surface &surface) . . . 271
10.56 LLGL::Interface Class Reference . . . 272
10.56.1 Detailed Description . . . 272
10.56.2 Member Function Documentation . . . 273
10.56.2.1 IsInstanceOf(int id) const . . . 273
10.57 LLGL::InterfaceID Struct Reference . . . 273
10.57.1 Detailed Description . . . 273
10.57.2 Member Enumeration Documentation . . . 274
10.57.2.1 anonymous enum . . . 274
10.58 LLGL::RenderingDebugger::Message Class Reference . . . 276
10.58.1 Detailed Description . . . 277
10.58.2 Constructor & Destructor Documentation . . . 277
10.58.2.1 Message()=default . . . 277
10.58.2.2 Message(const Message &)=default . . . 277
10.58.2.3 Message(const StringView &text, const StringView &source, const StringView &groupName) . . . 277
10.58.3 Member Function Documentation . . . 277
10.58.3.1 Block() . . . 277
10.58.3.2 BlockAfter(std::size_t occurrences) . . . 277
10.58.3.3 GetGroupName() const . . . 277
10.58.3.4 GetOccurrences() const . . . 277
10.58.3.5 GetSource() const . . . 277
10.58.3.6 GetText() const . . . 278
10.58.3.7 IncOccurrence() . . . 278
10.58.3.8 IsBlocked() const . . . 278

10.58.3.9 operator=(const Message &)=default . . . 278
10.58.3.10 ToReportString(const StringView &reportTypeName) const . . . 278
10.58.4 Friends And Related Function Documentation . . . 278
10.58.4.1 RenderingDebugger . . . 278
10.59 LLGL::MetalDependentStateDescriptor Struct Reference . . . 278
10.59.1 Detailed Description . . . 279
10.59.2 Member Data Documentation . . . 279
10.59.2.1 tessFactorBufferSlot . . . 279
10.60 LLGL::MiscFlags Struct Reference . . . 279
10.60.1 Detailed Description . . . 279
10.60.2 Member Enumeration Documentation . . . 280
10.60.2.1 anonymous enum . . . 280
10.61 LLGL::NativeContextHandle Struct Reference . . . 281
10.61.1 Detailed Description . . . 281
10.61.2 Member Data Documentation . . . 282
10.61.2.1 colorMap . . . 282
10.61.2.2 display . . . 282
10.61.2.3 parentView . . . 282
10.61.2.4 parentWindow . . . 282
10.61.2.5 parentWindow . . . 282
10.61.2.6 parentWindow . . . 282
10.61.2.7 parentWindow . . . 282
10.61.2.8 screen . . . 282
10.61.2.9 visual . . . 282
10.62 LLGL::NativeHandle Struct Reference . . . 282
10.62.1 Detailed Description . . . 283
10.62.2 Member Data Documentation . . . 283
10.62.2.1 display . . . 283
10.62.2.2 view . . . 283
10.62.2.3 visual . . . 283

10.62.2.4 window . . . 283
10.62.2.5 window . . . 283
10.62.2.6 window . . . 283
10.62.2.7 window . . . 283
10.63 LLGL::NonCopyable Class Reference . . . 284
10.63.1 Detailed Description . . . 284
10.63.2 Constructor & Destructor Documentation . . . 285
10.63.2.1 NonCopyable(const NonCopyable &)=delete . . . 285
10.63.2.2 ~NonCopyable()=default . . . 285
10.63.2.3 NonCopyable()=default . . . 285
10.63.3 Member Function Documentation . . . 285
10.63.3.1 operator=(const NonCopyable &)=delete . . . 285
10.64 LLGL::Offset2D Struct Reference . . . 285
10.64.1 Detailed Description . . . 285
10.64.2 Constructor & Destructor Documentation . . . 286
10.64.2.1 Offset2D()=default . . . 286
10.64.2.2 Offset2D(const Offset2D &)=default . . . 286
10.64.2.3 Offset2D(std::int32_t x, std::int32_t y) . . . 286
10.64.3 Member Data Documentation . . . 286
10.64.3.1 x . . . 286
10.64.3.2 y . . . 286
10.65 LLGL::Offset3D Struct Reference . . . 286
10.65.1 Detailed Description . . . 287
10.65.2 Constructor & Destructor Documentation . . . 287
10.65.2.1 Offset3D()=default . . . 287
10.65.2.2 Offset3D(const Offset3D &)=default . . . 287
10.65.2.3 Offset3D(std::int32_t x, std::int32_t y, std::int32_t z) . . . 287
10.65.3 Member Data Documentation . . . 287
10.65.3.1 x . . . 287
10.65.3.2 y . . . 287

10.65.3.3 z . . . 287
10.66 LLGL::PipelineLayout Class Reference . . . 287
10.66.1 Detailed Description . . . 288
10.66.2 Member Function Documentation . . . 288
10.66.2.1 GetNumBindings() const =0 . . . 288
10.67 LLGL::PipelineLayoutDescriptor Struct Reference . . . 288
10.67.1 Detailed Description . . . 289
10.67.2 Member Data Documentation . . . 289
10.67.2.1 bindings . . . 289
10.68 LLGL::PipelineState Class Reference . . . 289
10.68.1 Detailed Description . . . 290
10.68.2 Member Function Documentation . . . 290
10.68.2.1 GetReport() const =0 . . . 290
10.69 LLGL::ProfileTimeRecord Struct Reference . . . 290
10.69.1 Detailed Description . . . 290
10.69.2 Member Data Documentation . . . 291
10.69.2.1 annotation . . . 291
10.69.2.2 elapsedTime . . . 291
10.70 LLGL::QueryHeap Class Reference . . . 291
10.70.1 Detailed Description . . . 291
10.70.2 Constructor & Destructor Documentation . . . 292
10.70.2.1 QueryHeap(const QueryType type) . . . 292
10.70.3 Member Function Documentation . . . 292
10.70.3.1 GetType() const . . . 292
10.71 LLGL::QueryHeapDescriptor Struct Reference . . . 292
10.71.1 Detailed Description . . . 292
10.71.2 Member Data Documentation . . . 292
10.71.2.1 numQueries . . . 292
10.71.2.2 renderCondition . . . 293
10.71.2.3 type . . . 293

10.72 LLGL::QueryPipelineStatistics Struct Reference . . . 293
10.72.1 Detailed Description . . . 294
10.72.2 Member Data Documentation . . . 294
10.72.2.1 clippingInvocations . . . 294
10.72.2.2 clippingPrimitives . . . 295
10.72.2.3 computeShaderInvocations . . . 295
10.72.2.4 fragmentShaderInvocations . . . 295
10.72.2.5 geometryShaderInvocations . . . 295
10.72.2.6 geometryShaderPrimitives . . . 295
10.72.2.7 inputAssemblyPrimitives . . . 295
10.72.2.8 inputAssemblyVertices . . . 295
10.72.2.9 tessControlShaderInvocations . . . 295
10.72.2.10 tessEvaluationShaderInvocations . . . 295
10.72.2.11 vertexShaderInvocations . . . 295
10.73 LLGL::RasterizerDescriptor Struct Reference . . . 296
10.73.1 Detailed Description . . . 296
10.73.2 Member Data Documentation . . . 296
10.73.2.1 antiAliasedLineEnabled . . . 296
10.73.2.2 conservativeRasterization . . . 297
10.73.2.3 cullMode . . . 297
10.73.2.4 depthBias . . . 297
10.73.2.5 depthClampEnabled . . . 297
10.73.2.6 discardEnabled . . . 297
10.73.2.7 frontCCW . . . 297
10.73.2.8 lineWidth . . . 298
10.73.2.9 multiSampleEnabled . . . 298
10.73.2.10 polygonMode . . . 298
10.73.2.11 scissorTestEnabled . . . 298
10.74 LLGL::RendererConfigurationOpenGL Struct Reference . . . 298
10.74.1 Detailed Description . . . 299

10.74.2 Member Data Documentation . . . 299
10.74.2.1 contextProfile . . . 299
10.74.2.2 majorVersion . . . 299
10.74.2.3 minorVersion . . . 299
10.75 LLGL::RendererConfigurationOpenGLES3 Struct Reference . . . 300
10.75.1 Detailed Description . . . 300
10.75.2 Member Data Documentation . . . 300
10.75.2.1 majorVersion . . . 300
10.75.2.2 minorVersion . . . 300
10.76 LLGL::RendererConfigurationVulkan Struct Reference . . . 300
10.76.1 Detailed Description . . . 301
10.76.2 Member Data Documentation . . . 301
10.76.2.1 application . . . 301
10.76.2.2 enabledLayers . . . 301
10.76.2.3 minDeviceMemoryAllocationSize . . . 301
10.76.2.4 reduceDeviceMemoryFragmentation . . . 302
10.77 LLGL::RendererID Struct Reference . . . 302
10.77.1 Detailed Description . . . 303
10.77.2 Member Data Documentation . . . 303
10.77.2.1 Direct3D10 . . . 303
10.77.2.2 Direct3D11 . . . 303
10.77.2.3 Direct3D12 . . . 303
10.77.2.4 Direct3D9 . . . 303
10.77.2.5 Metal . . . 303
10.77.2.6 Null . . . 303
10.77.2.7 OpenGL . . . 303
10.77.2.8 OpenGLES1 . . . 304
10.77.2.9 OpenGLES2 . . . 304
10.77.2.10 OpenGLES3 . . . 304
10.77.2.11 Reserved . . . 304

10.77.2.12 Undefined . . . 304
10.77.2.13 Vulkan . . . 304
10.78 LLGL::RendererInfo Struct Reference . . . 304
10.78.1 Detailed Description . . . 305
10.78.2 Member Data Documentation . . . 305
10.78.2.1 deviceName . . . 305
10.78.2.2 extensionNames . . . 305
10.78.2.3 rendererName . . . 305
10.78.2.4 shadingLanguageName . . . 305
10.78.2.5 vendorName . . . 305
10.79 LLGL::RenderingCapabilities Struct Reference . . . 305
10.79.1 Detailed Description . . . 306
10.79.2 Member Data Documentation . . . 306
10.79.2.1 clippingRange . . . 306
10.79.2.2 features . . . 306
10.79.2.3 limits . . . 306
10.79.2.4 screenOrigin . . . 307
10.79.2.5 shadingLanguages . . . 307
10.79.2.6 textureFormats . . . 307
10.80 LLGL::RenderingDebugger Class Reference . . . 307
10.80.1 Detailed Description . . . 308
10.80.2 Constructor & Destructor Documentation . . . 308
10.80.2.1 RenderingDebugger() . . . 308
10.80.2.2 ~RenderingDebugger() . . . 308
10.80.3 Member Function Documentation . . . 309
10.80.3.1 OnError(ErrorType type, Message &message) . . . 309
10.80.3.2 OnWarning(WarningType type, Message &message) . . . 309
10.80.3.3 PostError(const ErrorType type, const StringView &message) . . . 309
10.80.3.4 PostWarning(const WarningType type, const StringView &message) . . . 309
10.80.3.5 SetDebugGroup(const char *name) . . . 310

10.80.3.6 SetSource(const char *source) . . . 310

10.81 LLGL::RenderingFeatures Struct Reference . . . 310

10.81.1 Detailed Description . . . 312

10.81.2 Member Data Documentation . . . 312

10.81.2.1 has3DTextures . . . 312

10.81.2.2 hasArrayTextures . . . 312

10.81.2.3 hasBufferViews . . . 312

10.81.2.4 hasComputeShaders . . . 313

10.81.2.5 hasConservativeRasterization . . . 313

10.81.2.6 hasConstantBuffers . . . 313

10.81.2.7 hasCubeArrayTextures . . . 313

10.81.2.8 hasCubeTextures . . . 313

10.81.2.9 hasDirectResourceBinding . . . 314

10.81.2.10 hasGeometryShaders . . . 314

10.81.2.11 hasIndirectDrawing . . . 314

10.81.2.12 hasInstancing . . . 314

10.81.2.13 hasLogicOp . . . 314

10.81.2.14 hasMultiSampleTextures . . . 315

10.81.2.15 hasOffsetInstancing . . . 315

10.81.2.16 hasPipelineStatistics . . . 315

10.81.2.17 hasRenderCondition . . . 315

10.81.2.18 hasRenderTargets . . . 315

10.81.2.19 hasSamplers . . . 315

10.81.2.20 hasStorageBuffers . . . 316

10.81.2.21 hasStreamOutputs . . . 316

10.81.2.22 hasTessellationShaders . . . 316

10.81.2.23 hasTessellatorStage . . . 316

10.81.2.24 hasTextureViews . . . 317

10.81.2.25 hasTextureViewSwizzle . . . 317

10.81.2.26 hasUniforms . . . 317

10.81.2.27 hasViewportArrays . . . 317
10.82 LLGL::RenderingLimits Struct Reference . . . 318
10.82.1 Detailed Description . . . 319
10.82.2 Member Data Documentation . . . 319
10.82.2.1 lineWidthRange . . . 319
10.82.2.2 max1DTextureSize . . . 319
10.82.2.3 max2DTextureSize . . . 319
10.82.2.4 max3DTextureSize . . . 319
10.82.2.5 maxAnisotropy . . . 320
10.82.2.6 maxBufferSize . . . 320
10.82.2.7 maxColorAttachments . . . 320
10.82.2.8 maxComputeShaderWorkGroups . . . 320
10.82.2.9 maxComputeShaderWorkGroupSize . . . 320
10.82.2.10 maxConstantBufferSize . . . 321
10.82.2.11 maxCubeTextureSize . . . 321
10.82.2.12 maxPatchVertices . . . 321
10.82.2.13 maxStreamOutputs . . . 321
10.82.2.14 maxTessFactor . . . 322
10.82.2.15 maxTextureArrayLayers . . . 322
10.82.2.16 maxViewports . . . 322
10.82.2.17 maxViewportSize . . . 322
10.82.2.18 minConstantBufferAlignment . . . 323
10.82.2.19 minSampledBufferAlignment . . . 323
10.82.2.20 minStorageBufferAlignment . . . 323
10.83 LLGL::RenderingProfiler Class Reference . . . 323
10.83.1 Detailed Description . . . 324
10.83.2 Member Function Documentation . . . 324
10.83.2.1 Accumulate(const FrameProfile &profile) . . . 324
10.83.2.2 NextProfile(FrameProfile *outputProfile=nullptr) . . . 324
10.83.3 Member Data Documentation . . . 324

10.83.3.1 frameProfile . . . 324
10.83.3.2 timeRecordingEnabled . . . 325
10.84 LLGL::RenderPass Class Reference . . . 325
10.84.1 Detailed Description . . . 325
10.85 LLGL::RenderPassDescriptor Struct Reference . . . 326
10.85.1 Detailed Description . . . 326
10.85.2 Member Data Documentation . . . 326
10.85.2.1 colorAttachments . . . 326
10.85.2.2 depthAttachment . . . 327
10.85.2.3 samples . . . 327
10.85.2.4 stencilAttachment . . . 327
10.86 LLGL::RenderSystem Class Reference . . . 327
10.86.1 Detailed Description . . . 330
10.86.2 Constructor & Destructor Documentation . . . 331
10.86.2.1 ~RenderSystem() . . . 331
10.86.2.2 RenderSystem() . . . 331
10.86.3 Member Function Documentation . . . 331
10.86.3.1 AssertCreateBuffer(const BufferDescriptor &bufferDesc, std::uint64_t maxSize) . 331
10.86.3.2 AssertCreateBufferArray(std::uint32_t numBuffers, Buffer *const *bufferArray) . 331
10.86.3.3 AssertCreateRenderTarget(const RenderTargetDescriptor &renderTargetDesc) . 331
10.86.3.4 AssertCreateShader(const ShaderDescriptor &shaderDesc) . . . 331
10.86.3.5 AssertImageDataSize(std::size_t dataSize, std::size_t requiredDataSize, const char *info=nullptr) . . . 331
10.86.3.6 CopyTextureImageData(const DstImageDescriptor &dstImageDesc, const Extent3D &extent, const Format format, const void *data, std::size_t rowStride=0) 331
10.86.3.7 CreateBuffer(const BufferDescriptor &bufferDesc, const void *initialData=nullptr)=0 331
10.86.3.8 CreateBufferArray(std::uint32_t numBuffers, Buffer *const *bufferArray)=0 . . . 332
10.86.3.9 CreateCommandBuffer(const CommandBufferDescriptor &commandBufferDesc={})=0 . . . 332
10.86.3.10 CreateFence()=0 . . . 333
10.86.3.11 CreatePipelineLayout(const PipelineLayoutDescriptor &pipelineLayoutDesc)=0 . 333
10.86.3.12 CreatePipelineState(const Blob &serializedCache)=0 . . . 333

10.86.3.13 CreatePipelineState(const GraphicsPipelineDescriptor &pipelineStateDesc, std::unique_ptr< Blob > *serializedCache=nullptr)=0 334

10.86.3.14 CreatePipelineState(const ComputePipelineDescriptor &pipelineStateDesc, std::unique_ptr< Blob > *serializedCache=nullptr)=0 335

10.86.3.15 CreateQueryHeap(const QueryHeapDescriptor &queryHeapDesc)=0 335

10.86.3.16 CreateRenderPass(const RenderPassDescriptor &renderPassDesc)=0 335

10.86.3.17 CreateRenderTarget(const RenderTargetDescriptor &renderTargetDesc)=0 . . . 335

10.86.3.18 CreateResourceHeap(const ResourceHeapDescriptor &resourceHeapDesc, const ArrayView< ResourceViewDescriptor > &initialResourceViews={})=0 . . . 336

10.86.3.19 CreateSampler(const SamplerDescriptor &samplerDesc)=0 336

10.86.3.20 CreateShader(const ShaderDescriptor &shaderDesc)=0 337

10.86.3.21 CreateSwapChain(const SwapChainDescriptor &swapChainDesc, const std↩::shared_ptr< Surface > &surface={})=0 337

10.86.3.22 CreateTexture(const TextureDescriptor &textureDesc, const SrcImageDescriptor *imageDesc=nullptr)=0 337

10.86.3.23 FindModules() 338

10.86.3.24 GetCommandQueue()=0 338

10.86.3.25 GetName() const 338

10.86.3.26 GetRendererID() const 338

10.86.3.27 GetRendererInfo() const 338

10.86.3.28 GetRenderingCaps() const 338

10.86.3.29 Load(const RenderSystemDescriptor &renderSystemDesc, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr) 338

10.86.3.30 MapBuffer(Buffer &buffer, const CPUAccess access)=0 339

10.86.3.31 MapBuffer(Buffer &buffer, const CPUAccess access, std::uint64_t offset, std↩::uint64_t length)=0 340

10.86.3.32 ReadBuffer(Buffer &buffer, std::uint64_t offset, void *data, std::uint64_t dataSize)=0 340

10.86.3.33 ReadTexture(Texture &texture, const TextureRegion &textureRegion, const Dst↩ImageDescriptor &imageDesc)=0 341

10.86.3.34 Release(SwapChain &swapChain)=0 342

10.86.3.35 Release(CommandBuffer &commandBuffer)=0 342

10.86.3.36 Release(Buffer &buffer)=0 342

10.86.3.37 Release(BufferArray &bufferArray)=0 342

10.86.3.38 Release(Texture &texture)=0 342

10.86.3.39 Release(Sampler &sampler)=0 . . . 342
10.86.3.40 Release(ResourceHeap &resourceHeap)=0 . . . 343
10.86.3.41 Release(RenderPass &renderPass)=0 . . . 343
10.86.3.42 Release(RenderTarget &renderTarget)=0 . . . 343
10.86.3.43 Release(Shader &shader)=0 . . . 343
10.86.3.44 Release(PipelineLayout &pipelineLayout)=0 . . . 343
10.86.3.45 Release(PipelineState &pipelineState)=0 . . . 343
10.86.3.46 Release(QueryHeap &queryHeap)=0 . . . 343
10.86.3.47 Release(Fence &fence)=0 . . . 343
10.86.3.48 SetRendererInfo(const RendererInfo &info) . . . 343
10.86.3.49 SetRenderingCaps(const RenderingCapabilities &caps) . . . 343
10.86.3.50 Unload(RenderSystemPtr &&renderSystem) . . . 344
10.86.3.51 UnmapBuffer(Buffer &buffer)=0 . . . 344
10.86.3.52 WriteBuffer(Buffer &buffer, std::uint64_t offset, const void *data, std::uint64_t dataSize)=0 . . . 344
10.86.3.53 WriteResourceHeap(ResourceHeap &resourceHeap, std::uint32_t firstDescriptor, const ArrayView< ResourceViewDescriptor > &resourceViews)=0 . 345
10.86.3.54 WriteTexture(Texture &texture, const TextureRegion &textureRegion, const SrcImageDescriptor &imageDesc)=0 . . . 345
10.87 LLGL::RenderSystemChild Class Reference . . . 346
10.87.1 Detailed Description . . . 346
10.87.2 Member Function Documentation . . . 346
10.87.2.1 SetName(const char *name) . . . 346
10.88 LLGL::RenderSystemDeleter Class Reference . . . 347
10.88.1 Detailed Description . . . 348
10.88.2 Member Typedef Documentation . . . 348
10.88.2.1 RenderSystemDeleterFuncPtr . . . 348
10.88.3 Constructor & Destructor Documentation . . . 348
10.88.3.1 RenderSystemDeleter() noexcept=default . . . 348
10.88.3.2 RenderSystemDeleter(const RenderSystemDeleter &) noexcept=default . . . 348
10.88.3.3 RenderSystemDeleter(RenderSystemDeleterFuncPtr deleterFuncPtr) . . . 348
10.88.4 Member Function Documentation . . . 348

10.88.4.1 operator()(RenderSystem *ptr) const . . . 348
10.88.4.2 operator=(const RenderSystemDeleter &) noexcept=default . . . 348
10.89 LLGL::RenderSystemDescriptor Struct Reference . . . 349
10.89.1 Detailed Description . . . 349
10.89.2 Constructor & Destructor Documentation . . . 349
10.89.2.1 RenderSystemDescriptor()=default . . . 349
10.89.2.2 RenderSystemDescriptor(const RenderSystemDescriptor &)=default . . . 349
10.89.2.3 RenderSystemDescriptor(const std::string &moduleName) . . . 349
10.89.2.4 RenderSystemDescriptor(const char *moduleName) . . . 350
10.89.3 Member Function Documentation . . . 350
10.89.3.1 operator=(const RenderSystemDescriptor &)=default . . . 350
10.89.4 Member Data Documentation . . . 350
10.89.4.1 debugCallback . . . 350
10.89.4.2 moduleName . . . 350
10.89.4.3 rendererConfig . . . 350
10.89.4.4 rendererConfigSize . . . 351
10.90 LLGL::RenderTarget Class Reference . . . 351
10.90.1 Detailed Description . . . 352
10.90.2 Member Function Documentation . . . 352
10.90.2.1 GetNumColorAttachments() const =0 . . . 352
10.90.2.2 GetRenderPass() const =0 . . . 352
10.90.2.3 GetResolution() const =0 . . . 353
10.90.2.4 GetSamples() const =0 . . . 353
10.90.2.5 HasDepthAttachment() const =0 . . . 353
10.90.2.6 HasStencilAttachment() const =0 . . . 354
10.90.2.7 ValidateMipResolution(const Texture &texture, std::uint32_t mipLevel) . . . 354
10.90.2.8 ValidateResolution(const Extent2D &resolution) . . . 354
10.91 LLGL::RenderTargetDescriptor Struct Reference . . . 355
10.91.1 Detailed Description . . . 355
10.91.2 Member Data Documentation . . . 355

10.91.2.1 attachments . . . 355
10.91.2.2 customMultiSampling . . . 356
10.91.2.3 renderPass . . . 356
10.91.2.4 resolution . . . 356
10.91.2.5 samples . . . 357
10.92 LLGL::Report Class Reference . . . 357
10.92.1 Detailed Description . . . 357
10.92.2 Member Function Documentation . . . 358
10.92.2.1 GetText() const =0 . . . 358
10.92.2.2 HasErrors() const =0 . . . 358
10.93 LLGL::ResizeBuffersFlags Struct Reference . . . 358
10.93.1 Detailed Description . . . 358
10.93.2 Member Enumeration Documentation . . . 358
10.93.2.1 anonymous enum . . . 358
10.94 LLGL::Resource Class Reference . . . 359
10.94.1 Detailed Description . . . 359
10.94.2 Member Function Documentation . . . 360
10.94.2.1 GetResourceType() const =0 . . . 360
10.95 LLGL::ResourceHeap Class Reference . . . 360
10.95.1 Detailed Description . . . 361
10.95.2 Member Function Documentation . . . 361
10.95.2.1 GetNumDescriptorSets() const =0 . . . 361
10.96 LLGL::ResourceHeapDescriptor Struct Reference . . . 361
10.96.1 Detailed Description . . . 362
10.96.2 Constructor & Destructor Documentation . . . 362
10.96.2.1 ResourceHeapDescriptor()=default . . . 362
10.96.2.2 ResourceHeapDescriptor(PipelineLayout *pipelineLayout, std::uint32_t num↩ ResourceViews=0, long barrierFlags=0) . . . 362
10.96.3 Member Data Documentation . . . 362
10.96.3.1 barrierFlags . . . 362
10.96.3.2 numResourceViews . . . 363

10.96.3.3 pipelineLayout . . . 363

10.97 LLGL::ResourceViewDescriptor Struct Reference . . . 363

10.97.1 Detailed Description . . . 364

10.97.2 Constructor & Destructor Documentation . . . 364

10.97.2.1 ResourceViewDescriptor()=default . . . 364

10.97.2.2 ResourceViewDescriptor(Resource *resource) . . . 364

10.97.2.3 ResourceViewDescriptor(Texture *texture, const TextureViewDescriptor &subresourceDesc) . . . 364

10.97.2.4 ResourceViewDescriptor(Buffer *buffer, const BufferViewDescriptor &subresourceDesc) . . . 364

10.97.3 Member Data Documentation . . . 364

10.97.3.1 bufferView . . . 364

10.97.3.2 initialCount . . . 365

10.97.3.3 resource . . . 365

10.97.3.4 textureView . . . 365

10.98 LLGL::Sampler Class Reference . . . 366

10.98.1 Detailed Description . . . 366

10.98.2 Constructor & Destructor Documentation . . . 366

10.98.2.1 Sampler()=default . . . 366

10.98.3 Member Function Documentation . . . 366

10.98.3.1 GetResourceType() const override final . . . 366

10.99 LLGL::SamplerDescriptor Struct Reference . . . 367

10.99.1 Detailed Description . . . 367

10.99.2 Member Data Documentation . . . 367

10.99.2.1 addressModeU . . . 367

10.99.2.2 addressModeV . . . 368

10.99.2.3 addressModeW . . . 368

10.99.2.4 borderColor . . . 368

10.99.2.5 compareEnabled . . . 368

10.99.2.6 compareOp . . . 368

10.99.2.7 magFilter . . . 368

10.99.2.8 maxAnisotropy . . . 368
10.99.2.9 maxLOD . . . 368
10.99.2.10 minFilter . . . 369
10.99.2.11 minLOD . . . 369
10.99.2.12 mipMapFilter . . . 369
10.99.2.13 mipMapLODBias . . . 369
10.99.2.14 mipMapping . . . 369
10.100 LLGL::Scissor Struct Reference . . . 370
10.100.1 Detailed Description . . . 370
10.100.2 Constructor & Destructor Documentation . . . 370
10.100.2.1 Scissor()=default . . . 370
10.100.2.2 Scissor(const Scissor &)=default . . . 370
10.100.2.3 Scissor(std::int32_t x, std::int32_t y, std::int32_t width, std::int32_t height) . . . 370
10.100.2.4 Scissor(const Offset2D &offset, const Extent2D &extent) . . . 371
10.100.3 Member Data Documentation . . . 371
10.100.3.1 height . . . 371
10.100.3.2 width . . . 371
10.100.3.3 x . . . 371
10.100.3.4 y . . . 371
10.101 LLGL::Shader Class Reference . . . 371
10.101.1 Detailed Description . . . 372
10.101.2 Constructor & Destructor Documentation . . . 372
10.101.2.1 Shader(const ShaderType type) . . . 372
10.101.3 Member Function Documentation . . . 372
10.101.3.1 FindUniformLocation(const char *name) const . . . 372
10.101.3.2 GetReport() const =0 . . . 373
10.101.3.3 GetStageFlags() const . . . 373
10.101.3.4 GetType() const . . . 373
10.101.3.5 IsPostTessellationVertex() const . . . 373
10.101.3.6 Reflect(ShaderReflection &reflection) const =0 . . . 373

10.102 LGL::ShaderCompileFlags Struct Reference . . . 374
10.102.1 Detailed Description . . . 375
10.102.2 Member Enumeration Documentation . . . 375
10.102.2.1 anonymous enum . . . 375
10.103 LGL::ShaderDescriptor Struct Reference . . . 377
10.103.1 Detailed Description . . . 378
10.103.2 Constructor & Destructor Documentation . . . 378
10.103.2.1 ShaderDescriptor()=default . . . 378
10.103.2.2 ShaderDescriptor(const ShaderDescriptor &)=default . . . 378
10.103.2.3 ShaderDescriptor(const ShaderType type, const char *source) . . . 378
10.103.2.4 ShaderDescriptor(const ShaderType type, const char *source, const char *entryPoint, const char *profile, long flags=0) . . . 378
10.103.3 Member Function Documentation . . . 378
10.103.3.1 operator=(const ShaderDescriptor &)=default . . . 378
10.103.4 Member Data Documentation . . . 378
10.103.4.1 compute . . . 378
10.103.4.2 defines . . . 379
10.103.4.3 entryPoint . . . 379
10.103.4.4 flags . . . 379
10.103.4.5 fragment . . . 379
10.103.4.6 profile . . . 380
10.103.4.7 source . . . 380
10.103.4.8 sourceSize . . . 380
10.103.4.9 sourceType . . . 381
10.103.4.10 type . . . 381
10.103.4.11 vertex . . . 381
10.104 LGL::ShaderMacro Struct Reference . . . 381
10.104.1 Detailed Description . . . 382
10.104.2 Constructor & Destructor Documentation . . . 382
10.104.2.1 ShaderMacro()=default . . . 382
10.104.2.2 ShaderMacro(const ShaderMacro &)=default . . . 382

10.104.2.3 ShaderMacro(const char *name, const char *definition=nullptr) . . . 382
10.104.3 Member Function Documentation . . . 382
10.104.3.1 operator=(const ShaderMacro &)=default . . . 382
10.104.4 Member Data Documentation . . . 382
10.104.4.1 definition . . . 382
10.104.4.2 name . . . 382
10.105 LGL::ShaderReflection Struct Reference . . . 383
10.105.1 Detailed Description . . . 383
10.105.2 Member Data Documentation . . . 383
10.105.2.1 compute . . . 383
10.105.2.2 fragment . . . 383
10.105.2.3 resources . . . 383
10.105.2.4 uniforms . . . 384
10.105.2.5 vertex . . . 384
10.106 LGL::ShaderResource Struct Reference . . . 384
10.106.1 Detailed Description . . . 384
10.106.2 Member Data Documentation . . . 385
10.106.2.1 binding . . . 385
10.106.2.2 constantBufferSize . . . 385
10.106.2.3 storageBufferType . . . 385
10.107 LGL::ShaderUniform Struct Reference . . . 385
10.107.1 Detailed Description . . . 386
10.107.2 Member Data Documentation . . . 386
10.107.2.1 location . . . 386
10.107.2.2 name . . . 386
10.107.2.3 size . . . 386
10.107.2.4 type . . . 386
10.108 LGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy > Class Template Reference . . . 386
10.108.1 Detailed Description . . . 388
10.108.2 Member Typedef Documentation . . . 388

10.108.2.1 const_iterator . . . 388
10.108.2.2 const_pointer . . . 388
10.108.2.3 const_reference . . . 388
10.108.2.4 const_reverse_iterator . . . 388
10.108.2.5 difference_type . . . 389
10.108.2.6 iterator . . . 389
10.108.2.7 pointer . . . 389
10.108.2.8 reference . . . 389
10.108.2.9 reverse_iterator . . . 389
10.108.2.10 size_type . . . 389
10.108.2.11 value_type . . . 389
10.108.3 Constructor & Destructor Documentation . . . 389
10.108.3.1 SmallVector() . . . 389
10.108.3.2 SmallVector(const SmallVector &other) . . . 389
10.108.3.3 SmallVector(SmallVector &&other) . . . 389
10.108.3.4 SmallVector(InputIter from, InputIter to) . . . 389
10.108.3.5 SmallVector(const std::initializer_list< T > &list) . . . 389
10.108.3.6 SmallVector(const OtherContainer< T, OtherAllocator > &other) . . . 390
10.108.3.7 ~SmallVector() . . . 390
10.108.4 Member Function Documentation . . . 390
10.108.4.1 at(size_type pos) noexcept . . . 390
10.108.4.2 at(size_type pos) const noexcept . . . 390
10.108.4.3 back() noexcept . . . 390
10.108.4.4 back() const noexcept . . . 390
10.108.4.5 begin() . . . 390
10.108.4.6 begin() const . . . 390
10.108.4.7 capacity() const noexcept . . . 390
10.108.4.8 cbegin() const . . . 390
10.108.4.9 cend() const . . . 390
10.108.4.10 clear() . . . 390

10.108.4.11 crbegin() const . . . 391
10.108.4.12 crend() const . . . 391
10.108.4.13 data() noexcept . . . 391
10.108.4.14 data() const noexcept . . . 391
10.108.4.15 empty() const noexcept . . . 391
10.108.4.16 end() . . . 391
10.108.4.17 end() const . . . 391
10.108.4.18 erase(const_iterator pos) . . . 391
10.108.4.19 erase(const_iterator from, const_iterator to) . . . 391
10.108.4.20 front() noexcept . . . 391
10.108.4.21 front() const noexcept . . . 391
10.108.4.22 insert(const_iterator pos, const value_type &value) . . . 391
10.108.4.23 insert(const_iterator pos, InputIter from, InputIter to) . . . 391
10.108.4.24 insert(const_iterator pos, const std::initializer_list< T > &list) . . . 392
10.108.4.25 operator ArrayView< T >() const . . . 392
10.108.4.26 operator=(const SmallVector &rhs) . . . 392
10.108.4.27 operator=(SmallVector &&rhs) . . . 392
10.108.4.28 operator[](size_type pos) const . . . 392
10.108.4.29 operator[](size_type pos) . . . 392
10.108.4.30 pop_back() . . . 392
10.108.4.31 push_back(const value_type &value) . . . 392
10.108.4.32 push_back(value_type &&value) . . . 392
10.108.4.33 rbegin() . . . 392
10.108.4.34 rbegin() const . . . 392
10.108.4.35 rend() . . . 392
10.108.4.36 rend() const . . . 393
10.108.4.37 reserve(size_type size) . . . 393
10.108.4.38 resize(size_type size) . . . 393
10.108.4.39 resize(size_type size, const value_type &value) . . . 393
10.108.4.40 shrink_to_fit() . . . 393

10.108.4.4 size() const noexcept . . . 393
10.108.4.4 swap(SmallVector &other) . . . 393
10.109 LLGL::SrcImageDescriptor Struct Reference . . . 393
10.109.1 Detailed Description . . . 394
10.109.2 Constructor & Destructor Documentation . . . 394
10.109.2.1 SrcImageDescriptor()=default . . . 394
10.109.2.2 SrcImageDescriptor(const SrcImageDescriptor &)=default . . . 394
10.109.2.3 SrcImageDescriptor(ImageFormat format, DataType dataType, const void *data, std::size_t dataSize) . . . 394
10.109.3 Member Data Documentation . . . 394
10.109.3.1 data . . . 394
10.109.3.2 dataSize . . . 395
10.109.3.3 dataType . . . 395
10.109.3.4 format . . . 395
10.110 LLGL::StageFlags Struct Reference . . . 395
10.110.1 Detailed Description . . . 395
10.110.2 Member Enumeration Documentation . . . 396
10.110.2.1 anonymous enum . . . 396
10.111 LLGL::StencilDescriptor Struct Reference . . . 396
10.111.1 Detailed Description . . . 396
10.111.2 Member Data Documentation . . . 397
10.111.2.1 back . . . 397
10.111.2.2 front . . . 397
10.111.2.3 referenceDynamic . . . 397
10.111.2.4 testEnabled . . . 397
10.112 LLGL::StencilFaceDescriptor Struct Reference . . . 397
10.112.1 Detailed Description . . . 398
10.112.2 Member Data Documentation . . . 398
10.112.2.1 compareOp . . . 398
10.112.2.2 depthFailOp . . . 398
10.112.2.3 depthPassOp . . . 398

10.112.2.4 readMask . . . 398
10.112.2.5 reference . . . 399
10.112.2.6 stencilFailOp . . . 399
10.112.2.7 writeMask . . . 399
10.113 LGL::Surface Class Reference . . . 400
10.113.1 Detailed Description . . . 400
10.113.2 Member Function Documentation . . . 401
10.113.2.1 AdaptForVideoMode(Extent2D *resolution, bool *fullscreen)=0 . . . 401
10.113.2.2 FindResidentDisplay() const =0 . . . 402
10.113.2.3 GetContentSize() const =0 . . . 402
10.113.2.4 GetNativeHandle(void *nativeHandle, std::size_t nativeHandleSize) const =0 . . . 402
10.113.2.5 ProcessEvents()=0 . . . 403
10.113.2.6 ResetPixelFormat()=0 . . . 403
10.114 LGL::SwapChain Class Reference . . . 404
10.114.1 Detailed Description . . . 405
10.114.2 Constructor & Destructor Documentation . . . 405
10.114.2.1 ~SwapChain() . . . 405
10.114.2.2 SwapChain() . . . 405
10.114.2.3 SwapChain(const SwapChainDescriptor &desc) . . . 405
10.114.3 Member Function Documentation . . . 406
10.114.3.1 GetColorFormat() const =0 . . . 406
10.114.3.2 GetDepthStencilFormat() const =0 . . . 406
10.114.3.3 GetNumColorAttachments() const final . . . 406
10.114.3.4 GetResolution() const final . . . 406
10.114.3.5 GetSurface() const . . . 407
10.114.3.6 HasDepthAttachment() const final . . . 407
10.114.3.7 HasStencilAttachment() const final . . . 407
10.114.3.8 Present()=0 . . . 407
10.114.3.9 ResetDisplayFullscreenMode() . . . 407
10.114.3.10 ResizeBuffers(const Extent2D &resolution, long flags=0) . . . 407

10.114.3.11 ResizeBuffersPrimary(const Extent2D &resolution)=0 . . . 408
10.114.3.12 SetDisplayFullscreenMode(const Extent2D &resolution) . . . 408
10.114.3.13 SetOrCreateSurface(const std::shared_ptr< Surface > &surface, const Extent2D &size, bool fullscreen, const void *windowContext) . . . 408
10.114.3.14 SetVsyncInterval(std::uint32_t vsyncInterval)=0 . . . 409
10.114.3.15 ShareSurfaceAndConfig(SwapChain &other) . . . 409
10.114.3.16 SwitchFullscreen(bool enable) . . . 409
10.115 LGL::SwapChainDescriptor Struct Reference . . . 410
10.115.1 Detailed Description . . . 410
10.115.2 Member Data Documentation . . . 411
10.115.2.1 colorBits . . . 411
10.115.2.2 depthBits . . . 411
10.115.2.3 fullscreen . . . 411
10.115.2.4 resolution . . . 411
10.115.2.5 samples . . . 412
10.115.2.6 stencilBits . . . 412
10.115.2.7 swapBuffers . . . 412
10.116 LGL::TessellationDescriptor Struct Reference . . . 412
10.116.1 Detailed Description . . . 413
10.116.2 Member Data Documentation . . . 413
10.116.2.1 indexFormat . . . 413
10.116.2.2 maxTessFactor . . . 413
10.116.2.3 outputWindingCCW . . . 414
10.116.2.4 partition . . . 414
10.117 LGL::Texture Class Reference . . . 414
10.117.1 Detailed Description . . . 415
10.117.2 Constructor & Destructor Documentation . . . 415
10.117.2.1 Texture(const TextureType type, long bindFlags) . . . 415
10.117.3 Member Function Documentation . . . 415
10.117.3.1 GetBindFlags() const . . . 415
10.117.3.2 GetDesc() const =0 . . . 415

10.117.3.3 GetFormat() const =0 . . . 416
10.117.3.4 GetMemoryFootprint() const . . . 416
10.117.3.5 GetMemoryFootprint(const Extent3D &extent, const TextureSubresource &subresource) const . . . 416
10.117.3.6 GetMipExtent(std::uint32_t mipLevel) const =0 . . . 416
10.117.3.7 GetResourceType() const override final . . . 417
10.117.3.8 GetType() const . . . 417
10.118 LGL::TextureDescriptor Struct Reference . . . 417
10.118.1 Detailed Description . . . 418
10.118.2 Member Data Documentation . . . 418
10.118.2.1 arrayLayers . . . 418
10.118.2.2 bindFlags . . . 419
10.118.2.3 clearValue . . . 419
10.118.2.4 extent . . . 420
10.118.2.5 format . . . 420
10.118.2.6 mipLevels . . . 420
10.118.2.7 miscFlags . . . 420
10.118.2.8 samples . . . 421
10.118.2.9 type . . . 421
10.119 LGL::TextureLocation Struct Reference . . . 421
10.119.1 Detailed Description . . . 422
10.119.2 Constructor & Destructor Documentation . . . 422
10.119.2.1 TextureLocation()=default . . . 422
10.119.2.2 TextureLocation(const TextureLocation &)=default . . . 422
10.119.2.3 TextureLocation(const Offset3D &offset, std::uint32_t arrayLayer=0, std::uint32_t mipLevel=0) . . . 422
10.119.3 Member Data Documentation . . . 422
10.119.3.1 arrayLayer . . . 422
10.119.3.2 mipLevel . . . 422
10.119.3.3 offset . . . 423
10.120 LGL::TextureRegion Struct Reference . . . 423

10.120.1 Detailed Description . . . 423
10.120.2 Constructor & Destructor Documentation . . . 424
10.120.2.1 TextureRegion()=default . . . 424
10.120.2.2 TextureRegion(const TextureRegion &)=default . . . 424
10.120.2.3 TextureRegion(const Offset3D &offset, const Extent3D &extent) . . . 424
10.120.2.4 TextureRegion(const TextureSubresource &subresource, const Offset3D &offset, const Extent3D &extent) . . . 424
10.120.3 Member Data Documentation . . . 424
10.120.3.1 extent . . . 424
10.120.3.2 offset . . . 424
10.120.3.3 subresource . . . 424
10.121 LLGL::TextureSubresource Struct Reference . . . 425
10.121.1 Detailed Description . . . 425
10.121.2 Constructor & Destructor Documentation . . . 425
10.121.2.1 TextureSubresource()=default . . . 425
10.121.2.2 TextureSubresource(const TextureSubresource &)=default . . . 425
10.121.2.3 TextureSubresource(std::uint32_t baseArrayLayer, std::uint32_t baseMipLevel) . . . 425
10.121.2.4 TextureSubresource(std::uint32_t baseArrayLayer, std::uint32_t numArrayLayers, std::uint32_t baseMipLevel, std::uint32_t numMipLevels) . . . 426
10.121.3 Member Data Documentation . . . 426
10.121.3.1 baseArrayLayer . . . 426
10.121.3.2 baseMipLevel . . . 426
10.121.3.3 numArrayLayers . . . 426
10.121.3.4 numMipLevels . . . 427
10.122 LLGL::TextureSwizzleRGBA Struct Reference . . . 427
10.122.1 Detailed Description . . . 427
10.122.2 Member Data Documentation . . . 428
10.122.2.1 a . . . 428
10.122.2.2 b . . . 428
10.122.2.3 g . . . 428
10.122.2.4 r . . . 428

10.123 LGL::TextureViewDescriptor Struct Reference . . . 428
10.123.1 Detailed Description . . . 429
10.123.2 Member Data Documentation . . . 429
10.123.2.1 format . . . 429
10.123.2.2 subresource . . . 429
10.123.2.3 swizzle . . . 430
10.123.2.4 type . . . 430
10.124 LGL::UninitializeTag Struct Reference . . . 430
10.124.1 Detailed Description . . . 431
10.125 LGL::UTF8String Class Reference . . . 431
10.125.1 Detailed Description . . . 433
10.125.2 Member Typedef Documentation . . . 433
10.125.2.1 const_iterator . . . 433
10.125.2.2 const_pointer . . . 433
10.125.2.3 const_reference . . . 433
10.125.2.4 const_reverse_iterator . . . 433
10.125.2.5 difference_type . . . 433
10.125.2.6 iterator . . . 433
10.125.2.7 pointer . . . 433
10.125.2.8 reference . . . 433
10.125.2.9 reverse_iterator . . . 433
10.125.2.10 size_type . . . 433
10.125.2.11 value_type . . . 433
10.125.3 Constructor & Destructor Documentation . . . 433
10.125.3.1 UTF8String() . . . 433
10.125.3.2 UTF8String(const UTF8String &rhs) . . . 433
10.125.3.3 UTF8String(UTF8String &&rhs) . . . 433
10.125.3.4 UTF8String(const StringView &str) . . . 434
10.125.3.5 UTF8String(const WStringView &str) . . . 434
10.125.3.6 UTF8String(const char *str) . . . 434

10.125.3.7 UTF8String(const wchar_t *str) . . . 434
10.125.3.8 UTF8String(const TString< TChar, Traits, Allocator > &str) . . . 434
10.125.4 Member Function Documentation . . . 434
10.125.4.1 at(size_type pos) const . . . 434
10.125.4.2 back() const . . . 434
10.125.4.3 begin() const . . . 434
10.125.4.4 c_str() const . . . 434
10.125.4.5 capacity() const . . . 434
10.125.4.6 cbegin() const . . . 434
10.125.4.7 cend() const . . . 434
10.125.4.8 clear() . . . 434
10.125.4.9 compare(const StringView &str) const . . . 434
10.125.4.10 compare(size_type pos1, size_type count1, const StringView &str) const . . . 434
10.125.4.11 compare(size_type pos1, size_type count1, const StringView &str, size_type pos2, size_type count2=npos) const . . . 435
10.125.4.12 compare(const WStringView &str) const . . . 435
10.125.4.13 compare(size_type pos1, size_type count1, const WStringView &str) const . . . 435
10.125.4.14 compare(size_type pos1, size_type count1, const WStringView &str, size_type pos2, size_type count2=npos) const . . . 435
10.125.4.15 crbegin() const . . . 435
10.125.4.16 crend() const . . . 435
10.125.4.17 data() const . . . 435
10.125.4.18 empty() const . . . 435
10.125.4.19 end() const . . . 435
10.125.4.20 front() const . . . 435
10.125.4.21 length() const . . . 435
10.125.4.22 operator const_pointer() const . . . 435
10.125.4.23 operator StringView() const . . . 435
10.125.4.24 operator+=(const UTF8String &rhs) . . . 435
10.125.4.25 operator+=(const StringView &rhs) . . . 435
10.125.4.26 operator+=(const WStringView &rhs) . . . 435

10.125.4.27 operator+=(const char *rhs) . . . 435
10.125.4.28 operator+=(const wchar_t *rhs) . . . 435
10.125.4.29 operator+=(char chr) . . . 435
10.125.4.30 operator+=(wchar_t chr) . . . 435
10.125.4.31 operator=(const UTF8String &rhs) . . . 436
10.125.4.32 operator=(UTF8String &&rhs) . . . 436
10.125.4.33 operator[](size_type pos) const . . . 436
10.125.4.34 begin() const . . . 436
10.125.4.35 end() const . . . 436
10.125.4.36 size() const . . . 436
10.125.4.37 ToWCharArray() const . . . 436
10.125.5 Member Data Documentation . . . 436
10.125.5.1 npos . . . 436
10.126 LGL::VertexAttribute Struct Reference . . . 436
10.126.1 Detailed Description . . . 437
10.126.2 Constructor & Destructor Documentation . . . 437
10.126.2.1 VertexAttribute()=default . . . 437
10.126.2.2 VertexAttribute(const VertexAttribute &)=default . . . 437
10.126.2.3 VertexAttribute(const StringView &name, const Format format, std::uint32_t location=0, std::uint32_t instanceDivisor=0, const SystemValue systemValue=↩ SystemValue::Undefined) . . . 437
10.126.2.4 VertexAttribute(const StringView &semanticName, std::uint32_t semanticIndex, const Format format, std::uint32_t location=0, std::uint32_t instanceDivisor=0) . . . 438
10.126.2.5 VertexAttribute(const StringView &name, const Format format, std::uint32_t location, std::uint32_t offset, std::uint32_t stride, std::uint32_t slot=0, std::uint32_t instanceDivisor=0) . . . 438
10.126.2.6 VertexAttribute(const StringView &semanticName, std::uint32_t semanticIndex, const Format format, std::uint32_t location, std::uint32_t offset, std::uint32_↩ t stride, std::uint32_t slot=0, std::uint32_t instanceDivisor=0) . . . 438
10.126.3 Member Function Documentation . . . 438
10.126.3.1 GetSize() const . . . 438
10.126.3.2 operator=(const VertexAttribute &)=default . . . 438
10.126.4 Member Data Documentation . . . 438

10.126.4.1 format . . . 438
10.126.4.2 instanceDivisor . . . 439
10.126.4.3 location . . . 439
10.126.4.4 name . . . 439
10.126.4.5 offset . . . 439
10.126.4.6 semanticIndex . . . 440
10.126.4.7 slot . . . 440
10.126.4.8 stride . . . 440
10.126.4.9 systemValue . . . 440
10.127 LGL::VertexFormat Struct Reference . . . 441
10.127.1 Detailed Description . . . 441
10.127.2 Member Function Documentation . . . 441
10.127.2.1 AppendAttribute(const VertexAttribute &attrib) . . . 441
10.127.2.2 GetStride() const . . . 442
10.127.2.3 GetStride(std::uint32_t slot) const . . . 442
10.127.2.4 SetSlot(std::uint32_t slot) . . . 442
10.127.2.5 SetStride(std::uint32_t stride) . . . 443
10.127.2.6 SetStride(std::uint32_t stride, std::uint32_t slot) . . . 443
10.127.3 Member Data Documentation . . . 443
10.127.3.1 attributes . . . 443
10.128 LGL::VertexShaderAttributes Struct Reference . . . 443
10.128.1 Detailed Description . . . 444
10.128.2 Member Data Documentation . . . 444
10.128.2.1 inputAttribs . . . 444
10.128.2.2 outputAttribs . . . 444
10.129 LGL::VideoAdapterDescriptor Struct Reference . . . 444
10.129.1 Detailed Description . . . 445
10.129.2 Member Data Documentation . . . 445
10.129.2.1 name . . . 445
10.129.2.2 outputs . . . 445

10.129.2.3 vendor . . . 445
10.129.2.4 videoMemory . . . 445
10.130 LLGL::VideoOutputDescriptor Struct Reference . . . 446
10.130.1 Detailed Description . . . 446
10.130.2 Member Data Documentation . . . 446
10.130.2.1 displayModes . . . 446
10.131 LLGL::Viewport Struct Reference . . . 446
10.131.1 Detailed Description . . . 447
10.131.2 Constructor & Destructor Documentation . . . 447
10.131.2.1 Viewport()=default . . . 447
10.131.2.2 Viewport(const Viewport &)=default . . . 447
10.131.2.3 Viewport(float x, float y, float width, float height) . . . 447
10.131.2.4 Viewport(float x, float y, float width, float height, float minDepth, float maxDepth) . 447
10.131.2.5 Viewport(const Extent2D &extent) . . . 447
10.131.2.6 Viewport(const Extent2D &extent, float minDepth, float maxDepth) . . . 448
10.131.2.7 Viewport(const Offset2D &offset, const Extent2D &extent) . . . 448
10.131.2.8 Viewport(const Offset2D &offset, const Extent2D &extent, float minDepth, float maxDepth) . . . 448
10.131.3 Member Data Documentation . . . 448
10.131.3.1 height . . . 448
10.131.3.2 maxDepth . . . 448
10.131.3.3 minDepth . . . 448
10.131.3.4 width . . . 449
10.131.3.5 x . . . 449
10.131.3.6 y . . . 449
10.132 LLGL::Window Class Reference . . . 449
10.132.1 Detailed Description . . . 451
10.132.2 Constructor & Destructor Documentation . . . 451
10.132.2.1 ~Window() . . . 451
10.132.2.2 Window() . . . 451
10.132.3 Member Function Documentation . . . 452

10.132.3.1 AdaptForVideoMode(Extent2D *resolution, bool *fullscreen) override 452
10.132.3.2 AddEventListener(const std::shared_ptr< EventListener > &eventListener) . . . 452
10.132.3.3 Create(const WindowDescriptor &desc) 452
10.132.3.4 FindResidentDisplay() const override final 452
10.132.3.5 GetBehavior() const 453
10.132.3.6 GetDesc() const =0 453
10.132.3.7 GetPosition() const =0 453
10.132.3.8 GetSize(bool useClientArea=true) const =0 453
10.132.3.9 GetTitle() const =0 453
10.132.3.10 HasFocus() const 453
10.132.3.11 HasQuit() const 453
10.132.3.12 IsShown() const =0 453
10.132.3.13 OnProcessEvents()=0 453
10.132.3.14 PostChar(wchar_t chr) 454
10.132.3.15 PostDoubleClick(Key keyCode) 454
10.132.3.16 PostGetFocus() 454
10.132.3.17 PostGlobalMotion(const Offset2D &motion) 454
10.132.3.18 PostKeyDown(Key keyCode) 454
10.132.3.19 PostKeyUp(Key keyCode) 454
10.132.3.20 PostLocalMotion(const Offset2D &position) 454
10.132.3.21 PostLostFocus() 455
10.132.3.22 PostQuit() 455
10.132.3.23 PostResize(const Extent2D &clientAreaSize) 455
10.132.3.24 PostTimer(std::uint32_t timerID) 455
10.132.3.25 PostWheelMotion(int motion) 455
10.132.3.26 ProcessEvents() override final 456
10.132.3.27 RemoveEventListener(const EventListener *eventListener) 456
10.132.3.28 SetBehavior(const WindowBehavior &behavior) 456
10.132.3.29 SetDesc(const WindowDescriptor &desc)=0 456
10.132.3.30 SetPosition(const Offset2D &position)=0 456

10.132.3.31 SetSize(const Extent2D &size, bool useClientArea=true)=0 456

10.132.3.32 SetTitle(const UTF8String &title)=0 456

10.132.3.33 Show(bool show=true)=0 457

10.133 LGL::WindowBehavior Struct Reference 457

10.133.1 Detailed Description 457

10.133.2 Member Data Documentation 457

10.133.2.1 disableClearOnResize 457

10.133.2.2 moveAndResizeTimerID 458

10.134 LGL::WindowDescriptor Struct Reference 458

10.134.1 Detailed Description 459

10.134.2 Member Data Documentation 459

10.134.2.1 acceptDropFiles 459

10.134.2.2 borderless 459

10.134.2.3 centered 459

10.134.2.4 position 459

10.134.2.5 resizable 459

10.134.2.6 size 460

10.134.2.7 title 460

10.134.2.8 visible 460

10.134.2.9 windowContext 460

11 File Documentation 461

11.1 AndroidNativeHandle.h File Reference . . . 461

11.2 ArrayView.h File Reference . . . 461

11.3 Blob.h File Reference . . . 462

11.4 Buffer.h File Reference . . . 462

11.5 BufferArray.h File Reference . . . 462

11.6 BufferFlags.h File Reference . . . 463

11.7 Canvas.h File Reference . . . 463

11.8 CanvasFlags.h File Reference . . . 464

11.9 Color.h File Reference . . . 464

11.10 ColorRGB.h File Reference . . . 465

11.11 ColorRGBA.h File Reference . . . 466

11.12 CommandBuffer.h File Reference . . . 467

11.13 CommandBufferFlags.h File Reference . . . 467

11.14 CommandQueue.h File Reference . . . 468

11.15 Constants.h File Reference . . . 468

11.16 Display.h File Reference . . . 469

11.17 DisplayFlags.h File Reference . . . 469

11.18 Export.h File Reference . . . 470

11.18.1 Macro Definition Documentation . . . 470

11.18.1.1 LLGL_EXPORT . . . 470

11.19 Fence.h File Reference . . . 470

11.20 Format.h File Reference . . . 470

11.21 ForRange.h File Reference . . . 473

11.21.1 Macro Definition Documentation . . . 473

11.21.1.1 for_range . . . 473

11.21.1.2 for_range_end . . . 473

11.21.1.3 for_range_reverse . . . 473

11.21.1.4 for_range_reverse_iter . . . 473

11.21.1.5 for_subrange . . . 473

11.21.1.6 for_subrange_reverse . . . 473
11.22 ForwardDecls.h File Reference . . . 474
11.23 FragmentAttribute.h File Reference . . . 474
11.24 Image.h File Reference . . . 474
11.25 ImageFlags.h File Reference . . . 475
11.26 IndirectArguments.h File Reference . . . 476
11.27 Input.h File Reference . . . 476
11.28 Interface.h File Reference . . . 476
11.28.1 Macro Definition Documentation . . . 477
11.28.1.1 LLGL_DECLARE_INTERFACE . . . 477
11.28.1.2 LLGL_IMPLEMENT_INTERFACE . . . 477
11.29 InterfaceID.h File Reference . . . 477
11.30 IOSNativeHandle.h File Reference . . . 478
11.31 Key.h File Reference . . . 478
11.32 LinuxNativeHandle.h File Reference . . . 479
11.33 LLGL.h File Reference . . . 479
11.34 Log.h File Reference . . . 480
11.35 MacOSNativeHandle.h File Reference . . . 481
11.36 NativeHandle.h File Reference . . . 481
11.37 NonCopyable.h File Reference . . . 481
11.38 PipelineLayout.h File Reference . . . 481
11.39 PipelineLayoutFlags.h File Reference . . . 482
11.40 PipelineState.h File Reference . . . 482
11.41 PipelineStateFlags.h File Reference . . . 483
11.42 Platform.h File Reference . . . 485
11.42.1 Macro Definition Documentation . . . 485
11.42.1.1 LLGL_API . . . 485
11.42.1.2 LLGL_API_CDECL . . . 485
11.42.1.3 LLGL_API_STDCALL . . . 485
11.42.1.4 LLGL_API_THISCALL . . . 485

11.43 QueryHeap.h File Reference . . . 485
11.44 QueryHeapFlags.h File Reference . . . 486
11.45 RendererConfiguration.h File Reference . . . 486
11.46 RenderingDebugger.h File Reference . . . 487
11.47 RenderingProfiler.h File Reference . . . 488
11.48 RenderPass.h File Reference . . . 488
11.49 RenderPassFlags.h File Reference . . . 488
11.50 RenderSystem.h File Reference . . . 489
11.51 RenderSystemChild.h File Reference . . . 491
11.52 RenderSystemFlags.h File Reference . . . 491
11.53 RenderTarget.h File Reference . . . 493
11.54 RenderTargetFlags.h File Reference . . . 493
11.55 Report.h File Reference . . . 493
11.56 Resource.h File Reference . . . 494
11.57 ResourceFlags.h File Reference . . . 494
11.58 ResourceHeap.h File Reference . . . 495
11.59 ResourceHeapFlags.h File Reference . . . 495
11.60 Sampler.h File Reference . . . 495
11.61 SamplerFlags.h File Reference . . . 496
11.62 Shader.h File Reference . . . 496
11.63 ShaderFlags.h File Reference . . . 497
11.64 ShaderReflection.h File Reference . . . 498
11.65 SmallVector.h File Reference . . . 499
11.66 StaticLimits.h File Reference . . . 500
11.66.1 Macro Definition Documentation . . . 500
11.66.1.1 LLGL_MAX_NUM_ATTACHMENTS . . . 500
11.66.1.2 LLGL_MAX_NUM_COLOR_ATTACHMENTS . . . 500
11.66.1.3 LLGL_MAX_NUM_SAMPLES . . . 500
11.66.1.4 LLGL_MAX_NUM_SO_BUFFERS . . . 500
11.66.1.5 LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS . . . 500

11.67 Strings.h File Reference 501
11.68 StringView.h File Reference 502
11.69 Surface.h File Reference 502
11.70 SwapChain.h File Reference 503
11.71 SwapChainFlags.h File Reference 503
11.72 SystemValue.h File Reference 503
11.73 Tags.h File Reference 504
11.74 Texture.h File Reference 504
11.75 TextureFlags.h File Reference 505
11.76 Timer.h File Reference 506
11.77 TypeInfo.h File Reference 507
11.78 TypeNames.h File Reference 507
11.79 Types.h File Reference 508
11.80 UTF8String.h File Reference 509
11.81 Utility.h File Reference 510
11.82 Version.h File Reference 511
11.83 VertexAttribute.h File Reference 512
11.84 VertexFormat.h File Reference 512
11.85 VideoAdapter.h File Reference 513
11.86 Win32NativeHandle.h File Reference 513
11.87 Window.h File Reference 513
11.88 WindowFlags.h File Reference 514

Index **515**

Chapter 1

LLGL 0.03 Beta Documentation

LLGL (Low Level Graphics Library)

Overview

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

Progress

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

Getting Started

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

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

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

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

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

    //TO BE CONTINUED ...

    // Main loop
    while (window->ProcessEvents() && !input->KeyPressed(LLGL::Key::Escape))
    {

        // Draw with OpenGL, or Direct3D, or Vulkan, or whatever ...

    }

    return 0;
}
```

Thin Abstraction Layer

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

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

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

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

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

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

Chapter 2

Todo List

Member LLGL::AttachmentType

Remove this enum and use Format instead.

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

Incomplete for: D3D12, Vulkan, Metal.

Member LLGL::DataType

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

Member LLGL::ESProfile

Not supported yet

Member LLGL::FrameProfile::bufferReads

Not available yet.

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

Not implemented yet.

Member LLGL::Image::MirrorXYPlane ()

Not implemented yet

Member LLGL::Image::MirrorXZPlane ()

Not implemented yet

Member LLGL::Image::MirrorYZPlane ()

Not implemented yet

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

Not implemented yet.

Member LLGL::ImageFormat

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

Class LLGL::Input

Make Window::EventListener a member of Input instead of extending it. Also add a member of Canvas::↩ EventListener and add touch event functions.

Class LLGL::MetalDependentStateDescriptor

Move into namespace Metal

Member LLGL::MiscFlags::DynamicUsage

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

Member LLGL::PrimitiveTopology

For Direct3D 12, the restart primitive index value (aka. strip cut value) can only be `0xFFFFFFFF` at the moment as this needs to be specified at PSO creation time.

Class LLGL::RendererConfigurationOpenGLES3

Replace with RendererConfigurationOpenGL and make use of OpenGLContextProfile::ESProfile.

Member LLGL::RendererConfigurationVulkan::minDeviceMemoryAllocationSize

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

Member LLGL::RendererConfigurationVulkan::reduceDeviceMemoryFragmentation

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

Class LLGL::RenderingDebugger::Message

Rename to "Report"

Member LLGL::RenderingFeatures::hasSamplers

Rename to `hasNativeSamplerStates` or remove entirely.

Class LLGL::RenderingProfiler

Refactor this for the new ResourceHeap and RenderPass interfaces.

Member LLGL::RenderTargetDescriptor::customMultiSampling

Remove this attribute and support custom resolve attachments instead.

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

(Maybe) replace by IsInstanceOf

Member LLGL::SamplerDescriptor::mipMapping

Rename to `mipMapEnabled`.

Member LLGL::Texture::GetMemoryFootprint () const

Maybe make this virtual to allow memory alignment (especially for D3D12).

Member LLGL::Texture::GetMemoryFootprint (const Extent3D &extent, const TextureSubresource &subresource) const

Maybe make this virtual to allow memory alignment (especially for D3D12).

Class LLGL::VideoAdapterDescriptor

Currently unused in the interface.

Class LLGL::VideoOutputDescriptor

Currently unused in the interface.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Global functions for Strict-Weak-Order (SWO) comparisons. 21
Global operators for basic data structures. 22
Global type aliases to callback interfaces. 25
Buffer utility functions to determine buffer types. 27
Hardware format utility functions. 28
Data type utility functions. 32
Image utility functions to classify and convert image data. 33
Buffer argument structures for indirect draw and compute commands. 38
Global type-to-string conversion functions. 39
Global utility functions, especially to fill descriptor structures. 41
Texture utility functions to determine texture dimension and buffer sizes. 47

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

LLGL 55

LLGL::Constants

Namespace with all constants used as default arguments 111

LLGL::Log 113

LLGL::Timer 116

LLGL::Version

Namespace with functions to determine LLGL version 117

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

LLGL::ApplicationDescriptor . . . 119
LLGL::ArrayView< T > . . . 120
LLGL::ArrayView< const char * > . . . 120
LLGL::ArrayView< LLGL::VertexAttribute > . . . 120
LLGL::AttachmentClear . . . 124
LLGL::AttachmentDescriptor . . . 126
LLGL::AttachmentFormatDescriptor . . . 128
LLGL::BarrierFlags . . . 130
LLGL::BasicStringView< TChar, Traits > . . . 131
LLGL::BindFlags . . . 135
LLGL::BindingDescriptor . . . 138
LLGL::BlendDescriptor . . . 140
LLGL::BlendTargetDescriptor . . . 144
LLGL::BufferDescriptor . . . 151
LLGL::BufferViewDescriptor . . . 154
LLGL::CanvasDescriptor . . . 160
LLGL::ClearFlags . . . 161
LLGL::ClearValue . . . 161
LLGL::Color< T, N > . . . 163
LLGL::Color< bool > . . . 163
LLGL::Color< float > . . . 163
LLGL::Color< T, 3u > . . . 167
LLGL::Color< T, 4u > . . . 172
LLGL::CommandBufferDescriptor . . . 204
LLGL::CommandBufferFlags . . . 206
LLGL::ComputePipelineDescriptor . . . 211
LLGL::ComputeShaderAttributes . . . 212
LLGL::CPUAccessFlags . . . 212
LLGL::DepthBiasDescriptor . . . 213
LLGL::DepthDescriptor . . . 214
LLGL::DispatchIndirectArguments . . . 215
LLGL::DisplayModeDescriptor . . . 222
LLGL::DrawIndexedIndirectArguments . . . 223
LLGL::DrawIndirectArguments . . . 224
LLGL::DrawPatchIndirectArguments . . . 226

LLGL::DstImageDescriptor . . . 227
LLGL::Extent2D . . . 234
LLGL::Extent3D . . . 235
LLGL::FormatAttributes . . . 237
LLGL::FormatFlags . . . 239
LLGL::FragmentAttribute . . . 241
LLGL::FragmentShaderAttributes . . . 244
LLGL::FrameProfile . . . 244
LLGL::GraphicsPipelineDescriptor . . . 253
LLGL::GrowStrategyAddHalf . . . 257
LLGL::GrowStrategyDouble . . . 257
LLGL::GrowStrategyRoundUpPow2 . . . 258
LLGL::Image . . . 259
LLGL::InterfaceID . . . 273
LLGL::RenderingDebugger::Message . . . 276
LLGL::MetalDependentStateDescriptor . . . 278
LLGL::MiscFlags . . . 279
LLGL::NativeContextHandle . . . 281
LLGL::NativeHandle . . . 282
LLGL::NonCopyable . . . 284
- LLGL::Blob . . . 145
- LLGL::Interface . . . 272
  - LLGL::Canvas::EventListener . . . 233
  - LLGL::Display . . . 216
  - LLGL::Input . . . 268
  - LLGL::RenderSystem . . . 327
  - LLGL::RenderSystemChild . . . 346
    - LLGL::BufferArray . . . 150
    - LLGL::CommandBuffer . . . 177
    - LLGL::CommandQueue . . . 208
    - LLGL::Fence . . . 237
    - LLGL::PipelineLayout . . . 287
    - LLGL::PipelineState . . . 289
    - LLGL::QueryHeap . . . 291
    - LLGL::RenderPass . . . 325
    - LLGL::RenderTarget . . . 351
      - LLGL::SwapChain . . . 404
    - LLGL::Resource . . . 359
      - LLGL::Buffer . . . 148
      - LLGL::Sampler . . . 366
      - LLGL::Texture . . . 414
    - LLGL::ResourceHeap . . . 360
    - LLGL::Shader . . . 371
  - LLGL::Report . . . 357
  - LLGL::Surface . . . 400
    - LLGL::Canvas . . . 156
    - LLGL::Window . . . 449
  - LLGL::Window::EventListener . . . 229

LLGL::Offset2D . . . 285
LLGL::Offset3D . . . 286
LLGL::PipelineLayoutDescriptor . . . 288
LLGL::ProfileTimeRecord . . . 290
LLGL::QueryHeapDescriptor . . . 292
LLGL::QueryPipelineStatistics . . . 293
LLGL::RasterizerDescriptor . . . 296
LLGL::RendererConfigurationOpenGL . . . 298
LLGL::RendererConfigurationOpenGLES3 . . . 300
LLGL::RendererConfigurationVulkan . . . 300

LLGL::RendererID . . . 302
LLGL::RendererInfo . . . 304
LLGL::RenderingCapabilities . . . 305
LLGL::RenderingDebugger . . . 307
LLGL::RenderingFeatures . . . 310
LLGL::RenderingLimits . . . 318
LLGL::RenderingProfiler . . . 323
LLGL::RenderPassDescriptor . . . 326
LLGL::RenderSystemDeleter . . . 347
LLGL::RenderSystemDescriptor . . . 349
LLGL::RenderTargetDescriptor . . . 355
LLGL::ResizeBuffersFlags . . . 358
LLGL::ResourceHeapDescriptor . . . 361
LLGL::ResourceViewDescriptor . . . 363
LLGL::SamplerDescriptor . . . 367
LLGL::Scissor . . . 370
LLGL::ShaderCompileFlags . . . 374
LLGL::ShaderDescriptor . . . 377
LLGL::ShaderMacro . . . 381
LLGL::ShaderReflection . . . 383
LLGL::ShaderResource . . . 384
LLGL::ShaderUniform . . . 385
LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy > . . . 386
LLGL::SmallVector< char > . . . 386
LLGL::SrcImageDescriptor . . . 393
LLGL::StageFlags . . . 395
LLGL::StencilDescriptor . . . 396
LLGL::StencilFaceDescriptor . . . 397
LLGL::SwapChainDescriptor . . . 410
LLGL::TessellationDescriptor . . . 412
LLGL::TextureDescriptor . . . 417
LLGL::TextureLocation . . . 421
LLGL::TextureRegion . . . 423
LLGL::TextureSubresource . . . 425
LLGL::TextureSwizzleRGBA . . . 427
LLGL::TextureViewDescriptor . . . 428
LLGL::UninitializeTag . . . 430
LLGL::UTF8String . . . 431
LLGL::VertexAttribute . . . 436
LLGL::VertexFormat . . . 441
LLGL::VertexShaderAttributes . . . 443
LLGL::VideoAdapterDescriptor . . . 444
LLGL::VideoOutputDescriptor . . . 446
LLGL::Viewport . . . 446
LLGL::WindowBehavior . . . 457
LLGL::WindowDescriptor . . . 458

Chapter 6

Class Index

6.1 Class List

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

LLGL::ApplicationDescriptor
Application descriptor structure 119
LLGL::ArrayView< T >
Constant array view container template. Keeps pointer to array and length 120
LLGL::AttachmentClear
Attachment clear command structure 124
LLGL::AttachmentDescriptor
Render target attachment descriptor structure 126
LLGL::AttachmentFormatDescriptor
Render pass attachment descriptor structure 128
LLGL::BarrierFlags
Flags for memory barriers in resource heaps 130
LLGL::BasicStringView< TChar, Traits >
Constant string view container template. Keeps pointer to string and its length 131
LLGL::BindFlags
Flags for Buffer and Texture resources that describe for which purposes they will be used . . . 135
LLGL::BindingDescriptor
Layout structure for a single binding point of the pipeline layout descriptor 138
LLGL::BlendDescriptor
Blending state descriptor structure 140
LLGL::BlendTargetDescriptor
Blend target state descriptor structure 144
LLGL::Blob
CPU read-only buffer of arbitrary size 145
LLGL::Buffer
Hardware buffer interface 148
LLGL::BufferArray
Hardware buffer container interface 150
LLGL::BufferDescriptor
Hardware buffer descriptor structure 151
LLGL::BufferViewDescriptor
Buffer view descriptor structure 154
LLGL::Canvas
Canvas interface for mobile platforms 156
LLGL::CanvasDescriptor
Canvas descriptor structure 160

LLGL::ClearFlags
Command buffer clear flags . . . 161
LLGL::ClearValue
Clear value structure for color, depth, and stencil clear operations . . . 161
LLGL::Color< T, N >
Base color class with N components . . . 163
LLGL::Color< T, 3u >
RGB color class with components: r, g, and b . . . 167
LLGL::Color< T, 4u >
RGBA color class with components: r, g, b, and a . . . 172
LLGL::CommandBuffer
Command buffer interface used for storing and encoding GPU commands . . . 177
LLGL::CommandBufferDescriptor
Command buffer descriptor structure . . . 204
LLGL::CommandBufferFlags
Command buffer creation flags . . . 206
LLGL::CommandQueue
Command queue interface . . . 208
LLGL::ComputePipelineDescriptor
Compute pipeline state descriptor structure . . . 211
LLGL::ComputeShaderAttributes
Compute shader specific descriptor structure . . . 212
LLGL::CPUAccessFlags
CPU read/write access flag enumeration for mapping GPU into CPU memory space . . . 212
LLGL::DepthBiasDescriptor
Depth bias descriptor structure to control fragment depth values . . . 213
LLGL::DepthDescriptor
Depth state descriptor structure . . . 214
LLGL::DispatchIndirectArguments
Format structure for the arguments of an indirect compute command . . . 215
LLGL::Display
Display interface to query the attributes of all connected displays/monitors . . . 216
LLGL::DisplayModeDescriptor
Display mode descriptor structure . . . 222
LLGL::DrawIndexedIndirectArguments
Format structure for the arguments of an indirect indexed draw command . . . 223
LLGL::DrawIndirectArguments
Format structure for the arguments of an indirect draw command . . . 224
LLGL::DrawPatchIndirectArguments
Format structure for the arguments of an indirect draw/indexed-draw command for tessellation on Metal renderer . . . 226
LLGL::DstImageDescriptor
Descriptor structure for an image that is used as destination for writing the image data . . . 227
LLGL::Window::EventListener
Interface for all window event listeners . . . 229
LLGL::Canvas::EventListener
Interface for all canvas event listeners . . . 233
LLGL::Extent2D
2-Dimensional extent structure . . . 234
LLGL::Extent3D
3-Dimensional extent structure . . . 235
LLGL::Fence
Fence interface for CPU/GPU synchronization . . . 237
LLGL::FormatAttributes
Vertex and texture format attributes structure . . . 237
LLGL::FormatFlags
Format attribute flags enumeration . . . 239

LLGL::FragmentAttribute
Fragment output attribute structure . . . 241
LLGL::FragmentShaderAttributes
Fragment shader specific descriptor structure . . . 244
LLGL::FrameProfile
Profile of a rendered frame . . . 244
LLGL::GraphicsPipelineDescriptor
Graphics pipeline state descriptor structure . . . 253
LLGL::GrowStrategyAddHalf
Container capacity grow strategy to increase the capacity to 150% . . . 257
LLGL::GrowStrategyDouble
Container capacity grow strategy to increase the capacity to 200% . . . 257
LLGL::GrowStrategyRoundUpPow2
Container capacity grow strategy to increase the capacity to the next higher power of two . . . 258
LLGL::Image
Utility class to manage the storage and attributes of an image . . . 259
LLGL::Input
Default window event listener to receive user input . . . 268
LLGL::Interface
Base class for all interfaces in LLGL . . . 272
LLGL::InterfaceID
Enumeration of all interfaces in LLGL . . . 273
LLGL::RenderingDebugger::Message
Rendering debugger message class . . . 276
LLGL::MetalDependentStateDescriptor
Graphics API dependent state descriptor for the Metal renderer . . . 278
LLGL::MiscFlags
Miscellaneous resource flags . . . 279
LLGL::NativeContextHandle
Android native context handle structure . . . 281
LLGL::NativeHandle
Android native handle structure . . . 282
LLGL::NonCopyable
Base class for all classes with virtual function tables in LLGL . . . 284
LLGL::Offset2D
2-Dimensional offset structure . . . 285
LLGL::Offset3D
3-Dimensional offset structure . . . 286
LLGL::PipelineLayout
Pipeline layout interface . . . 287
LLGL::PipelineLayoutDescriptor
Pipeline layout descritpor structure . . . 288
LLGL::PipelineState
Graphics and compute pipeline state interface . . . 289
LLGL::ProfileTimeRecord
Structure with annotation and elapsed time for a timer profile . . . 290
LLGL::QueryHeap
Query heap interface that holds a certain number of queries that are all of the same type . . . 291
LLGL::QueryHeapDescriptor
Query heap descriptor structure . . . 292
LLGL::QueryPipelineStatistics
Query data structure for pipeline statistics . . . 293
LLGL::RasterizerDescriptor
Rasterizer state descriptor structure . . . 296
LLGL::RendererConfigurationOpenGL
OpenGL profile descriptor structure . . . 298
LLGL::RendererConfigurationOpenGLES3
OpenGL ES 3 profile descriptor structure . . . 300

LLGL::RendererConfigurationVulkan
Structure for a Vulkan renderer specific configuration 300
LLGL::RendererID
Renderer identification number enumeration 302
LLGL::RendererInfo
Renderer basic information structure 304
LLGL::RenderingCapabilities
Structure with all attributes describing the rendering capabilities of the render system 305
LLGL::RenderingDebugger
Rendering debugger interface 307
LLGL::RenderingFeatures
Contains the attributes for all supported rendering features 310
LLGL::RenderingLimits
Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc 318
LLGL::RenderingProfiler
Rendering profiler model class 323
LLGL::RenderPass
Render pass interface 325
LLGL::RenderPassDescriptor
Render pass descriptor structure 326
LLGL::RenderSystem
Render system interface 327
LLGL::RenderSystemChild
Base class for all interfaces whoes instances are owned by the RenderSystem 346
LLGL::RenderSystemDeleter
Delegate to delete an instance of the RenderSystem interface 347
LLGL::RenderSystemDescriptor
Render system descriptor structure 349
LLGL::RenderTarget
Render target interface 351
LLGL::RenderTargetDescriptor
Render target descriptor structure 355
LLGL::Report
Error and warning report interface 357
LLGL::ResizeBuffersFlags
Swap-chain resize buffers flags 358
LLGL::Resource
Base class for all hardware resource interfaces 359
LLGL::ResourceHeap
Resource heap interface 360
LLGL::ResourceHeapDescriptor
Resource heap descriptor structure 361
LLGL::ResourceViewDescriptor
Resource view descriptor structure 363
LLGL::Sampler
Sampler interface 366
LLGL::SamplerDescriptor
Texture sampler descriptor structure 367
LLGL::Scissor
Scissor dimensions 370
LLGL::Shader
Shader interface 371
LLGL::ShaderCompileFlags
Shader compilation flags enumeration 374
LLGL::ShaderDescriptor
Shader source and binary code descriptor structure 377
LLGL::ShaderMacro
Shader macro structure with name and optional body definition 381

LLGL::ShaderReflection
Shader reflection structure . . . 383
LLGL::ShaderResource
Shader reflection resource structure . . . 384
LLGL::ShaderUniform
Shader reflection uniform structure . . . 385
LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >
Generic container class for consecutive arrays optimized for small sizes . . . 386
LLGL::SrcImageDescriptor
Descriptor structure for an image that is used as source for reading the image data . . . 393
LLGL::StageFlags
Shader stage flags enumeration . . . 395
LLGL::StencilDescriptor
Stencil state descriptor structure . . . 396
LLGL::StencilFaceDescriptor
Stencil face descriptor structure . . . 397
LLGL::Surface
The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on mobile platforms) . . . 400
LLGL::SwapChain
Swap-chain interface . . . 404
LLGL::SwapChainDescriptor
Swap chain descriptor structure . . . 410
LLGL::TessellationDescriptor
Tessellation descriptor structure for the graphics pipeline . . . 412
LLGL::Texture
Texture interface . . . 414
LLGL::TextureDescriptor
Texture descriptor structure . . . 417
LLGL::TextureLocation
Texture location structure: MIP-map level and offset . . . 421
LLGL::TextureRegion
Texture region structure: Subresource (MIP-map level and array layer range), offset, and extent 423
LLGL::TextureSubresource
Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource . . . 425
LLGL::TextureSwizzleRGBA
Texture component swizzle structure for red, green, blue, and alpha components . . . 427
LLGL::TextureViewDescriptor
Texture view descriptor structure . . . 428
LLGL::UninitializeTag
Common uninitialize tag . . . 430
LLGL::UTF8String
Container class for UTF-8 encoded strings . . . 431
LLGL::VertexAttribute
Vertex input/output attribute structure . . . 436
LLGL::VertexFormat
Utility structure to store vertex attributes . . . 441
LLGL::VertexShaderAttributes
Vertex (or geometry) shader specific structure . . . 443
LLGL::VideoAdapterDescriptor
Video adapter descriptor structure . . . 444
LLGL::VideoOutputDescriptor
Video output structure . . . 446
LLGL::Viewport
Viewport dimensions . . . 446
LLGL::Window
Window interface for desktop platforms . . . 449

LLGL::WindowBehavior
Window behavior structure 457
LLGL::WindowDescriptor
Window descriptor structure 458

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

AndroidNativeHandle.h . . . 461
ArrayView.h . . . 461
Blob.h . . . 462
Buffer.h . . . 462
BufferArray.h . . . 462
BufferFlags.h . . . 463
Canvas.h . . . 463
CanvasFlags.h . . . 464
Color.h . . . 464
ColorRGB.h . . . 465
ColorRGBA.h . . . 466
CommandBuffer.h . . . 467
CommandBufferFlags.h . . . 467
CommandQueue.h . . . 468
Constants.h . . . 468
Display.h . . . 469
DisplayFlags.h . . . 469
Export.h . . . 470
Fence.h . . . 470
Format.h . . . 470
ForRange.h . . . 473
ForwardDecls.h . . . 474
FragmentAttribute.h . . . 474
Image.h . . . 474
ImageFlags.h . . . 475
IndirectArguments.h . . . 476
Input.h . . . 476
Interface.h . . . 476
InterfaceID.h . . . 477
IOSNativeHandle.h . . . 478
Key.h . . . 478
LinuxNativeHandle.h . . . 479
LLGL.h . . . 479
Log.h . . . 480
MacOSNativeHandle.h . . . 481

NativeHandle.h . . . 481
NonCopyable.h . . . 481
PipelineLayout.h . . . 481
PipelineLayoutFlags.h . . . 482
PipelineState.h . . . 482
PipelineStateFlags.h . . . 483
Platform.h . . . 485
QueryHeap.h . . . 485
QueryHeapFlags.h . . . 486
RendererConfiguration.h . . . 486
RenderingDebugger.h . . . 487
RenderingProfiler.h . . . 488
RenderPass.h . . . 488
RenderPassFlags.h . . . 488
RenderSystem.h . . . 489
RenderSystemChild.h . . . 491
RenderSystemFlags.h . . . 491
RenderTarget.h . . . 493
RenderTargetFlags.h . . . 493
Report.h . . . 493
Resource.h . . . 494
ResourceFlags.h . . . 494
ResourceHeap.h . . . 495
ResourceHeapFlags.h . . . 495
Sampler.h . . . 495
SamplerFlags.h . . . 496
Shader.h . . . 496
ShaderFlags.h . . . 497
ShaderReflection.h . . . 498
SmallVector.h . . . 499
StaticLimits.h . . . 500
Strings.h . . . 501
StringView.h . . . 502
Surface.h . . . 502
SwapChain.h . . . 503
SwapChainFlags.h . . . 503
SystemValue.h . . . 503
Tags.h . . . 504
Texture.h . . . 504
TextureFlags.h . . . 505
Timer.h . . . 506
TypeInfo.h . . . 507
TypeNames.h . . . 507
Types.h . . . 508
UTF8String.h . . . 509
Utility.h . . . 510
Version.h . . . 511
VertexAttribute.h . . . 512
VertexFormat.h . . . 512
VideoAdapter.h . . . 513
Win32NativeHandle.h . . . 513
Window.h . . . 513
WindowFlags.h . . . 514

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. `UINT_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. `UINT_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 `[INT_MIN, INT_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 `[INT_MIN, INT_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 `[INT_MIN, INT_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 `[INT_MIN, INT_MAX]`.

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

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

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

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

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

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

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

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

8.3 Global type aliases to callback interfaces.

Typedefs

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

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

  Callback interface for the ValidateRenderingCaps function.

8.3.1 Detailed Description

8.3.2 Typedef Documentation

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

Debug callback function interface.

Parameters

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

Remarks

This output is renderer dependent.

Note

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

See also

RenderSystemDescriptor::debugCallback

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

Callback interface for the ValidateRenderingCaps function.

Parameters

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

Returns

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

See also

ValidateRenderingCaps

8.4 Buffer utility functions to determine buffer types.

Functions

- LLGL_EXPORT bool LLGL::IsTypedBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a typed buffer, i.e. `Buffer` or `RWBuffer` in HLSL.
- LLGL_EXPORT bool LLGL::IsStructuredBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a structured buffer, i.e. `StructuredBuffer`, `RWStructured↩Buffer`, `AppendStructuredBuffer`, or `ConsumeStructuredBuffer` in HLSL.
- LLGL_EXPORT bool LLGL::IsByteAddressBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a byte addresse buffer, i.e. `ByteAddressBuffer` or `RWByte↩AddressBuffer` in HLSL.

8.4.1 Detailed Description

8.4.2 Function Documentation

8.4.2.1 LLGL_EXPORT bool LLGL::IsByteAddressBuffer (const BufferDescriptor & *desc*)

Returns true if the buffer descriptor denotes a byte addresse buffer, i.e. `ByteAddressBuffer` or `RWByte↩AddressBuffer` in HLSL.

8.4.2.2 LLGL_EXPORT bool LLGL::IsStructuredBuffer (const BufferDescriptor & *desc*)

Returns true if the buffer descriptor denotes a structured buffer, i.e. `StructuredBuffer`, `RWStructured↩Buffer`, `AppendStructuredBuffer`, or `ConsumeStructuredBuffer` in HLSL.

8.4.2.3 LLGL_EXPORT bool LLGL::IsTypedBuffer (const BufferDescriptor & *desc*)

Returns true if the buffer descriptor denotes a typed buffer, i.e. `Buffer` or `RWBuffer` in HLSL.

8.5 Hardware format utility functions.

Functions

- LLGL_EXPORT const FormatAttributes & LLGL::GetFormatAttribs (const Format format)

  Returns the attributes for the specified hardware format or a zero-initialized structure in case of an invalid argument.
- 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::GetMemoryFootprint (const Format format, std::uint32_t count)

  Returns the memory footprint (in bytes) of a texture or buffer with the specified hardware format and number of elements.
- LLGL_EXPORT std::uint32_t LLGL::GetMemoryFootprint (const ImageFormat imageFormat, const DataType dataType, std::uint32_t count)

  Returns the memory footprint (in bytes) of an image with the specified format, data type, and number of elements.
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const Format format)

  Returns true if the specified hardware format is a compressed format, e.g. Format::BC1UNorm, Format::BC2UNorm_sRGB, Format::BC4SNorm, etc.
- 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 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::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::IsDepthFormat (const Format format)

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

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

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

  Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::RGBA8UNorm, Format::R8SInt etc.).
- LLGL_EXPORT bool LLGL::IsFloatFormat (const Format format)

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

8.5.1 Detailed Description

8.5.2 Function Documentation

8.5.2.1 LLGL_EXPORT const FormatAttributes& LLGL::GetFormatAttribs (const Format *format*)

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

See also

FormatAttributes
Format

8.5.2.2 LLGL_EXPORT std::uint32_t LLGL::GetMemoryFootprint (const Format *format*, std::uint32_t *count*)

Returns the memory footprint (in bytes) of a texture or buffer with the specified hardware format and number of elements.

Parameters

in	*format*	Specifies the hardware format.
in	*count*	Specifies the number of elements that make up the memory footprint. For the DXT compressed formats, this must be a multiple of 16, since these formats compress images in 4x4 texel blocks.

Returns

The memory footprint size (in bytes), or zero if the input is invalid.

See also

GetMemoryFootprint(const TextureType, const Format, const Extent3D&, const TextureSubresource&)
Texture::GetMemoryFootprint

8.5.2.3 LLGL_EXPORT std::uint32_t LLGL::GetMemoryFootprint (const ImageFormat *imageFormat*, const DataType *dataType*, std::uint32_t *count*)

Returns the memory footprint (in bytes) of an image with the specified format, data type, and number of elements.

Parameters

in	*imageFormat*	Specifies the image format.
in	*dataType*	Specifies the data type of each pixel component.
in	*count*	Specifies the number of elements that make up the memory footprint.

8.5.2.4 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

8.5.2.5 LLGL_EXPORT bool LLGL::IsCompressedFormat (const Format *format*)

Returns true if the specified hardware format is a compressed format, e.g. Format::BC1UNorm, Format::BC2U↩Norm_sRGB, Format::BC4SNorm, etc.

See also

Format

8.5.2.6 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

8.5.2.7 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

8.5.2.8 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

8.5.2.9 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

8.5.2.10 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

8.5.2.11 LLGL_EXPORT bool LLGL::IsIntegralFormat (const Format *format*)

Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::RGBA8UNorm, Format::R8SInt etc.).

Remarks

This also includes all normalized formats.

See also

IsNormalizedFormat
Format

8.5.2.12 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

8.5.2.13 LLGL_EXPORT bool LLGL::IsStencilFormat (const Format *format*)

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

See also

Format

8.6 Data type utility functions.

Functions

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

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

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

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

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

8.6.1 Detailed Description

8.6.2 Function Documentation

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

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

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

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

Returns

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

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

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

Returns

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

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

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

Returns

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

8.7 Image utility functions to classify and convert image data.

Functions

- LLGL_EXPORT bool LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const DstImageDescriptor &dstImageDesc, unsigned 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, ImageFormat dstFormat, DataType dstDataType, unsigned 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::DecompressImageBufferToRGBA8UNorm (const SrcImageDescriptor &srcImageDesc, const Extent2D &extent, unsigned threadCount=0)

  Decompresses the specified image buffer to RGBA format with 8-bit unsigned normalized integers.
- LLGL_EXPORT void LLGL::CopyImageBufferRegion (const DstImageDescriptor &dstImageDesc, const Offset3D &dstOffset, std::uint32_t dstRowStride, std::uint32_t dstSliceStride, const SrcImageDescriptor &srcImageDesc, const Offset3D &srcOffset, std::uint32_t srcRowStride, std::uint32_t srcSliceStride, const Extent3D &extent)

  Copies an image buffer region from the source buffer to the destination 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::AllocateByteBuffer (std::size_t bufferSize)

  Generates a new byte buffer with zeros in each byte.
- LLGL_EXPORT ByteBuffer LLGL::AllocateByteBuffer (std::size_t bufferSize, UninitializeTag)

  Generates a new and uninitialized byte buffer.

8.7.1 Detailed Description

8.7.2 Function Documentation

8.7.2.1 LLGL_EXPORT ByteBuffer LLGL::AllocateByteBuffer (std::size_t *bufferSize*)

Generates a new byte buffer with zeros in each byte.

Parameters

in	*bufferSize*	Specifies the size (in bytes) of the buffer.

Returns

The new allocated and initialized byte buffer.

Remarks

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

See also

GenerateImageBuffer
AllocateByteBuffer(std::size_t, UninitializeTag)

8.7.2.2 LLGL_EXPORT ByteBuffer LLGL::AllocateByteBuffer (std::size_t *bufferSize*, UninitializeTag)

Generates a new and uninitialized byte buffer.

Parameters

in	*bufferSize*	Specifies the size (in bytes) of the buffer.

Returns

The new allocated and uninitialized byte buffer.

See also

AllocateByteBuffer(std::size_t)

8.7.2.3 LLGL_EXPORT bool LLGL::ConvertImageBuffer (const SrcImageDescriptor & *srcImageDesc*, const DstImageDescriptor & *dstImageDesc*, unsigned *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

8.7.2.4 LLGL_EXPORT ByteBuffer LLGL::ConvertImageBuffer (const SrcImageDescriptor & *srcImageDesc*, ImageFormat *dstFormat*, DataType *dstDataType*, unsigned *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

8.7.2.5 LLGL_EXPORT void LLGL::CopyImageBufferRegion (const DstImageDescriptor & *dstImageDesc*, const Offset3D & *dstOffset*, std::uint32_t *dstRowStride*, std::uint32_t *dstSliceStride*, const SrcImageDescriptor & *srcImageDesc*, const Offset3D & *srcOffset*, std::uint32_t *srcRowStride*, std::uint32_t *srcSliceStride*, const Extent3D & *extent*)

Copies an image buffer region from the source buffer to the destination buffer.

Parameters

out	*dstImageDesc*	Specifies the destination image descriptor.
in	*dstOffset*	Specifies the 3D offset of the destination image.
in	*dstRowStride*	Specifies the number of pixels for each row in the destination image.
in	*dstSliceStride*	Specifies the number of pixels for each slice in the destination image.
in	*srcImageDesc*	Specifies the source image descriptor.
in	*srcOffset*	Specifies the 3D offset of the source image.
in	*srcRowStride*	Specifies the number of pixels for each row in the source image.
in	*srcSliceStride*	Specifies the number of pixels for each slice in the source image.
in	*extent*	Specifies the region extent to be copied.

Remarks

Only performs a bitwise copy. No blending or other operation is performed.

Exceptions

std::invalid_argument	If the destination 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 source buffer is a null pointer.
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 source and destination image descriptors do not have the same format and data type.
std::out_of_range	If `srcOffset` plus `extent` is outside the boundary of the source image.
std::out_of_range	If `dstOffset` plus `extent` is outside the boundary of the destination image.

8.7.2.6 LLGL_EXPORT ByteBuffer LLGL::DecompressImageBufferToRGBA8UNorm (const SrcImageDescriptor & *srcImageDesc*, const Extent2D & *extent*, unsigned *threadCount* = `0`)

Decompresses the specified image buffer to RGBA format with 8-bit unsigned normalized integers.

Parameters

in	*srcImageDesc*	Specifies the source image descriptor.
in	*extent*	Specifies the image extent. This is required as most compression formats work in block sizes.
in	*threadCount*	Specifies the number of threads to use for decompression. 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 decompressed image data or null if the compression format is not supported for decompression.

8.7.2.7 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 }
);
```

8.8 Buffer argument structures for indirect draw and compute commands.

Classes

- struct LLGL::DrawIndirectArguments

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

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

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

  Format structure for the arguments of an indirect compute command.

8.8.1 Detailed Description

8.9 Global type-to-string conversion functions.

Functions

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

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

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

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

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

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

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

8.9.1 Detailed Description

8.9.2 Function Documentation

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

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

Remarks

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

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

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

Remarks

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

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

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

Remarks

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

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

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

Remarks

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

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

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

Remarks

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

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

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

Remarks

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

8.10 Global utility functions, especially to fill descriptor structures.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Returns a ShaderDescriptor structure.
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const ShaderReflection &reflection)

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

  Generates a pipeline layout descriptor by parsing the specified string.
- LLGL_EXPORT RenderPassDescriptor LLGL::RenderPassDesc (const RenderTargetDescriptor &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 *cpuAccessFlags* = 0)

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 Format *format*, long *cpuAccessFlags* = 0)

Returns a BufferDescriptor structure for an index buffer.

See also

RenderSystem::CreateBuffer

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

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

Remarks

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

8.10.2.4 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 pair of binding point type (i.e. BindingDescriptor::type) and binding flags (i.e. BindingDescriptor::bindFlags) is specified by one of the following identifiers: – `cbuffer` for constant buffers (i.e. ResourceType::Buffer and BindFlags::ConstantBuffer). – `buffer` for sampled buffers (i.e. ResourceType::Buffer and BindFlags::Sampled). – `rwbuffer` for read/write storage buffers (i.e. ResourceType::Buffer and BindFlags::Storage). – `texture` for textures (i.e. ResourceType::Texture and BindFlags::Sampled). – `rwtexture` for read/write textures (i.e. ResourceType::Texture and BindFlags::Storage). – `sampler` for sampler states (i.e. ResourceType::Sampler). • Optionally, the resource **name** is specified as an arbitrary identifier followed by the at-sign (e.g. `"texture(myColorMap@1)"`). • 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 being optional (e.g. `"texture(1),sampler(2),"` or `"texture(1),sampler(2)"`). • The stage flags (i.e. BindingDescriptor::stageFlags) can be specified after 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{ "Scene",
      LLGL::ResourceType::Buffer,
      LLGL::BindFlags::ConstantBuffer,
      LLGL::StageFlags::FragmentStage |
      LLGL::StageFlags::VertexStage, 0u,     },
    LLGL::BindingDescriptor{
      LLGL::ResourceType::Texture, LLGL::BindFlags::Sampled,
            LLGL::StageFlags::FragmentStage,                                1u
        },
    LLGL::BindingDescriptor{ "TexArray",
      LLGL::ResourceType::Texture, LLGL::BindFlags::Sampled,
            LLGL::StageFlags::FragmentStage,                                2u, 4u
      , },
    LLGL::BindingDescriptor{
      LLGL::ResourceType::Sampler, 0,
      LLGL::StageFlags::FragmentStage,                              3u      },
};

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

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

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

Exceptions

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

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

Returns a BufferDescriptor structure for a storage buffer.

See also

RenderSystem::CreateBuffer

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

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

See also

RenderSystem::CreateTexture

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

Returns a BufferDescriptor structure for a vertex buffer.

See also

RenderSystem::CreateBuffer

8.11 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 TextureType type, const Extent3D &extent)

  Returns the number of MIP-map levels for the specified texture attributes.
- 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::NumMipTexels (const TextureType type, const Extent3D &extent, std↩::uint32_t mipLevel)

  Returns the number of texture elements (texels) for the specified texture attributes, or zero extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureType type, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the number of texture elements (texels) for the specified texture subresource range.
- LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureDescriptor &textureDesc, std::uint32_↩t mipLevel=∼0u)

  Returns the number of texture elements (texels) for the specified texture descriptor.
- LLGL_EXPORT std::uint32_t LLGL::NumMipDimensions (const TextureType type)

  Returns the number of MIP-map dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.
- LLGL_EXPORT std::uint32_t LLGL::NumTextureDimensions (const TextureType type)

  Returns the number of texture dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.
- LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureType type, const Extent3D &extent, std↩::uint32_t mipLevel)

  Returns the MIP-map extent (including array layers) for the specified texture type, or an empty extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureDescriptor &textureDesc, std::uint32_t mip↩Level=0)

  Returns the MIP-map extent (including array layers) for the specified texture descriptor.
- std::uint32_t LLGL::GetMemoryFootprint (const TextureType type, const Format format, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the memory footprint (in bytes) of a texture subresource with the specified hardware format and extent.
- LLGL_EXPORT bool LLGL::IsMipMappedTexture (const TextureDescriptor &textureDesc)

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

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

  Returns true if the specified texture type is a multi-sample texture.
- LLGL_EXPORT bool LLGL::IsCubeTexture (const TextureType type)

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

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

8.11.1 Detailed Description

8.11.2 Function Documentation

8.11.2.1 std::uint32_t LLGL::GetMemoryFootprint (const TextureType *type,* const Format *format,* const Extent3D & *extent,* const TextureSubresource & *subresource*)

Returns the memory footprint (in bytes) of a texture subresource with the specified hardware format and extent.

Parameters

in	*type*	Specifies the texture type which determines how extent and subresource will result into the final texture dimensions.
in	*format*	Specifies the hardware format.
in	*extent*	Specifies the extent of the texture subresource.
in	*subresource*	Specifies the texture subresource.

See also

GetMemoryFootprint(const Format, std::uint32_t)
Texture::GetMemoryFootprint
Texture::GetMipExtent
Texture::GetFormat

8.11.2.2 LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureType *type*, const Extent3D & *extent*, std::uint32_t *mipLevel*)

Returns the MIP-map extent (including array layers) for the specified texture type, or an empty extent if `mipLevel` is out of bounds (see `NumMipLevels`).

Parameters

in	*type*	Specifies the texture type for which the MIP-map extent is to be determined.
in	*extent*	Specifies the extent of the first MIP-map level.
in	*mipLevel*	Specifies the MIP-map level whose extent is to be determined. The first and largest MIP-map level has index zero.

See also

Texture::GetMipExtent
TextureDescriptor::extent
NumMipLevels(const TextureType, const Extent3D&)

8.11.2.3 LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureDescriptor & *textureDesc*, std::uint32_t *mipLevel* = `0`)

Returns the MIP-map extent (including array layers) for the specified texture descriptor.

Parameters

in	*textureDesc*	Specifies the texture descriptor.
in	*mipLevel*	Specifies the MIP-map level whose extent is to be determined. The first and largest MIP-map level has index zero.

See also

Texture::GetMipExtent
TextureDescriptor::extent
TextureDescriptor::arrayLayers

8.11.2.4 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.

8.11.2.5 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.

8.11.2.6 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

8.11.2.7 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.

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

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

Returns

True if the components are mapped as follows:

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

See also

TextureSwizzle

8.11.2.9 LLGL_EXPORT std::uint32_t LLGL::NumMipDimensions (const TextureType *type*)

Returns the number of MIP-map dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.

Remarks

MIP-map dimensions **do count** array layers as a dimension, e.g. for TextureType::Texture2DArray this function returns 3.

See also

Texture::GetMipExtent
NumTextureDimensions

8.11.2.10 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.

8.11.2.11 LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (const TextureType *type*, const Extent3D & *extent*)

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

Parameters

in	*type*	Specifies the texture type for which the MIP-map extent is to be determined.
in	*extent*	Specifies the extent of the first MIP-map level.

See also

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

8.11.2.12 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(const TextureType, const Extent3D&)

8.11.2.13 LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureType *type*, const Extent3D & *extent*, std::uint32_t *mipLevel*)

Returns the number of texture elements (texels) for the specified texture attributes, or zero extent if `mipLevel` is out of bounds (see `NumMipLevels`).

Parameters

in	*type*	Specifies the texture type for which the number of texels are to be determined.
in	*extent*	Specifies the extent of the first MIP-map level.
in	*mipLevel*	Specifies the MIP-map level whose number of texels is to be determined. The first and largest MIP-map level has index zero.

See also

TextureDescriptor::type
NumMipLevels(const TextureType, const Extent3D&)

8.11.2.14 LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureType *type*, const Extent3D & *extent*, const TextureSubresource & *subresource*)

Returns the number of texture elements (texels) for the specified texture subresource range.

Parameters

in	*type*	Specifies the texture type for which the number of texels are to be determined.
in	*extent*	Specifies the extent of the first MIP-map level.
in	*subresource*	Specifies the subresource range. Which dimension in `extent` specifies the number of array layers depends on the texture type.

See also

TextureDescriptor::arrayLayers
TextureSubresource::baseArrayLayer
TextureSubresource::numArrayLayers
NumMipTexels(const TextureType, const Extent3D&, std::uint32_t)

8.11.2.15 LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureDescriptor & *textureDesc*, std::uint32_t *mipLevel* = `~0u`)

Returns the number of texture elements (texels) for the specified texture descriptor.

Parameters

in	*textureDesc*	Specifies the texture descriptor for which the number of MIP-map texels are determined.
in	*mipLevel*	Optional parameter to specify only a single MIP-map level. If this is `0xFFFFFFFF`, the number of texels for the entire MIP-map chain is determined. By default `0xFFFFFFFF`. The size of the MIP-map chain is determined by `textureDesc.mipLevels`.

See also

NumMipTexels(const TextureType, const Extent3D&, std::uint32_t)
TextureDescriptor::mipLevels

8.11.2.16 LLGL_EXPORT std::uint32_t LLGL::NumTextureDimensions (const TextureType *type*)

Returns the number of texture dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.

Remarks

Texture dimensions **don't count** array layers as a dimension, e.g. for TextureType::Texture2DArray this function returns 2.

See also

TextureDescriptor::extent
NumMipDimensions

Chapter 9

Namespace Documentation

9.1 LLGL Namespace Reference

Namespaces

- Constants

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

  Namespace with functions to determine LLGL version.

Classes

- struct ApplicationDescriptor

  Application descriptor structure.
- class ArrayView

  Constant array view container template. Keeps pointer to array and length.
- struct AttachmentClear

  Attachment clear command structure.
- struct AttachmentDescriptor

  Render target attachment descriptor structure.
- struct AttachmentFormatDescriptor

  Render pass attachment descriptor structure.
- struct BarrierFlags

  Flags for memory barriers in resource heaps.
- class BasicStringView

  Constant string view container template. Keeps pointer to string and its length.
- struct BindFlags

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

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

  Blending state descriptor structure.
- struct BlendTargetDescriptor

Blend target state descriptor structure.

- class Blob

  CPU read-only buffer of arbitrary size.
- class Buffer

  Hardware buffer interface.
- class BufferArray

  Hardware buffer container interface.
- struct BufferDescriptor

  Hardware buffer descriptor structure.
- struct BufferViewDescriptor

  Buffer view descriptor structure.
- class Canvas

  Canvas interface for mobile platforms.
- struct CanvasDescriptor

  Canvas descriptor structure.
- struct ClearFlags

  Command buffer clear flags.
- struct ClearValue

  Clear value structure for color, depth, and stencil clear operations.
- class Color

  Base color class with N components.
- class Color< T, 3u >

  RGB color class with components: r, g, and b.
- class Color< T, 4u >

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

  Command buffer interface used for storing and encoding GPU commands.
- struct CommandBufferDescriptor

  Command buffer descriptor structure.
- struct CommandBufferFlags

  Command buffer creation flags.
- class CommandQueue

  Command queue interface.
- struct ComputePipelineDescriptor

  Compute pipeline state descriptor structure.
- struct ComputeShaderAttributes

  Compute shader specific descriptor structure.
- struct CPUAccessFlags

  CPU read/write access flag enumeration for mapping GPU into CPU memory space.
- struct DepthBiasDescriptor

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

  Depth state descriptor structure.
- struct DispatchIndirectArguments

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

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

  Display mode descriptor structure.
- struct DrawIndexedIndirectArguments

  Format structure for the arguments of an indirect indexed draw command.

- struct DrawIndirectArguments

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

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

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

  2-Dimensional extent structure.
- struct Extent3D

  3-Dimensional extent structure.
- class Fence

  Fence interface for CPU/GPU synchronization.
- struct FormatAttributes

  Vertex and texture format attributes structure.
- struct FormatFlags

  Format attribute flags enumeration.
- struct FragmentAttribute

  Fragment output attribute structure.
- struct FragmentShaderAttributes

  Fragment shader specific descriptor structure.
- struct FrameProfile

  Profile of a rendered frame.
- struct GraphicsPipelineDescriptor

  Graphics pipeline state descriptor structure.
- struct GrowStrategyAddHalf

  Container capacity grow strategy to increase the capacity to 150%.
- struct GrowStrategyDouble

  Container capacity grow strategy to increase the capacity to 200%.
- struct GrowStrategyRoundUpPow2

  Container capacity grow strategy to increase the capacity to the next higher power of two.
- class Image

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

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

  Base class for all interfaces in LLGL.
- struct InterfaceID

  Enumeration of all interfaces in LLGL.
- struct MetalDependentStateDescriptor

  Graphics API dependent state descriptor for the Metal renderer.
- struct MiscFlags

  Miscellaneous resource flags.
- struct NativeContextHandle

  Android native context handle structure.
- struct NativeHandle

  Android native handle structure.
- class NonCopyable

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

  2-Dimensional offset structure.
- struct Offset3D

3-Dimensional offset structure.

- class PipelineLayout

  Pipeline layout interface.
- struct PipelineLayoutDescriptor

  Pipeline layout descritpor structure.
- class PipelineState

  Graphics and compute pipeline state interface.
- struct ProfileTimeRecord

  Structure with annotation and elapsed time for a timer profile.
- 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.
- struct RendererConfigurationOpenGL

  OpenGL profile descriptor structure.
- struct RendererConfigurationOpenGLES3

  OpenGL ES 3 profile descriptor structure.
- struct RendererConfigurationVulkan

  Structure for a Vulkan renderer specific configuration.
- struct RendererID

  Renderer identification number enumeration.
- struct RendererInfo

  Renderer basic information structure.
- struct RenderingCapabilities

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

  Rendering debugger interface.
- struct RenderingFeatures

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

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

  Rendering profiler model class.
- class RenderPass

  Render pass interface.
- struct RenderPassDescriptor

  Render pass descriptor structure.
- class RenderSystem

  Render system interface.
- class RenderSystemChild

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

  Delegate to delete an instance of the RenderSystem interface.
- struct RenderSystemDescriptor

  Render system descriptor structure.
- class RenderTarget

  Render target interface.

- struct RenderTargetDescriptor

  Render target descriptor structure.
- class Report

  Error and warning report interface.
- struct ResizeBuffersFlags

  Swap-chain resize buffers flags.
- class Resource

  Base class for all hardware resource interfaces.
- class ResourceHeap

  Resource heap interface.
- struct ResourceHeapDescriptor

  Resource heap descriptor structure.
- struct ResourceViewDescriptor

  Resource view descriptor structure.
- class Sampler

  Sampler interface.
- struct SamplerDescriptor

  Texture sampler descriptor structure.
- struct Scissor

  Scissor dimensions.
- class Shader

  Shader interface.
- struct ShaderCompileFlags

  Shader compilation flags enumeration.
- struct ShaderDescriptor

  Shader source and binary code descriptor structure.
- struct ShaderMacro

  Shader macro structure with name and optional body definition.
- struct ShaderReflection

  Shader reflection structure.
- struct ShaderResource

  Shader reflection resource structure.
- struct ShaderUniform

  Shader reflection uniform structure.
- class SmallVector

  Generic container class for consecutive arrays optimized for small sizes.
- 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.
- class Surface

  The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on mobile platforms).
- class SwapChain

  Swap-chain interface.
- struct SwapChainDescriptor

  Swap chain descriptor structure.
- struct TessellationDescriptor

Tessellation descriptor structure for the graphics pipeline.

- class Texture

  Texture interface.
- struct TextureDescriptor

  Texture descriptor structure.
- struct TextureLocation

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

  Texture region structure: Subresource (MIP-map level and array layer range), offset, and extent.
- struct TextureSubresource

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

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

  Texture view descriptor structure.
- struct UninitializeTag

  Common uninitialize tag.
- class UTF8String

  Container class for UTF-8 encoded strings.
- struct VertexAttribute

  Vertex input/output attribute structure.
- struct VertexFormat

  Utility structure to store vertex attributes.
- struct VertexShaderAttributes

  Vertex (or geometry) shader specific structure.
- struct VideoAdapterDescriptor

  Video adapter descriptor structure.
- struct VideoOutputDescriptor

  Video output structure.
- struct Viewport

  Viewport dimensions.
- 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 StringView = BasicStringView< char >

  ANSI character string view (char).
- using WStringView = BasicStringView< wchar_t >

  Wide character string view (wchar_t).
- using ByteBuffer = std::unique_ptr< char[ ]>

  Common byte buffer type.
- using RenderSystemPtr = std::unique_ptr< RenderSystem, RenderSystemDeleter >

  Unique pointer type for the RenderSystem interface with a custom deleter.
- 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 RenderConditionMode { RenderConditionMode::Wait, RenderConditionMode::NoWait, RenderConditionMode::ByRegionWait, RenderConditionMode::ByRegionNoWait, RenderConditionMode::WaitInverted, RenderConditionMode::NoWaitInverted, RenderConditionMode::By↩RegionWaitInverted, RenderConditionMode::ByRegionNoWaitInverted }

  Render condition mode enumeration.
- enum StencilFace { StencilFace::FrontAndBack, StencilFace::Front, StencilFace::Back }

  Stencil face enumeration.
- enum PipelineBindPoint { PipelineBindPoint::Undefined, PipelineBindPoint::Graphics, PipelineBindPoint::↩Compute }

  Pipeline binding point enumeration.
- enum Format { Format::Undefined, Format::A8UNorm, Format::R8UNorm, Format::R8SNorm, Format::R8UInt, Format::R8SInt, Format::R16UNorm, Format::R16SNorm, Format::R16UInt, Format::R16SInt, Format::R16Float, Format::R32UInt, Format::R32SInt, Format::R32Float, Format::R64Float, Format::RG8UNorm, Format::RG8SNorm, Format::RG8UInt, Format::RG8SInt, Format::RG16UNorm, Format::RG16SNorm, Format::RG16UInt, Format::RG16SInt, Format::RG16Float, Format::RG32UInt, Format::RG32SInt, Format::RG32Float, Format::RG64Float, Format::RGB8UNorm, Format::RGB8UNorm_sRGB, Format::RGB8SNorm, Format::RGB8UInt, Format::RGB8SInt, Format::RGB16UNorm, Format::RGB16SNorm, Format::RGB16UInt, Format::RGB16SInt, Format::RGB16Float, Format::RGB32UInt, Format::RGB32SInt, Format::RGB32Float, Format::RGB64Float, Format::RGBA8UNorm, Format::RGBA8UNorm_sRGB, Format::RGBA8SNorm, Format::RGBA8UInt, Format::RGBA8SInt, Format::RGBA16UNorm, Format::RGBA16SNorm, Format::RGBA16UInt, Format::RGBA16SInt, Format::RGBA16Float, Format::RGBA32UInt, Format::RGBA32SInt, Format::RGBA32Float, Format::RGBA64Float, Format::BGRA8UNorm, Format::BGRA8UNorm_sRGB, Format::BGRA8SNorm, Format::BGRA8UInt, Format::BGRA8SInt, Format::RGB10A2UNorm, Format::RGB10A2UInt, Format::RG11B10Float, Format::RGB9E5Float, Format::D16UNorm, Format::D24UNormS8UInt, Format::D32Float, Format::D32FloatS8X24UInt, Format::BC1UNorm, Format::BC1UNorm_sRGB, Format::BC2UNorm, Format::BC2UNorm_sRGB, Format::BC3UNorm, Format::BC3UNorm_sRGB, Format::BC4UNorm, Format::BC4SNorm, Format::BC5UNorm, Format::BC5SNorm }

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

- enum ImageFormat {
  ImageFormat::Alpha, ImageFormat::R, ImageFormat::RG, ImageFormat::RGB,
  ImageFormat::BGR, ImageFormat::RGBA, ImageFormat::BGRA, ImageFormat::ARGB,
  ImageFormat::ABGR, ImageFormat::Depth, ImageFormat::DepthStencil, ImageFormat::BC1,
  ImageFormat::BC2, ImageFormat::BC3, ImageFormat::BC4, ImageFormat::BC5 }

  Image format enumeration that applies to each pixel of an image.
- enum DataType {
  DataType::Undefined, DataType::Int8, DataType::UInt8, DataType::Int16,
  DataType::UInt16, DataType::Int32, DataType::UInt32, DataType::Float16,
  DataType::Float32, DataType::Float64 }

  Data types enumeration used for CPU side operations.
- enum 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 PrimitiveTopology {
  PrimitiveTopology::PointList, PrimitiveTopology::LineList, PrimitiveTopology::LineStrip, PrimitiveTopology::↩
  LineListAdjacency,
  PrimitiveTopology::LineStripAdjacency, PrimitiveTopology::TriangleList, PrimitiveTopology::TriangleStrip,
  PrimitiveTopology::TriangleListAdjacency,
  PrimitiveTopology::TriangleStripAdjacency, PrimitiveTopology::Patches1, PrimitiveTopology::Patches2,

PrimitiveTopology::Patches3, PrimitiveTopology::Patches4, PrimitiveTopology::Patches5, PrimitiveTopology::Patches6, PrimitiveTopology::Patches7, PrimitiveTopology::Patches8, PrimitiveTopology::Patches9, PrimitiveTopology::Patches10, PrimitiveTopology::Patches11, PrimitiveTopology::Patches12, PrimitiveTopology::Patches13, PrimitiveTopology::Patches14, PrimitiveTopology::Patches15, PrimitiveTopology::Patches16, PrimitiveTopology::Patches17, PrimitiveTopology::Patches18, PrimitiveTopology::Patches19, PrimitiveTopology::Patches20, PrimitiveTopology::Patches21, PrimitiveTopology::Patches22, PrimitiveTopology::Patches23, PrimitiveTopology::Patches24, PrimitiveTopology::Patches25, PrimitiveTopology::Patches26, PrimitiveTopology::Patches27, PrimitiveTopology::Patches28, PrimitiveTopology::Patches29, PrimitiveTopology::Patches30, PrimitiveTopology::Patches31, PrimitiveTopology::Patches32 }

Primitive topology enumeration.

- enum CompareOp { CompareOp::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 TessellationPartition { TessellationPartition::Undefined, TessellationPartition::Integer, TessellationPartition::Pow2, TessellationPartition::FractionalOdd, TessellationPartition::FractionalEven }

  Tessellation partition mode enumeration.
- enum QueryType { QueryType::SamplesPassed, QueryType::AnySamplesPassed, QueryType::AnySamplesPassedConservative, QueryType::TimeElapsed, QueryType::StreamOutPrimitivesWritten, QueryType::StreamOutOverflow, QueryType::PipelineStatistics }

Query type enumeration.

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

  OpenGL context profile enumeration.

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

  Rendering debugger error types enumeration.

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

  Rendering debugger warning types enumeration.

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

  Enumeration for render pass attachment load operations.

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

  Enumeration for render pass attachment store operations.

- enum ShadingLanguage {
  ShadingLanguage::GLSL = (0x10000), ShadingLanguage::GLSL_110 = (0x10000 | 110), 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::HLSL_6_0 = (0x30000 | 600), ShadingLanguage::HLSL_6_1 = (0x30000 | 601), ShadingLanguage::HLSL_6_2 = (0x30000 | 602), ShadingLanguage::HLSL_6_3 = (0x30000 | 603), ShadingLanguage::HLSL_6_4 = (0x30000 | 604), ShadingLanguage::Metal = (0x40000), Shading↩ Language::Metal_1_0 = (0x40000 | 100), ShadingLanguage::Metal_1_1 = (0x40000 | 110), ShadingLanguage::Metal_1_2 = (0x40000 | 120), ShadingLanguage::Metal_2_0 = (0x40000 | 200), ShadingLanguage::Metal_2_1 = (0x40000 | 210), ShadingLanguage::SPIRV = (0x50000), ShadingLanguage::SPIRV_100 = (0x50000 | 100), ShadingLanguage::VersionBitmask = 0x0000ffff }

  Shading language version enumation.

- enum ScreenOrigin { ScreenOrigin::LowerLeft, ScreenOrigin::UpperLeft }

  Screen coordinate system origin enumeration.

- enum ClippingRange { ClippingRange::MinusOneToOne, ClippingRange::ZeroToOne }

  Clipping depth range enumeration.

- enum CPUAccess { CPUAccess::ReadOnly, CPUAccess::WriteOnly, CPUAccess::WriteDiscard, 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::Buffer, 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, ShaderSourceType::BinaryBuffer, ShaderSourceType::BinaryFile }

  Shader source type enumeration.

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

  Shader uniform type enumeration.

- enum StorageBufferType { StorageBufferType::Undefined, StorageBufferType::TypedBuffer, StorageBufferType::StructuredBuffer, StorageBufferType::ByteAddressBuffer, StorageBufferType::RWTypedBuffer, StorageBufferType::RWStructuredBuffer, StorageBufferType::RWByteAddressBuffer, StorageBufferType::AppendStructuredBuffer, StorageBufferType::ConsumeStructuredBuffer }

  Storage buffer type enumeration for shader reflection.

- enum SystemValue { SystemValue::Undefined, SystemValue::ClipDistance, SystemValue::Color, SystemValue::CullDistance, SystemValue::Depth, SystemValue::DepthGreater, SystemValue::DepthLess, SystemValue::FrontFacing, SystemValue::InstanceID, SystemValue::Position, SystemValue::PrimitiveID, SystemValue::RenderTargetIndex, SystemValue::SampleMask, SystemValue::SampleID, SystemValue::Stencil, SystemValue::VertexID, SystemValue::ViewportIndex }

  Enumeration of system values for vertex input or fragment output attributes.

- enum TextureType { TextureType::Texture1D, TextureType::Texture2D, TextureType::Texture3D, TextureType::TextureCube, TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, TextureType::Texture2DMS, TextureType::Texture2DMSArray }

  Texture type enumeration.

- enum TextureSwizzle : std::uint8_t { TextureSwizzle::Zero, TextureSwizzle::One, TextureSwizzle::Red, TextureSwizzle::Green, TextureSwizzle::Blue, TextureSwizzle::Alpha }

  Texture component swizzle enumeration.

Functions

- LLGL_EXPORT bool IsTypedBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a typed buffer, i.e. `Buffer` or `RWBuffer` in HLSL.

- LLGL_EXPORT bool IsStructuredBuffer (const BufferDescriptor &desc)

Returns true if the buffer descriptor denotes a structured buffer, i.e. `StructuredBuffer`, `RWStructuredBuffer`, `AppendStructuredBuffer`, or `ConsumeStructuredBuffer` in HLSL.

- LLGL_EXPORT bool IsByteAddressBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a byte addresse buffer, i.e. `ByteAddressBuffer` or `RWByteAddressBuffer` in HLSL.

- 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.

- UTF8String operator+ (const UTF8String &lhs, const UTF8String &rhs)

  Concatenates the two UTF-8 string.

- UTF8String operator+ (const UTF8String &lhs, const StringView &rhs)

- UTF8String operator+ (const UTF8String &lhs, const WStringView &rhs)
- UTF8String operator+ (const UTF8String &lhs, const char *rhs)
- UTF8String operator+ (const UTF8String &lhs, const wchar_t *rhs)
- UTF8String operator+ (const StringView &lhs, const UTF8String &rhs)
- UTF8String operator+ (const WStringView &lhs, const UTF8String &rhs)
- UTF8String operator+ (const char *lhs, const UTF8String &rhs)
- UTF8String operator+ (const wchar_t *lhs, const UTF8String &rhs)
- bool operator== (const UTF8String &lhs, const UTF8String &rhs)

  Returns true if the two strings are equal.
- bool operator== (const UTF8String &lhs, const StringView &rhs)
- bool operator== (const UTF8String &lhs, const WStringView &rhs)
- bool operator== (const UTF8String &lhs, const char *rhs)
- bool operator== (const UTF8String &lhs, const wchar_t *rhs)
- bool operator== (const StringView &lhs, const StringView &rhs)
- bool operator== (const StringView &lhs, const UTF8String &rhs)
- bool operator== (const StringView &lhs, const char *rhs)
- bool operator== (const WStringView &lhs, const WStringView &rhs)
- bool operator== (const WStringView &lhs, const UTF8String &rhs)
- bool operator== (const WStringView &lhs, const wchar_t *rhs)
- bool operator== (const char *lhs, const UTF8String &rhs)
- bool operator== (const char *lhs, const StringView &rhs)
- bool operator== (const wchar_t *lhs, const UTF8String &rhs)
- bool operator== (const wchar_t *lhs, const WStringView &rhs)
- bool operator!= (const UTF8String &lhs, const UTF8String &rhs)

  Returns true if the two strings are not equal.
- bool operator!= (const UTF8String &lhs, const StringView &rhs)
- bool operator!= (const UTF8String &lhs, const WStringView &rhs)
- bool operator!= (const UTF8String &lhs, const char *rhs)
- bool operator!= (const UTF8String &lhs, const wchar_t *rhs)
- bool operator!= (const StringView &lhs, const StringView &rhs)
- bool operator!= (const StringView &lhs, const UTF8String &rhs)
- bool operator!= (const StringView &lhs, const char *rhs)
- bool operator!= (const WStringView &lhs, const WStringView &rhs)
- bool operator!= (const WStringView &lhs, const UTF8String &rhs)
- bool operator!= (const WStringView &lhs, const wchar_t *rhs)
- bool operator!= (const char *lhs, const UTF8String &rhs)
- bool operator!= (const char *lhs, const StringView &rhs)
- bool operator!= (const wchar_t *lhs, const UTF8String &rhs)
- bool operator!= (const wchar_t *lhs, const WStringView &rhs)
- LLGL_EXPORT bool operator== (const DisplayModeDescriptor &lhs, const DisplayModeDescriptor &rhs)

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

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

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

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

  Returns the attributes for the specified hardware format or a zero-initialized structure in case of an invalid argument.
- 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 GetMemoryFootprint (const Format format, std::uint32_t count)

Returns the memory footprint (in bytes) of a texture or buffer with the specified hardware format and number of elements.

- LLGL_EXPORT std::uint32_t GetMemoryFootprint (const ImageFormat imageFormat, const DataType dataType, std::uint32_t count)

  Returns the memory footprint (in bytes) of an image with the specified format, data type, and number of elements.
- LLGL_EXPORT bool IsCompressedFormat (const Format format)

  Returns true if the specified hardware format is a compressed format, e.g. Format::BC1UNorm, Format::BC2UNorm_sRGB, Format::BC4SNorm, etc.
- 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 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 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 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::D32FloatS8X24UInt.
- 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::RGBA8UNorm, 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 ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const DstImageDescriptor &dstImageDesc, unsigned 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, unsigned 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 DecompressImageBufferToRGBA8UNorm (const SrcImageDescriptor &srcImageDesc, const Extent2D &extent, unsigned threadCount=0)

  Decompresses the specified image buffer to RGBA format with 8-bit unsigned normalized integers.
- LLGL_EXPORT void CopyImageBufferRegion (const DstImageDescriptor &dstImageDesc, const Offset3D &dstOffset, std::uint32_t dstRowStride, std::uint32_t dstSliceStride, const SrcImageDescriptor &srcImageDesc, const Offset3D &srcOffset, std::uint32_t srcRowStride, std::uint32_t srcSliceStride, const Extent3D &extent)

Copies an image buffer region from the source buffer to the destination 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 AllocateByteBuffer (std::size_t bufferSize)

  Generates a new byte buffer with zeros in each byte.

- LLGL_EXPORT ByteBuffer AllocateByteBuffer (std::size_t bufferSize, UninitializeTag)

  Generates a new and uninitialized byte buffer.

- LLGL_EXPORT const char * ToString (const ShaderType t)

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

- LLGL_EXPORT const char * ToString (const ErrorType t)

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

- LLGL_EXPORT const char * ToString (const WarningType t)

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

- LLGL_EXPORT const char * ToString (const ShadingLanguage t)

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

- LLGL_EXPORT const char * ToString (const Format t)

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

- LLGL_EXPORT const char * ToString (const TextureType t)

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

- LLGL_EXPORT TextureDescriptor Texture1DDesc (Format format, std::uint32_t width, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- LLGL_EXPORT BufferDescriptor VertexBufferDesc (uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0)

  Returns a BufferDescriptor structure for a vertex buffer.

- LLGL_EXPORT BufferDescriptor IndexBufferDesc (uint64_t size, const Format format, long cpuAccessFlags=0)

Returns a BufferDescriptor structure for an index buffer.

- LLGL_EXPORT BufferDescriptor ConstantBufferDesc (uint64_t size, long cpuAccessFlags=0)

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

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

  Returns a ShaderDescriptor structure.
- LLGL_EXPORT PipelineLayoutDescriptor PipelineLayoutDesc (const ShaderReflection &reflection)

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

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

  Converts the specified render target descriptor into a render pass descriptor with default settings.
- LLGL_EXPORT bool IsPrimitiveTopologyPatches (const PrimitiveTopology primitiveTopology)

  Returns true if the specified primitive topology is a patch list.
- LLGL_EXPORT bool IsPrimitiveTopologyStrip (const PrimitiveTopology primitiveTopology)

  Returns true if the specified primitive topology is a strip that generates a new primitive with each new vertex.
- 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 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 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 TextureType type, const Extent3D &extent)

  Returns the number of MIP-map levels for the specified texture attributes.
- 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 NumMipTexels (const TextureType type, const Extent3D &extent, std::uint32_t mipLevel)

  Returns the number of texture elements (texels) for the specified texture attributes, or zero extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT std::uint32_t NumMipTexels (const TextureType type, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the number of texture elements (texels) for the specified texture subresource range.
- LLGL_EXPORT std::uint32_t NumMipTexels (const TextureDescriptor &textureDesc, std::uint32_t mip↩Level=~0u)

  Returns the number of texture elements (texels) for the specified texture descriptor.
- LLGL_EXPORT std::uint32_t NumMipDimensions (const TextureType type)

  Returns the number of MIP-map dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.
- LLGL_EXPORT std::uint32_t NumTextureDimensions (const TextureType type)

Returns the number of texture dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.

- LLGL_EXPORT Extent3D GetMipExtent (const TextureType type, const Extent3D &extent, std::uint32_↩t mipLevel)

  Returns the MIP-map extent (including array layers) for the specified texture type, or an empty extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT Extent3D GetMipExtent (const TextureDescriptor &textureDesc, std::uint32_t mipLevel=0)

  Returns the MIP-map extent (including array layers) for the specified texture descriptor.
- std::uint32_t GetMemoryFootprint (const TextureType type, const Format format, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the memory footprint (in bytes) of a texture subresource with the specified hardware format and extent.
- LLGL_EXPORT bool IsMipMappedTexture (const TextureDescriptor &textureDesc)

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

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

  Returns true if the specified texture type is a multi-sample texture.
- LLGL_EXPORT bool IsCubeTexture (const TextureType type)

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

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

  Returns whether the specified object is an instance of the interface specified by template parameter <T>.
- template<typename T >
  bool IsInstanceOf (const Interface &obj)

  Const-reference version of IsInstanceOf.
- template<typename T >
  const T * CastTo (const Interface *obj)

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

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

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

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

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

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

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

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

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

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

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

- LLGL_EXPORT Offset3D operator- (const Offset3D &lhs, const Offset3D &rhs)

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

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

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

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

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

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

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

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

  Returns true if the left hand side extent 'lhs' is unequal to the right hand side extent 'rhs'.
- LLGL_EXPORT 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::RenderSystemPtr = typedef std::unique_ptr<RenderSystem, RenderSystemDeleter>

Unique pointer type for the RenderSystem interface with a custom deleter.

See also

RenderSystem
RenderSystem::Load
RenderSystem::Unload
RenderSystemDeleter

9.1.1.13 using LLGL::StringView = typedef BasicStringView<char>

ANSI character string view (char).

9.1.1.14 using LLGL::UniformLocation = typedef std::int32_t

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

9.1.1.15 using LLGL::WStringView = typedef BasicStringView<wchar_t>

Wide character string view (wchar_t).

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

Todo Remove this enum and use Format instead.

Enumerator

Color Attachment is used for color output.

Remarks

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

Depth Attachment is used for depth component output.

Remarks

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

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

Remarks

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

Stencil Attachment is used for stencil index output.

Remarks

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

9.1.2.4 enum LLGL::BlendArithmetic [strong]

Blending arithmetic operations enumeration.

See also

BlendTargetDescriptor::colorArithmetic
BlendTargetDescriptor::alphaArithmetic

Enumerator

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

Subtract Subtract source 1 from source 2.

RevSubtract Subtract source 2 from source 1.

Min Find the minimum of source 1 and source 2.

Max Find the maximum of source 1 and source 2.

9.1.2.5 enum LLGL::BlendOp [strong]

Blending operations enumeration.

See also

BlendTargetDescriptor

Enumerator

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

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

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

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

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

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

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

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

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

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

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

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

See also

CommandBuffer::SetBlendFactor

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

See also

CommandBuffer::SetBlendFactor

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

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

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

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

9.1.2.6 enum LLGL::ClippingRange [strong]

Clipping depth range enumeration.

See also

RenderingCapabilities::clippingRange

Enumerator

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

Note

Native clipping depth range in: OpenGL.

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

Note

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

9.1.2.7 enum LLGL::CompareOp [strong]

Compare operations enumeration.

Remarks

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

See also

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

Enumerator

NeverPass Comparison never passes.

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

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

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

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

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

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

AlwaysPass Comparison always passes.

9.1.2.8 enum LLGL::CPUAccess [strong]

Classifications of CPU access to mapped resources.

See also

RenderSystem::MapBuffer
CPUAccessFlags

Enumerator

ReadOnly CPU read access to a mapped resource.

Remarks

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

WriteOnly CPU write access to a mapped resource.

Remarks

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

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

Remarks

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

Note

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

ReadWrite CPU read and write access to a mapped resource.

Remarks

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

9.1.2.9 enum LLGL::CullMode [strong]

Polygon culling modes enumeration.

See also

RasterizerDescriptor::cullMode

Enumerator

Disabled No culling.

Front Front face culling.

Back Back face culling.

9.1.2.10 enum LLGL::DataType [strong]

Data types enumeration used for CPU side operations.

Remarks

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

See also

SrcImageDescriptor::dataType
DstImageDescriptor::dataType

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

Enumerator

Undefined Undefined data type.

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

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

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

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

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

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

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

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

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

9.1.2.11 enum LLGL::ErrorType [strong]

Rendering debugger error types enumeration.

Enumerator

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

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

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

UndefinedBehavior Error due to arguments that cause undefined behavior.

9.1.2.12 enum LLGL::Format [strong]

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

Remarks

The counterpart of this enumeration for CPU side operations are LLGL::ImageFormat and LLGL::DataType.
Packed color formats are stored from least significant bit (LSB) to most significant bit (MSB), i.e. for the Format::RGB10A2UNorm the bit pattern is `A[31:30], B[29:20], G[19:10], R[9:0]` for instance.
The following table illustrates the bit-size for each component of all ordinary and packed color formats (depth-stencil and compressed formats excluded) with the following notation:

- **`n`** denotes unsigned normalized formats (`UNorm`)
- **`s`** denotes signed normalized formats (`SNorm`)
- **`u`** denotes unsigned integer formats (`UInt`)
- **`i`** denotes signed integer formats (`SInt`)
- **`f`** denotes floating-point formats (`Float`)

Format	Bits	Red	Green	Blue	Alpha	Shared
Format::A8UNorm	8				n8	
Format::R8UNorm	8	n8				
Format::R8SNorm	8	s8				
Format::R8UInt	8	u8				
Format::R8SInt	8	i8				
Format::R16UNorm	16	n16				
Format::R16SNorm	16	s16				
Format::R16UInt	16	u16				
Format::R16SInt	16	i16				
Format::R16Float	16	f16				
Format::R32UInt	32	u32				
Format::R32SInt	32	i32				
Format::R32Float	32	f32				
Format::R64Float	64	f64				
Format::RG8UNorm	16	n8	n8			
Format::RG8SNorm	16	s8	s8			
Format::RG8UInt	16	u8	u8			
Format::RG8SInt	16	i8	i8			
Format::RG16UNorm	32	n16	n16			

Format	Bits	Red	Green	Blue	Alpha	Shared
Format::RG16SNorm	32	s16	s16			
Format::RG16UInt	32	u16	u16			
Format::RG16SInt	32	i16	i16			
Format::RG16Float	32	f16	f16			
Format::RG32UInt	64	u32	u32			
Format::RG32SInt	64	i32	i32			
Format::RG32Float	64	f32	f32			
Format::RG64Float	128	f64	f64			
Format::RGB8UNorm	24	n8	n8	n8		
Format::RGB8UNorm_sRGB	24	n8	n8	n8		
Format::RGB8SNorm	24	s8	s8	s8		
Format::RGB8UInt	24	u8	u8	u8		
Format::RGB8SInt	24	i8	i8	i8		
Format::RGB16UNorm	48	n16	n16	n16		
Format::RGB16SNorm	48	s16	s16	s16		
Format::RGB16UInt	48	u16	u16	u16		
Format::RGB16SInt	48	i16	i16	i16		
Format::RGB16Float	48	f16	f16	f16		
Format::RGB32UInt	96	u32	u32	u32		
Format::RGB32SInt	96	i32	i32	i32		
Format::RGB32Float	96	f32	f32	f32		
Format::RGB64Float	192	f64	f64	f64		
Format::RGBA8UNorm	32	n8	n8	n8	n8	
Format::RGBA8UNorm_sRGB	32	n8	n8	n8	n8	
Format::RGBA8SNorm	32	s8	s8	s8	s8	
Format::RGBA8UInt	32	u8	u8	u8	u8	
Format::RGBA8SInt	32	i8	i8	i8	i8	
Format::RGBA16UNorm	64	n16	n16	n16	n16	
Format::RGBA16SNorm	64	s16	s16	s16	s16	
Format::RGBA16UInt	64	u16	u16	u16	u16	
Format::RGBA16SInt	64	i16	i16	i16	i16	
Format::RGBA16Float	64	f16	f16	f16	f16	
Format::RGBA32UInt	128	u32	u32	u32	u32	
Format::RGBA32SInt	128	i32	i32	i32	i32	
Format::RGBA32Float	128	f32	f32	f32	f32	
Format::RGBA64Float	256	f64	f64	f64	f64	
Format::BGRA8UNorm	32	n8	n8	n8	n8	
Format::BGRA8UNorm_sRGB	32	n8	n8	n8	n8	
Format::BGRA8SNorm	32	s8	s8	s8	s8	
Format::BGRA8UInt	32	u8	u8	u8	u8	
Format::BGRA8SInt	32	i8	i8	i8	i8	
Format::RGB10A2UNorm	32	n10	n10	n10	n2	
Format::RGB10A2UInt	32	u10	u10	u10	u2	
Format::RG11B10Float	32	f11	f11	f10		
Format::RGB9E5Float	32	9	9	9		5

See also

TextureDescriptor::format
VertexAttribute::format
RenderingCapabilities::textureFormats
OpenGL counterpart: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexImage1D.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 for textures `MTLPixelFormat`: https://developer.apple.com/documentation/metal/m
Metal counterpart for vertices `MTLVertexFormat`: https://developer.apple.com/documentation/metal/

Enumerator

Undefined Undefined format.

A8UNorm Alpha channel format: alpha 8-bit normalized unsigned integer component.

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

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

R8UInt Ordinary color format: red 8-bit unsigned integer component.

R8SInt Ordinary color format: red 8-bit signed integer component.

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

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

R16UInt Ordinary color format: red 16-bit unsigned interger component.

R16SInt Ordinary color format: red 16-bit signed interger component.

R16Float Ordinary color format: red 16-bit floating point component.

R32UInt Ordinary color format: red 32-bit unsigned interger component.

R32SInt Ordinary color format: red 32-bit signed interger component.

R32Float Ordinary color format: red 32-bit floating point component.

R64Float Ordinary color format: red 64-bit floating point component.

Note

Only supported with: Vulkan.

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

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

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

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

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

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

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

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

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

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

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

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

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

Note

Only supported with: Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

RGB8UNorm_sRGB Ordinary color format: red, green, blue 8-bit normalized unsigned integer components in non-linear sRGB color space.

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

Only supported with: OpenGL, Vulkan.

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

Note

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

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

Note

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

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

Note

As texture format only supported with: OpenGL, Vulkan, Direct3D 11, Direct3D 12.

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

Note

Only supported with: Vulkan.

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

RGBA8UNorm_sRGB Ordinary color format: red, green, blue, alpha 8-bit normalized unsigned integer components in non-linear sRGB color space.

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

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

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

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

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

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

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

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

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

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

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

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

Note

Only supported with: Vulkan.

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

BGRA8UNorm_sRGB Ordinary color format: blue, green, red, alpha 8-bit normalized unsigned integer components in non-linear sRGB color space.

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

Note

Only supported with: Vulkan.

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

Note

Only supported with: Vulkan.

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

Note

Only supported with: Vulkan.

RGB10A2UNorm Packed color format: red, green, blue 10-bit and alpha 2-bit normalized unsigned integer components.

RGB10A2UInt Packed color format: red, green, blue 10-bit and alpha 2-bit unsigned integer components.

RG11B10Float Packed color format: red, green 11-bit and blue 10-bit unsigned floating point, i.e. 6-bit mantissa for red and green, 5-bit mantissa for blue, and 5-bit exponent for all components.

RGB9E5Float Packed color format: red, green, blue 9-bit unsigned floating-point with shared 5-bit exponent, i.e. 9-bit mantissa for each component and one 5-bit exponent for all components.

D16UNorm Depth-stencil format: depth 16-bit normalized unsigned integer component.

D24UNormS8UInt Depth-stencil format: depth 24-bit normalized unsigned integer component, and 8-bit unsigned integer stencil component.

D32Float Depth-stencil format: depth 32-bit floating point component.

D32FloatS8X24UInt Depth-stencil format: depth 32-bit floating point component, and 8-bit unsigned integer stencil components (where the remaining 24 bits are unused).

BC1UNorm Compressed color format: S3TC BC1 compressed RGBA with normalized unsigned integer components in 64-bit per 4x4 block.

BC1UNorm_sRGB Compressed color format: S3TC BC1 compressed RGBA with normalized unsigned integer components in 64-bit per 4x4 block in non-linear sRGB color space.

BC2UNorm Compressed color format: S3TC BC2 compressed RGBA with normalized unsigned integer components in 128-bit per 4x4 block.

BC2UNorm_sRGB Compressed color format: S3TC BC2 compressed RGBA with normalized unsigned integer components in 128-bit per 4x4 block in non-linear sRGB color space.

BC3UNorm Compressed color format: S3TC BC3 compressed RGBA with normalized unsigned integer components in 128-bit per 4x4 block.

BC3UNorm_sRGB Compressed color format: S3TC BC3 compressed RGBA with normalized unsigned integer components in 128-bit per 4x4 block in non-linear sRGB color space.

BC4UNorm Compressed color format: S3TC BC4 compressed red channel with normalized unsigned integer component in 64-bit per 4x4 block.

BC4SNorm Compressed color format: S3TC BC4 compressed red channel with normalized signed integer component 64-bit per 4x4 block.

BC5UNorm Compressed color format: S3TC BC5 compressed red and green channels with normalized unsigned integer components in 64-bit per 4x4 block.

BC5SNorm Compressed color format: S3TC BC5 compressed red and green channels with normalized signed integer components in 128-bit per 4x4 block.

9.1.2.13 enum LLGL::ImageFormat [strong]

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

See also

SrcImageDescriptor::format
DstImageDescriptor::format
ImageFormatSize

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

Enumerator

Alpha Single color component: Alpha.

R Single color component: Red.

RG Two color components: Red, Green.

RGB Three color components: Red, Green, Blue.

BGR Three color components: Blue, Green, Red.

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

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

ARGB Four color components: Alpha, Red, Green, Blue. Legacy format, mainly used in Direct3D 9.

ABGR Four color components: Alpha, Blue, Green, Red. Legacy format, mainly used in Direct3D 9.

Depth Depth component.

DepthStencil Depth component and stencil index.

BC1 Block compression BC1.

BC2 Block compression BC2.

BC3 Block compression BC3.

BC4 Block compression BC4.

BC5 Block compression BC5.

9.1.2.14 enum LLGL::Key [strong]

Input key codes.

See also

Window::EventListener

Enumerator

LButton Left mouse button.

RButton Right mouse button.

Cancel Control-break processing.

MButton Middle mouse button (three-button mouse).

XButton1 Windows 2000/XP: X1 mouse button.

XButton2 Windows 2000/XP: X2 mouse button.

Back BACKSPACE key.

Tab TAB key.

Clear CLEAR key.

Return RETURN (or ENTER) key.

Shift SHIFT key.

Control CTRL key.

Menu ALT key.

Pause PAUSE key.

Capital CAPS LOCK key.

Escape Escape (ESC) key.

Space Space key.

PageUp Page up key.

PageDown Page down key.

End END key.

Home HOME (or POS1) key.

Left Left arrow key.

Up Up arrow key.

Right Right arrow key.

Down Down arrow key.

Select Select key.

Print Print key.

Exe Execute key.

Snapshot Snapshot key.

Insert Insert key.

Delete Delete key.

Help Help key.

D0 Digit 0.

D1 Digit 1.

D2 Digit 2.

D3 Digit 3.

D4 Digit 4.

D5 Digit 5.

D6 Digit 6.

D7 Digit 7.

D8 Digit 8.

D9 Digit 9.

A Letter A.

B Letter B.

C Letter C.

D Letter D.

E Letter E.

F Letter F.

G Letter G.

H Letter H.

I Letter I.

J Letter J.

K Letter K.

L Letter L.

M Letter M.

N Letter N.

O Letter O.

P Letter P.

Q Letter Q.

R Letter R.

S Letter S.

T Letter T.

U Letter U.

V Letter V.

W Letter W.

X Letter X.

Y Letter Y.

Z Letter Z.

LWin Left Windows key.

RWin Rigth Windows key.

Apps Application key.

Sleep Sleep key.

Keypad0 Keypad 0 key.

Keypad1 Keypad 1 key.

Keypad2 Keypad 2 key.

Keypad3 Keypad 3 key.

Keypad4 Keypad 4 key.

Keypad5 Keypad 5 key.

Keypad6 Keypad 6 key.

Keypad7 Keypad 7 key.

Keypad8 Keypad 8 key.

Keypad9 Keypad 9 key.

KeypadMultiply Keypad multiply '*'.

KeypadPlus Keypad plus '+'.

KeypadSeparator Keypad separator.

KeypadMinus Keypad minus '-'.

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

KeypadDivide Keypad divide '/'.

F1 F1 function key.

F2 F2 function key.

F3 F3 function key.

F4 F4 function key.

F5 F5 function key.

F6 F6 function key.

F7 F7 function key.

F8 F8 function key.

F9 F9 function key.

F10 F10 function key.

F11 F11 function key.

F12 F12 function key.

F13 F13 function key.

F14 F14 function key.

F15 F15 function key.

F16 F16 function key.

F17 F17 function key.

F18 F18 function key.

F19 F19 function key.

F20 F20 function key.

F21 F21 function key.

F22 F22 function key.

F23 F23 function key.

F24 F24 function key.

NumLock Num lock key.

ScrollLock Scroll lock key.

LShift Left shift key.

RShift Right shift key.

LControl Left control (CTRL) key.

RControl Right control (CTRL) key.

LMenu Left menu key.

RMenu Right menu key.

BrowserBack

BrowserForward

BrowserRefresh

BrowserStop

BrowserSearch

BrowserFavorits

BrowserHome

VolumeMute

VolumeDown

VolumeUp

MediaNextTrack

MediaPrevTrack

MediaStop

MediaPlayPause

LaunchMail

LaunchMediaSelect

LaunchApp1

LaunchApp2

Plus '+'

Comma ','

Minus '-'

Period '.'

Exponent '^'

Attn

CrSel

ExSel

ErEOF

Play

Zoom

NoName

PA1

OEMClear

Any Refers to any key.

9.1.2.15 enum LLGL::LogicOp [strong]

Logical pixel operation enumeration.

Remarks

These logical pixel operations are bitwise operations. In the following documentation, `src` denotes the source color and `dst` denotes the destination color.

Note

Only supported with: OpenGL, Vulkan, Direct3D 11.1+, Direct3D 12.0.

See also

BlendDescriptor::logicOp

Enumerator

Disabled No logical pixel operation.

Clear Resulting operation: `0`.

Set Resulting operation: `1`.

Copy Resulting operation: `src`.

CopyInverted Resulting operation: `~src`.

NoOp Resulting operation: `dst`.

Invert Resulting operation: `~dst`.

AND Resulting operation: `src & dst`.

ANDReverse Resulting operation: `src & ~dst`.

ANDInverted Resulting operation: `~src & dst`.

NAND Resulting operation: `~(src & dst)`.

OR Resulting operation: `src | dst`.

ORReverse Resulting operation: `src | ~dst`.

ORInverted Resulting operation: `~src | dst`.

NOR Resulting operation: `~(src | dst)`.

XOR Resulting operation: `src ^ dst`.

Equiv Resulting operation: `~(src ^ dst)`.

9.1.2.16 enum LLGL::OpenGLContextProfile [strong]

OpenGL context profile enumeration.

See also

RendererConfigurationOpenGL::contextProfile

Enumerator

CompatibilityProfile OpenGL compatibility profile.

CoreProfile OpenGL core profile.

ESProfile OpenGL ES profile.

Todo Not supported yet

9.1.2.17 enum LLGL::PipelineBindPoint [strong]

Pipeline binding point enumeration.

See also

CommandBuffer::SetResourceHeap

Enumerator

Undefined Automatically determine pipeline binding point for a resource heap.

Graphics Graphics pipeline binding point.

Compute Compute pipeline binding point.

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.

Remarks

All strip topologies (i.e. PrimitiveTopology::LineStrip, PrimitiveTopology::LineStripAdjacency, PrimitiveTopology::TriangleStrip, and PrimitiveTopology::TriangleStripAdjacency) use a fixed index value to restart the primitives. This fixed index value is the maximum possible value for the respective index buffer format, i.e. `2^16-1` (or `0xFFFF`) for Format::R16UInt and `2^32-1` (or `0xFFFFFFFF`) for Format::R32UInt.

Todo For Direct3D 12, the restart primitive index value (aka. strip cut value) can only be `0xFFFFFFFF` at the moment as this needs to be specified at PSO creation time.

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.

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.

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::QueryType [strong]

Query type enumeration.

See also

QueryHeapDescriptor::type

Enumerator

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

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

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

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

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

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

PipelineStatistics Pipeline statistics such as number of shader invocations, generated primitives, etc.

See also

QueryPipelineStatistics
RenderingFeatures::hasPipelineStatistics

9.1.2.21 enum LLGL::RenderConditionMode [strong]

Render condition mode enumeration.

Remarks

The condition is determined by the type of the QueryHeap object.

See also

CommandBuffer::BeginRenderCondition

Enumerator

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

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

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

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

WaitInverted Same as Wait, but the condition is inverted.

NoWaitInverted Same as NoWait, but the condition is inverted.

ByRegionWaitInverted Same as ByRegionWait, but the condition is inverted.

ByRegionNoWaitInverted Same as ByRegionNoWait, but the condition is inverted.

9.1.2.22 enum LLGL::ResourceType [strong]

Hardware resource type enumeration.

Remarks

This is primarily used to describe the source type for a layout binding (see BindingDescriptor), which is why all buffer types are enumerated but not the texture types.

See also

BindingDescriptor::type

Enumerator

Undefined Undefined resource type.

Buffer Buffer resource.

See also

Buffer
RenderSystem::CreateBuffer

Texture Texture resource.

See also

Texture
TextureType
RenderSystem::CreateTexture

Sampler Sampler state resource.

See also

Sampler
RenderSystem::CreateSampler

9.1.2.23 enum LLGL::SamplerAddressMode [strong]

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

See also

SamplerDescriptor::addressModeU
SamplerDescriptor::addressModeV
SamplerDescriptor::addressModeW

Enumerator

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

Figure 9.1 SamplerAddressMode::Repeat example

Mirror Flip texture coordinates at each integer junction.

Figure 9.2 SamplerAddressMode::Mirror example

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

Figure 9.3 SamplerAddressMode::Clamp example

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

Figure 9.4 SamplerAddressMode::Border example

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

Figure 9.5 SamplerAddressMode::MirrorOnce example

9.1.2.24 enum LLGL::SamplerFilter [strong]

Sampling filter enumeration.

See also

SamplerDescriptor::minFilter
SamplerDescriptor::magFilter
SamplerDescriptor::mipMapFilter
Image::Resize(const Extent3D&, const SamplerFilter)

Enumerator

Nearest Take the nearest texture sample.

Figure 9.6 SamplerFilter::Nearest example

Linear Interpolate between multiple texture samples.

Figure 9.7 SamplerFilter::Linear example

9.1.2.25 enum LLGL::ScreenOrigin [strong]

Screen coordinate system origin enumeration.

See also

RenderingCapabilities::screenOrigin

Enumerator

LowerLeft Specifies a screen origin in the lower-left.

Note

Native screen origin in: OpenGL (If `GL_ARB_clip_control` is *not* supported).

UpperLeft Specifies a screen origin in the upper-left.

Note

Native screen origin in: Direct3D 11, Direct3D 12, Vulkan, OpenGL (If `GL_ARB_clip_control` *is* supported), Metal.

9.1.2.26 enum LLGL::ShaderSourceType [strong]

Shader source type enumeration.

See also

ShaderDescriptor::sourceType
ShaderDescriptor::sourceSize

Enumerator

CodeString Refers to `sourceSize+1` bytes, describing shader high-level code (including null terminator).

CodeFile Refers to `sourceSize+1` bytes, describing the filename of the shader high-level code (including null terminator).

BinaryBuffer Refers to `sourceSize` bytes, describing shader binary code.

BinaryFile Refers to `sourceSize+1` bytes, describing the filename of the shader binary code (including null terminator).

9.1.2.27 enum LLGL::ShaderType [strong]

Shader type enumeration.

See also

ShaderDescriptor::type

Enumerator

Undefined Undefined shader type.

Vertex Vertex shader type.

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

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

Geometry Geometry shader type.

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

Compute Compute shader type.

9.1.2.28 enum LLGL::ShadingLanguage [strong]

Shading language version enumation.

Remarks

These enumeration entries can be casted to an integer using the bitmask ShadingLanguage::VersionBitmask to get the respective version number:

```
// 'versionNo' will have the value 330
static const auto versionGLSL330 = static_cast<std::uint32_t>(
      LLGL::ShadingLanguage::GLSL_330);
static const auto versionBitmask = static_cast<std::uint32_t>(
      LLGL::ShadingLanguage::VersionBitmask);
static const auto versionNo      = versionGLSL330 & versionBitmask;
```

Enumerator

GLSL GLSL (OpenGL Shading Language).

GLSL_110 GLSL 1.10 (since OpenGL 2.0).

GLSL_120 GLSL 1.20 (since OpenGL 2.1).

GLSL_130 GLSL 1.30 (since OpenGL 3.0).

GLSL_140 GLSL 1.40 (since OpenGL 3.1).

GLSL_150 GLSL 1.50 (since OpenGL 3.2).

GLSL_330 GLSL 3.30 (since OpenGL 3.3).

GLSL_400 GLSL 4.00 (since OpenGL 4.0).

GLSL_410 GLSL 4.10 (since OpenGL 4.1).

GLSL_420 GLSL 4.20 (since OpenGL 4.2).

GLSL_430 GLSL 4.30 (since OpenGL 4.3).

GLSL_440 GLSL 4.40 (since OpenGL 4.4).

GLSL_450 GLSL 4.50 (since OpenGL 4.5).

GLSL_460 GLSL 4.60 (since OpenGL 4.6).

ESSL ESSL (OpenGL ES Shading Language).

ESSL_100 ESSL 1.00 (since OpenGL ES 2.0).

ESSL_300 ESSL 3.00 (since OpenGL ES 3.0).

ESSL_310 ESSL 3.10 (since OpenGL ES 3.1).

ESSL_320 ESSL 3.20 (since OpenGL ES 3.2).

HLSL HLSL (High Level Shading Language).

HLSL_2_0 HLSL 2.0 (since Direct3D 9).

HLSL_2_0a HLSL 2.0a (since Direct3D 9a).

HLSL_2_0b HLSL 2.0b (since Direct3D 9b).

HLSL_3_0 HLSL 3.0 (since Direct3D 9c).

HLSL_4_0 HLSL 4.0 (since Direct3D 10).

HLSL_4_1 HLSL 4.1 (since Direct3D 10.1).

HLSL_5_0 HLSL 5.0 (since Direct3D 11).

HLSL_5_1 HLSL 5.1 (since Direct3D 11.3).

HLSL_6_0 HLSL 6.0 (since Direct3D 12). Shader model 6.0 adds wave intrinsics and 64-bit integer types to HLSL.

HLSL_6_1 HLSL 6.1 (since Direct3D 12). Shader model 6.1 adds `SV_ViewID` and `SV_Barycentrics` semantics to HLSL.

HLSL_6_2 HLSL 6.2 (since Direct3D 12). Shader model 6.2 adds 16-bit scalar types to HLSL.

HLSL_6_3 HLSL 6.3 (since Direct3D 12). Shader model 6.3 adds ray tracing (DXR) to HLSL.

HLSL_6_4 HLSL 6.4 (since Direct3D 12). Shader model 6.4 adds machine learning intrinsics to HLSL.

Metal Metal Shading Language.

Metal_1_0 Metal 1.0 (since iOS 8.0).

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

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

Metal_2_0 Metal 2.0 (since iOS 11.0 and macOS 10.13).

Metal_2_1 Metal 2.1 (since iOS 12.0 and macOS 10.14).

SPIRV SPIR-V Shading Language.

SPIRV_100 SPIR-V 1.0.

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

9.1.2.29 enum LLGL::StencilFace [strong]

Stencil face enumeration.

Remarks

To be compatible with Direct3D renderers, it is recommended to always use `FrontAndBack`.

See also

CommandBuffer::SetStencilReference

Enumerator

FrontAndBack Refers to both the front and back face of primitives.

Front Refers only to the front face of primitives.

Note

Only supported with: OpenGL, Vulkan, Metal.

Back Refers only to the back face of primitives.

Note

Only supported with: OpenGL, Vulkan, Metal.

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 for shader reflection.

Note

Only supported with: Direct3D 11, Direct3D 12.

See also

ShaderResource::storageBufferType

Enumerator

Undefined Undefined storage buffer type.

TypedBuffer Typed buffer, e.g. `Buffer<float4>` in HLSL.

See also

BufferDescriptor::format

StructuredBuffer Structured buffer, e.g. `StructuredBuffer<MyStruct>` in HLSL.

See also

BufferDescriptor::stride

ByteAddressBuffer Byte-address buffer, e.g. `ByteAddressBuffer` in HLSL.

See also

BufferDescriptor::format

RWTypedBuffer Typed read/write buffer, e.g. `RWBuffer<float4>` in HLSL.

See also

BufferDescriptor::format

RWStructuredBuffer Read/write structured buffer, e.g. `RWStructuredBuffer<MyStruct>` in HLSL.

See also

BufferDescriptor::stride

RWByteAddressBuffer Read/write byte-address buffer, e.g. `RWByteAddressBuffer` in HLSL.

See also

BufferDescriptor::stride

AppendStructuredBuffer Append structured buffer, e.g. `AppendStructuredBuffer<MyStruct>` in HLSL.

See also

BufferDescriptor::stride
MiscFlags::Append

ConsumeStructuredBuffer Consume structured buffer, e.g. `ConsumeStructuredBuffer<MyStruct>` in HLSL.

See also

BufferDescriptor::stride
MiscFlags::Append

9.1.2.32 enum LLGL::SystemValue [strong]

Enumeration of system values for vertex input or fragment output attributes.

Remarks

This is only used for shader code reflection.
Here is an overview of all supported system values and their mapping to the respective shading language:

Table 9.2 System value mapping

Type	HLSL	GLSL	SPIR-V	Metal
SystemValue::ClipDistance	SV_ClipDistance	gl_ClipDistance	gl_ClipDistance	[[clip_distance]]
SystemValue::Color	SV_Target	N/A	N/A	[[color(N)]]
SystemValue::CullDistance	SV_CullDistance	gl_CullDistance	gl_CullDistance	N/A
SystemValue::Depth	SV_Depth	gl_FragDepth	gl_FragDepth	[[depth(any)]]
SystemValue::DepthGreater	SV_DepthGreaterEqual	N/A	N/A	[[depth(greater)]]
SystemValue::DepthLess	SV_DepthLessEqual	N/A	N/A	[[depth(less)]]
SystemValue::FrontFacing	SV_IsFrontFace	gl_FrontFacing	gl_FrontFacing	[[front_facing]]
SystemValue::InstanceID	SV_InstanceID	gl_InstanceID	gl_InstanceIndex	[[instance_id]

Type	HLSL	GLSL	SPIR-V	Metal
SystemValue::↩ Position	SV_Position	gl_Position, gl_FragCoord	gl_Position, gl_FragCoord	[[position]]
SystemValue::↩ PrimitiveID	SV_↩ PrimitiveID	gl_↩ PrimitiveID	gl_↩ PrimitiveID	N/A
SystemValue::↩ RenderTarget↩ Index	SV_Render↩ Target↩ ArrayIndex	gl_Layer	gl_Layer	[[render↩ _target↩ _array_↩ index]]
SystemValue::↩ SampleMask	SV_Coverage	gl_Sample↩ Mask	gl_Sample↩ Mask	[[sample_↩ mask]]
SystemValue::↩ SampleID	SV_Sample↩ Index	gl_SampleID	gl_SampleID	[[sample_↩ id]]
SystemValue::↩ Stencil	SV_Stencil↩ Ref	N/A	N/A	[[stencil]]
SystemValue::↩ VertexID	SV_VertexID	gl_VertexID	gl_Vertex↩ Index	[[vertex_↩ id]]
SystemValue::↩ ViewportIndex	SV_↩ Viewport↩ ArrayIndex	gl_↩ Viewport↩ Index	gl_↩ Viewport↩ Index	[[viewport↩ _array_↩ index]]

See also

VertexAttribute::systemValue
FragmentAttribute::systemValue

Enumerator

Undefined Undefined system value.

ClipDistance Forward-compatible mechanism for vertex clipping.

Color Fragment output color value.

CullDistance Mechanism for controlling user culling.

Depth Fragment depth value.

DepthGreater Fragment depth value that is greater than or equal to the previous one.

DepthLess Fragment depth value that is less than or equal to the previous one.

FrontFacing Indicates whether a primitive is front or back facing.

InstanceID Index of the input instance.

Note

This value behalves differently between Direct3D and OpenGL.

See also

CommandBuffer::DrawInstanced(std::uint32_t, std::uint32_t, std::uint32_t, std::uint32_t)

Position Vertex or fragment position.

PrimitiveID Index of the geometry primitive.

RenderTargetIndex Index of the render target layer.

SampleMask Sample coverage mask.

SampleID Index of the input sample.

Stencil Fragment stencil value.

Note

Only supported with: Direct3D 11.3, Direct3D 12, Metal.

VertexID Index of the input vertex.

Note

This value behalves differently between Direct3D and OpenGL.

See also

CommandBuffer::Draw

ViewportIndex Index of the viewport array.

9.1.2.33 enum LLGL::TessellationPartition [strong]

Tessellation partition mode enumeration.

See also

TessellationDescriptor::partition

Enumerator

Undefined Undefined partition mode.

Integer Integer with integers only.

Remarks

Equivalent of `[partitioning("integer")]` in HLSL and `layout(equal_spacing)` in GLSL.

Pow2 Partition with power-of-two number only.

Remarks

Equivalent of `[partitioning("pow2")]` in HLSL.

FractionalOdd Partition with an odd, fractional number.

Remarks

Equivalent of `[partitioning("fractional_odd")]` in HLSL and `layout(fractional↩_odd_spacing)` in GLSL.

FractionalEven Partition with an even, fractional number.

Remarks

Equivalent of `[partitioning("fractional_even")]` in HLSL and `layout(fractional↩_even_spacing)` in GLSL.

9.1.2.34 enum LLGL::TextureSwizzle : std::uint8_t [strong]

Texture component swizzle enumeration.

Remarks

Can be used to change the order of texel components independently of a shader.

See also

TextureSwizzleRGBA

Enumerator

Zero The component is replaced by the constant zero.

One The component is replaced by the constant one.

Red The component is replaced by red component.

Green The component is replaced by green component.

Blue The component is replaced by blue component.

Alpha The component is replaced by alpha component.

9.1.2.35 enum LLGL::TextureType [strong]

Texture type enumeration.

Enumerator

Texture1D 1-Dimensional texture.

Note

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

Texture2D 2-Dimensional texture.

Texture3D 3-Dimensional texture.

TextureCube Cube texture.

See also

TextureDescriptor::arrayLayers

Texture1DArray 1-Dimensional texture.

Note

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

See also

TextureDescriptor::arrayLayers

Texture2DArray 2-Dimensional array texture.

See also

TextureDescriptor::arrayLayers

TextureCubeArray Cube array texture.

See also

TextureDescriptor::arrayLayers

Note

Only supported with: OpenGL, OpenGLES 3.2, Vulkan, Direct3D 11, Direct3D 12, Metal.

Texture2DMS 2-Dimensional multi-sample texture.

Note

Only supported with: OpenGL, OpenGLES 3.1, Vulkan, Direct3D 11, Direct3D 12, Metal.

See also

TextureDescriptor::samples

Texture2DMSArray 2-Dimensional multi-sample array texture.

Note

Only supported with: OpenGL, OpenGLES 3.2, Vulkan, Direct3D 11, Direct3D 12, Metal.

See also

TextureDescriptor::samples

9.1.2.36 enum LLGL::UniformType [strong]

Shader uniform type enumeration.

Remarks

Because "Bool" is a reserved identifier for an Xlib macro on GNU/Linux, all scalar types also have a component index (e.g. "Bool1" instead of "Bool").

Enumerator

Undefined Undefined uniform type.

Float1 float uniform.

Float2 float2/ vec2 uniform.

Float3 float3/ vec3 uniform.

Float4 float4/ vec4 uniform.

Double1 double uniform.

Double2 double2/ dvec2 uniform.

Double3 double3/ dvec3 uniform.

Double4 double4/ dvec4 uniform.

Int1 int uniform.

Int2 int2/ ivec2 uniform.

Int3 int3/ ivec3 uniform.

Int4 int4/ ivec4 uniform.

UInt1 uint uniform.

UInt2 uint2/ uvec2 uniform.

UInt3 uint3/ uvec3 uniform.

UInt4 uint4/ uvec4 uniform.

Bool1 bool uniform.

Bool2 bool2/ bvec2 uniform.

Bool3 bool3/ bvec3 uniform.

Bool4 bool4/ bvec4 uniform.

Float2x2 float2x2/ mat2 uniform.

Float2x3 float2x3/ mat2x3 uniform.

Float2x4 float2x4/ mat2x4 uniform.

Float3x2 float3x2/ mat3x2 uniform.

Float3x3 float3x3/ mat3 uniform.

Float3x4 float3x4/ mat3x4 uniform.

Float4x2 float4x2/ mat4x2 uniform.

Float4x3 float4x3/ mat4x3 uniform.

Float4x4 float4x4/ mat4 uniform.

Double2x2 double2x2/ dmat2 uniform.

Double2x3 double2x3/ dmat2x3 uniform.

Double2x4 double2x4/ dmat2x4 uniform.

Double3x2 double3x2/ dmat3x2 uniform.

Double3x3 double3x3/ dmat3 uniform.

Double3x4 double3x4/ dmat3x4 uniform.

Double4x2 double4x2/ dmat4x2 uniform.

Double4x3 double4x3/ dmat4x3 uniform.

Double4x4 double4x4/ dmat4 uniform.

Sampler Sampler uniform (e.g. "sampler2D").

Image Image uniform (e.g. "image2D").

AtomicCounter Atomic counter uniform (e.g. "atomic_uint").

9.1.2.37 enum LLGL::WarningType [strong]

Rendering debugger warning types enumeration.

Enumerator

ImproperArgument Warning due to improper argument (e.g. generating 4 vertices while having triangle list as primitive topology).

ImproperState Warning due to improper state (e.g. rendering while viewport is not visible).

PointlessOperation Warning due to a operation without any effect (e.g. drawing with 0 vertices).

VaryingBehavior Warning due to a varying behavior between the native APIs (e.g. `SV_VertexID` in HLSL behaves different to `gl_VertexID` in GLSL or `gl_VertexIndex` in SPIRV).

9.1.3 Function Documentation

9.1.3.1 template<typename Dst , typename Src > Dst LLGL::CastColorValue (const Src & *value*) [inline]

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

See also

MaxColorValue

9.1.3.2 template<> bool LLGL::CastColorValue< bool, bool > (const bool & *value*) [inline]

Specialized template which merely passes the input value as output.

9.1.3.3 template<> double LLGL::CastColorValue< double, double > (const double & *value*) [inline]

Specialized template which merely passes the input value as output.

9.1.3.4 template<> float LLGL::CastColorValue< float, float > (const float & *value*) [inline]

Specialized template which merely passes the input value as output.

9.1.3.5 template<> std::uint8_t LLGL::CastColorValue< std::uint8_t, std::uint8_t > (const std::uint8_t & *value*) [inline]

Specialized template which merely passes the input value as output.

9.1.3.6 template<typename T > const T* LLGL::CastTo (const Interface * *obj*) [inline]

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

Remarks

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
const LLGL::Surface* mySurface = ...
if (const LLGL::Window* myWindow = LLGL::CastTo<LLGL::Window>(mySurface))
{
    ...
}
```

9.1.3.7 template<typename T > T* LLGL::CastTo (Interface * *obj*) [inline]

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

Remarks

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
LLGL::Surface* mySurface = ...
if (LLGL::Window* myWindow = LLGL::CastTo<LLGL::Window>(mySurface))
{
    ...
}
```

9.1.3.8 template<typename T > const T& LLGL::CastTo (const Interface & *obj*) [inline]

Returns a constant reference to this instance of type `const T&` if it implements the specified interface.

Remarks

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
const LLGL::Surface& mySurface = ...
const LLGL::Window& myWindow = LLGL::CastTo<LLGL::Window>(mySurface);
```

9.1.3.9 template<typename T > T& LLGL::CastTo (Interface & *obj*) [inline]

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

Remarks

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

This template can be used like in the following example:

```
LLGL::Surface& mySurface = ...
LLGL::Window& myWindow = LLGL::CastTo<LLGL::Window&>(mySurface);
```

9.1.3.10 LLGL_EXPORT Extent2D LLGL::GetExtentRatio (const Extent2D & *extent*)

Returns the ratio of the specified extent as another extent, i.e. all attributes are divided by their greatest common divisor.

Remarks

This can be used to print out a display mode resolution in a better format (e.g. "16:9" rather than "1920:1080").

See also

DisplayModeDescriptor::resolution

9.1.3.11 LLGL_EXPORT std::uint32_t LLGL::GetPrimitiveTopologyPatchSize (const PrimitiveTopology *primitiveTopology*)

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

9.1.3.12 template<typename T > bool LLGL::IsInstanceOf (const Interface * *obj*) [inline]

Returns whether the specified object is an instance of the interface specified by template parameter <T>.

Remarks

The typename `T` must be a sub class of Interface and implement the following function:

```
static InterfaceID GetInterfaceID();
```

See also

Interface::IsInstanceOf

9.1.3.13 template<typename T > bool LLGL::IsInstanceOf (const Interface & *obj*) [inline]

Const-reference version of IsInstanceOf.

See also

IsInstanceOf(const Interface*)

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

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

9.1.3.15 LLGL_EXPORT bool LLGL::IsPrimitiveTopologyStrip (const PrimitiveTopology *primitiveTopology*)

Returns true if the specified primitive topology is a strip that generates a new primitive with each new vertex.

Returns

True if `primitiveTopology` is equal to one of the following primitive topologies:

- PrimitiveTopology::LineStrip
- PrimitiveTopology::LineStripAdjacency
- PrimitiveTopology::TriangleStrip
- PrimitiveTopology::TriangleStripAdjacency

9.1.3.16 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

9.1.3.17 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

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

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

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

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

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

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

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

Compares the two specified display mode descriptors on inequality.

9.1.3.22 bool LLGL::operator!= (const UTF8String & *lhs*, const UTF8String & *rhs*) [inline]

Returns true if the two strings are not equal.

9.1.3.23 bool LLGL::operator!= (const UTF8String & *lhs*, const StringView & *rhs*) [inline]

9.1.3.24 bool LLGL::operator!= (const UTF8String & *lhs*, const WStringView & *rhs*) [inline]

9.1.3.25 bool LLGL::operator!= (const UTF8String & *lhs*, const char * *rhs*) [inline]

9.1.3.26 bool LLGL::operator!= (const UTF8String & *lhs*, const wchar_t * *rhs*) [inline]

9.1.3.27 bool LLGL::operator!= (const StringView & *lhs*, const StringView & *rhs*) [inline]

9.1.3.28 bool LLGL::operator!= (const StringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.29 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).

9.1.3.30 bool LLGL::operator!= (const StringView & *lhs*, const char * *rhs*) [inline]

9.1.3.31 bool LLGL::operator!= (const WStringView & *lhs*, const WStringView & *rhs*) [inline]

9.1.3.32 bool LLGL::operator!= (const WStringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.33 bool LLGL::operator!= (const WStringView & *lhs*, const wchar_t * *rhs*) [inline]

9.1.3.34 bool LLGL::operator!= (const char * *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.35 bool LLGL::operator!= (const char * *lhs*, const StringView & *rhs*) [inline]

9.1.3.36 bool LLGL::operator!= (const wchar_t * *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.37 bool LLGL::operator!= (const wchar_t * *lhs*, const WStringView & *rhs*) [inline]

9.1.3.38 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.

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

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

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

9.1.3.42 UTF8String LLGL::operator+ (const UTF8String & *lhs*, const UTF8String & *rhs*) [inline]

Concatenates the two UTF-8 string.

9.1.3.43 UTF8String LLGL::operator+ (const UTF8String & *lhs*, const StringView & *rhs*) [inline]

9.1.3.44 UTF8String LLGL::operator+ (const UTF8String & *lhs*, const WStringView & *rhs*) [inline]

9.1.3.45 UTF8String LLGL::operator+ (const UTF8String & *lhs*, const char * *rhs*) [inline]

9.1.3.46 UTF8String LLGL::operator+ (const UTF8String & *lhs*, const wchar_t * *rhs*) [inline]

9.1.3.47 UTF8String LLGL::operator+ (const StringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.48 UTF8String LLGL::operator+ (const WStringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.49 UTF8String LLGL::operator+ (const char * *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.50 UTF8String LLGL::operator+ (const wchar_t * *lhs*, const UTF8String & *rhs*) [inline]

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

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

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

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

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

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

Compares the two specified display mode descriptors on equality.

9.1.3.57 bool LLGL::operator== (const UTF8String & *lhs*, const UTF8String & *rhs*) [inline]

Returns true if the two strings are equal.

9.1.3.58 bool LLGL::operator== (const UTF8String & *lhs*, const StringView & *rhs*) [inline]

9.1.3.59 bool LLGL::operator== (const UTF8String & *lhs*, const WStringView & *rhs*) [inline]

9.1.3.60 bool LLGL::operator== (const UTF8String & *lhs*, const char * *rhs*) [inline]

9.1.3.61 bool LLGL::operator== (const UTF8String & *lhs*, const wchar_t * *rhs*) [inline]

9.1.3.62 bool LLGL::operator== (const StringView & *lhs*, const StringView & *rhs*) [inline]

9.1.3.63 bool LLGL::operator== (const StringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.64 bool LLGL::operator== (const StringView & *lhs*, const char * *rhs*) [inline]

9.1.3.65 bool LLGL::operator== (const WStringView & *lhs*, const WStringView & *rhs*) [inline]

9.1.3.66 bool LLGL::operator== (const WStringView & *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.67 bool LLGL::operator== (const WStringView & *lhs*, const wchar_t * *rhs*) [inline]

9.1.3.68 bool LLGL::operator== (const char * *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.69 bool LLGL::operator== (const char * *lhs*, const StringView & *rhs*) [inline]

9.1.3.70 bool LLGL::operator== (const wchar_t * *lhs*, const UTF8String & *rhs*) [inline]

9.1.3.71 bool LLGL::operator== (const wchar_t * *lhs*, const WStringView & *rhs*) [inline]

9.1.3.72 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).

9.1.3.73 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.

9.1.3.74 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.

Variables

- constexpr unsigned maxThreadCount = -1

  Specifies the maximal number of threads the host system supports.
- constexpr std::uint32_t ignoreOffset = -1

  Offset value to determine the offset automatically, e.g. to append a vertex attribute at the end of a vertex format.
- constexpr std::uint32_t invalidSlot = -1

  Specifies an invalid binding slot for shader resources.
- constexpr std::uint32_t invalidLocation = -1

  Specifies an invalid shader uniform location.
- constexpr std::uint64_t wholeSize = -1

  Specifies to use the whole size of a resource.
- constexpr std::uint32_t invalidTimerID = 0u

  Specifies an invalid timer ID for window events.

9.2.1 Detailed Description

Namespace with all constants used as default arguments.

9.2.2 Variable Documentation

9.2.2.1 constexpr std::uint32_t LLGL::Constants::ignoreOffset = -1

Offset value to determine the offset automatically, e.g. to append a vertex attribute at the end of a vertex format.

See also

VertexFormat::AppendAttribute

9.2.2.2 constexpr std::uint32_t LLGL::Constants::invalidLocation = -1

Specifies an invalid shader uniform location.

See also

Shader::FindUniformLocation

9.2.2.3 constexpr std::uint32_t LLGL::Constants::invalidSlot = -1

Specifies an invalid binding slot for shader resources.

See also

ShaderResource::slot

9.2.2.4 constexpr std::uint32_t LLGL::Constants::invalidTimerID = 0u

Specifies an invalid timer ID for window events.

See also

WindowBehavior::moveAndResizeTimerID

9.2.2.5 constexpr unsigned LLGL::Constants::maxThreadCount = -1

Specifies the maximal number of threads the host system supports.

See also

ConvertImageBuffer
RenderSystem::CreateTexture
RenderSystem::WriteTexture
RenderSystem::ReadTexture

9.2.2.6 constexpr std::uint64_t LLGL::Constants::wholeSize = -1

Specifies to use the whole size of a resource.

See also

CommandBuffer::FillBuffer

9.3 LLGL::Log Namespace Reference

Typedefs

- using ReportCallback = std::function< void(ReportType type, const StringView &message, const StringView &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 StringView &message, const StringView &contextInfo={})

  Posts a report to the currently set report callback.
- 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)

  Sets the new report callback to the standard output streams.
- LLGL_EXPORT void SetReportLimit (std::size_t maxCount)

  Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.

9.3.1 Typedef Documentation

9.3.1.1 using LLGL::Log::ReportCallback = typedef std::function<void(ReportType type, const StringView& message, const StringView& 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 StringView & *message*, const StringView & *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*)

Sets the new report callback to the standard output streams.

Parameters

in	*stream*	Specifies a pointer to the output stream. If this is null, the standard output stream is effectively disabled.

See also

SetReportCallback

9.3.3.4 LLGL_EXPORT void LLGL::Log::SetReportLimit (std::size_t *maxCount*)

Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.

Parameters

in	*maxCount*	Specifies the maximum number of reports. If this is 0, there is affectively no limit. By default 0.

9.4 LLGL::Timer Namespace Reference

Functions

- LLGL_EXPORT std::uint64_t Frequency ()

  Returns the timer frequency of the OS. This is the number of ticks per seconds, e.g. 1,000,000 for microseconds.
- LLGL_EXPORT std::uint64_t Tick ()

  Returns the current 'tick' of a high resolution timer that is OS dependent. The tick frequency can be queried with Frequency.

9.4.1 Function Documentation

9.4.1.1 LLGL_EXPORT std::uint64_t LLGL::Timer::Frequency ( )

Returns the timer frequency of the OS. This is the number of ticks per seconds, e.g. 1,000,000 for microseconds.

See also

Tick

9.4.1.2 LLGL_EXPORT std::uint64_t LLGL::Timer::Tick ( )

Returns the current 'tick' of a high resolution timer that is OS dependent. The tick frequency can be queried with Frequency.

Remarks

The following example illustrates how to query the elapsed time in milliseconds between two time stamps:

```
const std::uint64_t startTime = LLGL::Timer::Tick();
// Some events ...
const std::uint64_t endTime = LLGL::Timer::Tick();
const double elapsedSeconds = static_cast<double>(endTime - startTime) /
      LLGL::Timer::Frequency();
std::cout << "Elapsed time: " << elapsedSeconds * 1000.0 << "ms" << std::endl;
```

See also

Frequency

9.5 LLGL::Version Namespace Reference

Namespace with functions to determine LLGL version.

Functions

- LLGL_EXPORT unsigned GetMajor ()

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

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

  Returns the revision version number. Must be less than 100.
- LLGL_EXPORT const char * GetStatus ()

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

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

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

9.5.1 Detailed Description

Namespace with functions to determine LLGL version.

9.5.2 Function Documentation

9.5.2.1 LLGL_EXPORT unsigned LLGL::Version::GetID ( )

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

9.5.2.2 LLGL_EXPORT unsigned LLGL::Version::GetMajor ( )

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

9.5.2.3 LLGL_EXPORT unsigned LLGL::Version::GetMinor ( )

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

9.5.2.4 LLGL_EXPORT unsigned LLGL::Version::GetRevision ( )

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

9.5.2.5 LLGL_EXPORT const char* LLGL::Version::GetStatus ( )

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

9.5.2.6 LLGL_EXPORT const char* LLGL::Version::GetString ( )

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

Chapter 10

Class Documentation

10.1 LLGL::ApplicationDescriptor Struct Reference

Application descriptor structure.

```
#include <RendererConfiguration.h>
```

Public Attributes

- const char * applicationName = nullptr

  Descriptive string of the application.
- std::uint32_t applicationVersion = 0

  Version number of the application.
- const char * engineName = nullptr

  Descriptive string of the engine or middleware.
- std::uint32_t engineVersion = 0

  Version number of the engine or middleware.

10.1.1 Detailed Description

Application descriptor structure.

Note

Only supported with: Vulkan.

See also

RendererConfigurationVulkan::application

10.1.2 Member Data Documentation

10.1.2.1 const char* LLGL::ApplicationDescriptor::applicationName = nullptr

Descriptive string of the application.

10.1.2.2 std::uint32_t LLGL::ApplicationDescriptor::applicationVersion = 0

Version number of the application.

10.1.2.3 const char* LLGL::ApplicationDescriptor::engineName = nullptr

Descriptive string of the engine or middleware.

10.1.2.4 std::uint32_t LLGL::ApplicationDescriptor::engineVersion = 0

Version number of the engine or middleware.

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

- RendererConfiguration.h

10.2 LLGL::ArrayView< T > Class Template Reference

Constant array view container template. Keeps pointer to array and length.

```
#include <ArrayView.h>
```

Public Types

- using value_type = const T
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using const_reference = const T &
- using reference = const_reference
- using const_pointer = const T *
- using pointer = const_pointer
- using const_iterator = const T *
- using iterator = const_iterator
- using const_reverse_iterator = std::reverse_iterator< const_iterator >
- using reverse_iterator = const_reverse_iterator

Public Member Functions

- ArrayView ()=default
- ArrayView (const ArrayView &)=default
- ArrayView (ArrayView &&)=default
- ArrayView & operator= (const ArrayView &)=default
- ArrayView & operator= (ArrayView &&)=default
- ArrayView (const std::initializer_list< T > &list)

  Initializes the array view with an std::initializer_list.
- ArrayView (const_pointer data, size_type size)

  Initializes the array view with a pointer to the data and size.
- template<std::size_t N>
  ArrayView (const T(&data)[N])

  Initializes the array view with the specified fixed size array. The size of this container will be equal to the template parameter <N>.
- template<template< typename, typename > class OtherContainer, typename OtherAllocator >
  ArrayView (const OtherContainer< T, OtherAllocator > &other)
- bool empty () const
- size_type size () const
- const_pointer data () const
- const_reference at (size_type pos) const
- const_reference front () const
- const_reference back () const
- const_iterator begin () const
- const_iterator cbegin () const
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const
- const_iterator end () const
- const_iterator cend () const
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const
- const_reference operator[ ] (size_type pos) const

10.2.1 Detailed Description

template<typename T>
class LLGL::ArrayView< T >

Constant array view container template. Keeps pointer to array and length.

Template Parameters

T	Specifies the data type of the container elements.

10.2.2 Member Typedef Documentation

10.2.2.1 template<typename T> using LLGL::ArrayView< T >::const_iterator = const T*

10.2.2.2 template<typename T> using LLGL::ArrayView< T >::const_pointer = const T*

10.2.2.3 template<typename T> using LLGL::ArrayView< T >::const_reference = const T&

10.2.2.4 template<typename T> using LLGL::ArrayView< T >::const_reverse_iterator = std::reverse_iterator<const_iterator>

10.2.2.5 template<typename T> using LLGL::ArrayView< T >::difference_type = std::ptrdiff_t

10.2.2.6 template<typename T> using LLGL::ArrayView< T >::iterator = const_iterator

10.2.2.7 template<typename T> using LLGL::ArrayView< T >::pointer = const_pointer

10.2.2.8 template<typename T> using LLGL::ArrayView< T >::reference = const_reference

10.2.2.9 template<typename T> using LLGL::ArrayView< T >::reverse_iterator = const_reverse_iterator

10.2.2.10 template<typename T> using LLGL::ArrayView< T >::size_type = std::size_t

10.2.2.11 template<typename T> using LLGL::ArrayView< T >::value_type = const T

10.2.3 Constructor & Destructor Documentation

10.2.3.1 template<typename T> LLGL::ArrayView< T >::ArrayView ( ) [default]

10.2.3.2 template<typename T> LLGL::ArrayView< T >::ArrayView (const ArrayView< T > &) [default]

10.2.3.3 template<typename T> LLGL::ArrayView< T >::ArrayView (ArrayView< T > &&) [default]

10.2.3.4 template<typename T> LLGL::ArrayView< T >::ArrayView (const std::initializer_list< T > & *list*) [inline]

Initializes the array view with an std::initializer_list.

10.2.3.5 template<typename T> LLGL::ArrayView< T >::ArrayView (const_pointer *data*, size_type *size*) [inline]

Initializes the array view with a pointer to the data and size.

10.2.3.6 template<typename T> template<std::size_t N> LLGL::ArrayView< T >::ArrayView (const T(&) *data[N]*) [inline]

Initializes the array view with the specified fixed size array. The size of this container will be equal to the template parameter <N>.

10.2.3.7 template<typename T> template<template< typename, typename > class OtherContainer, typename OtherAllocator > LLGL::ArrayView< T >::ArrayView (const OtherContainer< T, OtherAllocator > & *other*) [inline]

10.2.4 Member Function Documentation

10.2.4.1 template<typename T> const_reference LLGL::ArrayView< T >::at (size_type *pos*) const [inline]

10.2.4.2 template<typename T> const_reference LLGL::ArrayView< T >::back ( ) const [inline]

10.2.4.3 template<typename T> const_iterator LLGL::ArrayView< T >::begin ( ) const [inline]

10.2.4.4 template<typename T> const_iterator LLGL::ArrayView< T >::cbegin ( ) const [inline]

10.2.4.5 template<typename T> const_iterator LLGL::ArrayView< T >::cend ( ) const [inline]

10.2.4.6 template<typename T> const_reverse_iterator LLGL::ArrayView< T >::crbegin ( ) const [inline]

10.2.4.7 template<typename T> const_reverse_iterator LLGL::ArrayView< T >::crend ( ) const [inline]

10.2.4.8 template<typename T> const_pointer LLGL::ArrayView< T >::data ( ) const [inline]

10.2.4.9 template<typename T> bool LLGL::ArrayView< T >::empty ( ) const [inline]

10.2.4.10 template<typename T> const_iterator LLGL::ArrayView< T >::end ( ) const [inline]

10.2.4.11 template<typename T> const_reference LLGL::ArrayView< T >::front ( ) const [inline]

10.2.4.12 template<typename T> ArrayView& LLGL::ArrayView< T >::operator= (const ArrayView< T > &) [default]

10.2.4.13 template<typename T> ArrayView& LLGL::ArrayView< T >::operator= (ArrayView< T > &&) [default]

10.2.4.14 template<typename T> const_reference LLGL::ArrayView< T >::operator[] (size_type *pos*) const [inline]

10.2.4.15 template<typename T> const_reverse_iterator LLGL::ArrayView< T >::rbegin ( ) const [inline]

10.2.4.16 template<typename T> const_reverse_iterator LLGL::ArrayView< T >::rend ( ) const [inline]

10.2.4.17 template<typename T> size_type LLGL::ArrayView< T >::size ( ) const [inline]

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

- ArrayView.h

10.3 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.

10.3.1 Detailed Description

Attachment clear command structure.

See also

CommandBuffer::ClearAttachments

10.3.2 Constructor & Destructor Documentation

10.3.2.1 LLGL::AttachmentClear::AttachmentClear ( ) [default]

10.3.2.2 LLGL::AttachmentClear::AttachmentClear (const AttachmentClear &) [default]

10.3.2.3 LLGL::AttachmentClear::AttachmentClear (const ColorRGBAf & *color*, std::uint32_t *colorAttachment*) [inline]

Constructor for a color attachment clear command.

10.3.2.4 LLGL::AttachmentClear::AttachmentClear (float *depth*) [inline]

Constructor for a depth attachment clear command.

10.3.2.5 LLGL::AttachmentClear::AttachmentClear (std::uint32_t *stencil*) [inline]

Constructor for a stencil attachment clear command.

10.3.2.6 LLGL::AttachmentClear::AttachmentClear (float *depth*, std::uint32_t *stencil*) [inline]

Constructor for a depth-stencil attachment clear command.

10.3.3 Member Function Documentation

10.3.3.1 AttachmentClear& LLGL::AttachmentClear::operator= (const AttachmentClear &) [default]

10.3.4 Member Data Documentation

10.3.4.1 ClearValue LLGL::AttachmentClear::clearValue

Clear value for color, depth, and stencil buffers.

10.3.4.2 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

10.3.4.3 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

10.4 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 arrayLayer=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.

10.4.1 Detailed Description

Render target attachment descriptor structure.

See also

RenderTargetDescriptor

10.4.2 Constructor & Destructor Documentation

10.4.2.1 LLGL::AttachmentDescriptor::AttachmentDescriptor ( ) [default]

10.4.2.2 LLGL::AttachmentDescriptor::AttachmentDescriptor (const AttachmentDescriptor &) [default]

10.4.2.3 LLGL::AttachmentDescriptor::AttachmentDescriptor (AttachmentType *type*) [inline]

Constructor for the specified depth-, or stencil attachment.

10.4.2.4 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.

10.4.3 Member Data Documentation

10.4.3.1 std::uint32_t LLGL::AttachmentDescriptor::arrayLayer = 0

Specifies the array texture layer which is to be used as render target attachment.

Remarks

This is only used for array textures and cube textures (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCube, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray).
For cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray), each cube has its own 6 array layers. The layer index for the respective cube faces is described at the TextureDescriptor::arrayLayers member.
If `texture` is null, this attribute is ignored.

See also

TextureDescriptor::arrayLayers

10.4.3.2 std::uint32_t LLGL::AttachmentDescriptor::mipLevel = 0

Specifies the MIP-map level which is to be attached to a render target.

Remarks

This is only used for non-multi-sample textures. All multi-sample textures will always use the first MIP-map level (i.e. TextureType::Texture2DMS and TextureType::Texture2DMSArray).
If `texture` is null, this attribute is ignored.

10.4.3.3 Texture* LLGL::AttachmentDescriptor::texture = nullptr

Pointer to the texture which is to be used as target output. By default null.

Remarks

If this is null, the attribute `type` **must not** be AttachmentType::Color. The texture must also have been created either with the binding flag BindFlags::ColorAttachment or BindFlags::DepthStencilAttachment.

See also

AttachmentDescriptor::type
TextureDescriptor::bindFlags

10.4.3.4 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

10.5 LLGL::AttachmentFormatDescriptor Struct Reference

Render pass 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.

10.5.1 Detailed Description

Render pass attachment descriptor structure.

Remarks

Two attachment format descriptors are considered compatible when their formats and multi-sampling attributes are matching.

See also

RenderPassDescriptor

10.5.2 Constructor & Destructor Documentation

10.5.2.1 LLGL::AttachmentFormatDescriptor::AttachmentFormatDescriptor ( ) [default]

10.5.2.2 LLGL::AttachmentFormatDescriptor::AttachmentFormatDescriptor (const AttachmentFormatDescriptor &) [default]

10.5.2.3 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.

10.5.3 Member Data Documentation

10.5.3.1 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 swap-chain, the appropriate color format can be determined by the SwapChain::GetColorFormat function, and the appropriate depth-stencil format can be determined by the SwapChain::GetDepthStencilFormat function. If the render pass is used for render targets, the format depends on the render target attachments. If this is undefined, the corresponding attachment is not used.

See also

SwapChain::GetColorFormat
SwapChain::GetDepthStencilFormat

10.5.3.2 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

10.5.3.3 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

10.6 LLGL::BarrierFlags Struct Reference

Flags for memory barriers in resource heaps.

```
#include <ResourceHeapFlags.h>
```

Public Types

- enum { StorageBuffer = (1 << 0), StorageTexture = (1 << 1), Storage = (StorageBuffer | StorageTexture) }

10.6.1 Detailed Description

Flags for memory barriers in resource heaps.

See also

ResourceHeapDescriptor::barrierFlags

10.6.2 Member Enumeration Documentation

10.6.2.1 anonymous enum

Enumerator

StorageBuffer Memory barrier for Buffer resources that were created with the BindFlags::Storage bind flags.

Remarks

Shader access to the buffer will reflect all data written to by previous shaders.

See also

BindFlags::Storage

StorageTexture Memory barrier for Texture resources that were created with the BindFlags::Storage bind flags.

Remarks

Shader access to the texture will reflect all data written to by previous shaders.

See also

BindFlags::Storage

Storage Memory barrier for any storage resource. This is just a bitwise OR combination of `Storage↩Buffer` and `StorageTexture`.

Remarks

Renderer backends such as Direct3D 12 and Vulkan have bookkeeping for storage resources and don't have to distinguish between Buffer and Texture resource views for their barriers at time of creating the ResourceHeap. Hence, using BarrierFlags::Storage by default when any resource views in the ResourceHeap have to be synchronized is recommended. Only the OpenGL backend has to know at creation time what type of resources need a global barrier via `glMemoryBarrier`.

See also

BindFlags::Storage

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

- ResourceHeapFlags.h

10.7 LLGL::BasicStringView< TChar, Traits > Class Template Reference

Constant string view container template. Keeps pointer to string and its length.

```
#include <StringView.h>
```

Public Types

- using value_type = const TChar
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using const_reference = const TChar &
- using reference = const_reference
- using const_pointer = const TChar *
- using pointer = const_pointer
- using const_iterator = const TChar *
- using iterator = const_iterator
- using const_reverse_iterator = std::reverse_iterator< const_iterator >
- using reverse_iterator = const_reverse_iterator

Public Member Functions

- BasicStringView ()=default
- BasicStringView (const BasicStringView &)=default
- BasicStringView (BasicStringView &&)=default
- BasicStringView & operator= (const BasicStringView &)=default
- BasicStringView & operator= (BasicStringView &&)=default
- BasicStringView (const_pointer str)

  Initializes the string view with the specified null-terminated string.
- BasicStringView (const_pointer str, size_type len)

  Initializes the string view with a pointer to the data and size.

- template<template< class, class, class > class TString, class Allocator >
  BasicStringView (const TString< TChar, Traits, Allocator > &str)

  Initializes the string view with another templated string class. This would usually be std::basic_string.
- bool empty () const
- size_type size () const
- size_type length () const
- const_pointer data () const
- const_reference at (size_type pos) const
- const_reference front () const
- const_reference back () const
- const_iterator begin () const
- const_iterator cbegin () const
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const
- const_iterator end () const
- const_iterator cend () const
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const
- BasicStringView substr (size_type pos=0, size_type count=npos) const
- int compare (const BasicStringView &str) const
- int compare (size_type pos1, size_type count1, const BasicStringView &str) const
- int compare (size_type pos1, size_type count1, const BasicStringView &str, size_type pos2, size_type count2=npos) const
- const_reference operator[ ] (size_type pos) const

Static Public Attributes

- static const size_type npos = -1

10.7.1 Detailed Description

template<typename TChar, typename Traits = std::char_traits<TChar>>
class LLGL::BasicStringView< TChar, Traits >

Constant string view container template. Keeps pointer to string and its length.

Remarks

This does not provide a null-terminated string.

Template Parameters

TChar	Specifies the string character type.
Traits	Specifies the character traits. By default std::char_traits<TChar>.

10.7.2 Member Typedef Documentation

10.7.2.1 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**const_iterator** = const TChar*

10.7.2.2 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**const_pointer** = const TChar*

10.7.2.3 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**const_reference** = const TChar&

10.7.2.4 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**const_reverse_iterator** = std::reverse_iterator<**const_iterator**>

10.7.2.5 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**difference_type** = std::ptrdiff_t

10.7.2.6 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**iterator** = **const_iterator**

10.7.2.7 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**pointer** = **const_pointer**

10.7.2.8 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**reference** = **const_reference**

10.7.2.9 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**reverse_iterator** = **const_reverse_iterator**

10.7.2.10 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**size_type** = std::size_t

10.7.2.11 template<typename TChar , typename Traits = std::char_traits<TChar>> using **LLGL::BasicStringView**< TChar, Traits >::**value_type** = const TChar

10.7.3 Constructor & Destructor Documentation

10.7.3.1 template<typename TChar , typename Traits = std::char_traits<TChar>> **LLGL::BasicStringView**< TChar, Traits >::**BasicStringView** ( ) [default]

10.7.3.2 template<typename TChar , typename Traits = std::char_traits<TChar>> **LLGL::BasicStringView**< TChar, Traits >::**BasicStringView** (const **BasicStringView**< TChar, Traits > &) [default]

10.7.3.3 template<typename TChar , typename Traits = std::char_traits<TChar>> **LLGL::BasicStringView**< TChar, Traits >::**BasicStringView** (**BasicStringView**< TChar, Traits > &&) [default]

10.7.3.4 template<typename TChar , typename Traits = std::char_traits<TChar>> **LLGL::BasicStringView**< TChar, Traits >::**BasicStringView** (**const_pointer** *str*) [inline]

Initializes the string view with the specified null-terminated string.

10.7.3.5 template<typename TChar , typename Traits = std::char_traits<TChar>> LLGL::BasicStringView< TChar, Traits >::BasicStringView (const_pointer *str,* size_type *len*) [inline]

Initializes the string view with a pointer to the data and size.

10.7.3.6 template<typename TChar , typename Traits = std::char_traits<TChar>> template<template< class, class, class > class TString, class Allocator > LLGL::BasicStringView< TChar, Traits >::BasicStringView (const TString< TChar, Traits, Allocator > & *str*) [inline]

Initializes the string view with another templated string class. This would usually be std::basic_string.

10.7.4 Member Function Documentation

10.7.4.1 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reference LLGL::BasicStringView< TChar, Traits >::at (size_type *pos*) const [inline]

10.7.4.2 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reference LLGL::BasicStringView< TChar, Traits >::back ( ) const [inline]

10.7.4.3 template<typename TChar , typename Traits = std::char_traits<TChar>> const_iterator LLGL::BasicStringView< TChar, Traits >::begin ( ) const [inline]

10.7.4.4 template<typename TChar , typename Traits = std::char_traits<TChar>> const_iterator LLGL::BasicStringView< TChar, Traits >::cbegin ( ) const [inline]

10.7.4.5 template<typename TChar , typename Traits = std::char_traits<TChar>> const_iterator LLGL::BasicStringView< TChar, Traits >::cend ( ) const [inline]

10.7.4.6 template<typename TChar , typename Traits = std::char_traits<TChar>> int LLGL::BasicStringView< TChar, Traits >::compare (const BasicStringView< TChar, Traits > & *str*) const [inline]

10.7.4.7 template<typename TChar , typename Traits = std::char_traits<TChar>> int LLGL::BasicStringView< TChar, Traits >::compare (size_type *pos1,* size_type *count1,* const BasicStringView< TChar, Traits > & *str*) const [inline]

10.7.4.8 template<typename TChar , typename Traits = std::char_traits<TChar>> int LLGL::BasicStringView< TChar, Traits >::compare (size_type *pos1,* size_type *count1,* const BasicStringView< TChar, Traits > & *str,* size_type *pos2,* size_type *count2* = npos) const [inline]

10.7.4.9 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reverse_iterator LLGL::BasicStringView< TChar, Traits >::crbegin ( ) const [inline]

10.7.4.10 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reverse_iterator LLGL::BasicStringView< TChar, Traits >::crend ( ) const [inline]

10.7.4.11 template<typename TChar , typename Traits = std::char_traits<TChar>> const_pointer LLGL::BasicStringView< TChar, Traits >::data ( ) const [inline]

10.7.4.12 template<typename TChar , typename Traits = std::char_traits<TChar>> bool LLGL::BasicStringView< TChar, Traits >::empty ( ) const [inline]

10.7.4.13 template<typename TChar , typename Traits = std::char_traits<TChar>> const_iterator LLGL::BasicStringView< TChar, Traits >::end ( ) const [inline]

10.7.4.14 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reference LLGL::BasicStringView< TChar, Traits >::front ( ) const [inline]

10.7.4.15 template<typename TChar , typename Traits = std::char_traits<TChar>> size_type LLGL::BasicStringView< TChar, Traits >::length ( ) const [inline]

10.7.4.16 template<typename TChar , typename Traits = std::char_traits<TChar>> BasicStringView& LLGL::BasicStringView< TChar, Traits >::operator= (const BasicStringView< TChar, Traits > &) [default]

10.7.4.17 template<typename TChar , typename Traits = std::char_traits<TChar>> BasicStringView& LLGL::BasicStringView< TChar, Traits >::operator= (BasicStringView< TChar, Traits > &&) [default]

10.7.4.18 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reference LLGL::BasicStringView< TChar, Traits >::operator[] (size_type *pos*) const [inline]

10.7.4.19 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reverse_iterator LLGL::BasicStringView< TChar, Traits >::rbegin ( ) const [inline]

10.7.4.20 template<typename TChar , typename Traits = std::char_traits<TChar>> const_reverse_iterator LLGL::BasicStringView< TChar, Traits >::rend ( ) const [inline]

10.7.4.21 template<typename TChar , typename Traits = std::char_traits<TChar>> size_type LLGL::BasicStringView< TChar, Traits >::size ( ) const [inline]

10.7.4.22 template<typename TChar , typename Traits = std::char_traits<TChar>> BasicStringView LLGL::BasicStringView< TChar, Traits >::substr (size_type *pos* = 0, size_type *count* = npos) const [inline]

10.7.5 Member Data Documentation

10.7.5.1 template<typename TChar , typename Traits = std::char_traits<TChar>> const size_type LLGL::BasicStringView< TChar, Traits >::npos = -1 [static]

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

- StringView.h

10.8 LLGL::BindFlags Struct Reference

Flags for Buffer and Texture resources that describe for which purposes they will be used.

```
#include <ResourceFlags.h>
```

Public Types

- enum {
  VertexBuffer = (1 << 0), IndexBuffer = (1 << 1), ConstantBuffer = (1 << 2), StreamOutputBuffer = (1 << 3),
  IndirectBuffer = (1 << 4), Sampled = (1 << 5), Storage = (1 << 6), ColorAttachment = (1 << 7),
  DepthStencilAttachment = (1 << 8), CombinedSampler = (1 << 9), CopySrc = (1 << 10), CopyDst = (1 << 11) }

10.8.1 Detailed Description

Flags for Buffer and Texture resources that describe for which purposes they will be used.

Remarks

Resources can be created with both input and output binding flags, but they cannot be used together when the resource is bound. See the following table for compatibility:

Binding type	Binding flags
Input	BindFlags::Sampled, BindFlags::CopySrc, BindFlags::VertexBuffer, BindFlags::IndexBuffer, BindFlags::ConstantBuffer, BindFlags::IndirectBuffer
Output	BindFlags::Storage, BindFlags::CopyDst, BindFlags::ColorAttachment, BindFlags::DepthStencilAttachment, BindFlags::StreamOutputBuffer

See also

BufferDescriptor::bindFlags
TextureDescriptor::bindFlags
BindingDescriptor::bindFlags

10.8.2 Member Enumeration Documentation

10.8.2.1 anonymous enum

Enumerator

VertexBuffer The resource can be used to bind a stream of vertices.

Remarks

This can only be used for Buffer resources.

See also

CommandBuffer::SetVertexBuffer

IndexBuffer The resource can be used to bind a stream of indices.

Remarks

This can only be used for Buffer resources.

See also

CommandBuffer::SetIndexBuffer

ConstantBuffer The resource can be used to bind a set of constants.

Remarks

This can only be used for Buffer resources.

StreamOutputBuffer The resource can be used to bind an output stream buffer (also referred to as "transform feedback").

Remarks

This can only be used for Buffer resources.

See also

CommandBuffer::BeginStreamOutput

IndirectBuffer Hint to the renderer that the resource will hold the arguments for indirect commands.

Remarks

This can only be used for Buffer resources.

See also

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect
CommandBuffer::DispatchIndirect

Sampled The resource can be used to bind a buffer or texture for read access.

Remarks

This can be used for Buffer resources (e.g. `samplerBuffer` in GLSL, or `Structured↩Buffer` in HLSL) and Texture resources (e.g. `sampler2D` in GLSL, or `Texture2D` in HLSL).

Storage The resource can be used to bind a buffer or texture for unordered read/write access.

Remarks

This can be used for Buffer resources (e.g. `buffer` in GLSL, or `RWStructuredBuffer` in HLSL) and Texture resources (e.g. `image2D` in GLSL, or `RWTexture2D` in HLSL).

ColorAttachment Texture can be used as render target color attachment.

Remarks

This can only be used for Texture resources.

Note

This cannot be used together with the BindFlags::DepthStencilAttachment flag.

See also

AttachmentDescriptor::texture
AttachmentType::Color

DepthStencilAttachment Texture can be used as render target depth-stencil attachment.

Remarks

This can only be used for Texture resources.

Note

This cannot be used together with the BindFlags::ColorAttachment flag.

See also

AttachmentDescriptor::texture
AttachmentType::DepthStencil

CombinedSampler Specifies a resource as a combination of a Texture and Sampler (e.g. `sampler2D` in GLSL).

Remarks

This is only used for shader reflection and ignored by resource creation.

Note

Only supported with: OpenGL, Vulkan.

See also

ShaderResource::binding

CopySrc Specifies a resource can be used as source for a copy command.

See also

CommandBuffer::CopyBuffer
CommandBuffer::CopyBufferFromTexture
CommandBuffer::CopyTexture
CommandBuffer::CopyTextureFromBuffer

CopyDst Specifies a resource can be used as destination for a copy or fill command.

See also

CommandBuffer::CopyBuffer
CommandBuffer::CopyBufferFromTexture
CommandBuffer::CopyTexture
CommandBuffer::CopyTextureFromBuffer
CommandBuffer::FillBuffer

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

- ResourceFlags.h

10.9 LLGL::BindingDescriptor Struct Reference

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

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

Public Member Functions

- BindingDescriptor ()=default
- BindingDescriptor (const BindingDescriptor &)=default
- BindingDescriptor (ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1)

  Constructors with all primary attributes and a default value for a uniform array.
- BindingDescriptor (const std::string &name, ResourceType type, long bindFlags, long stageFlags, std::uint32_t slot, std::uint32_t arraySize=1)

  Constructors with all attributes.

Public Attributes

- std::string name

  Optional name for shading languages that do not support binding slots within the shader.
- ResourceType type = ResourceType::Undefined

  Resource view type for this layout binding. By default ResourceType::Undefined.
- long bindFlags = 0

  Specifies to which kind of resource slot the resource will be bound. By default 0.
- long stageFlags = 0

  Specifies which shader stages are affected by this layout binding. By default 0.
- std::uint32_t slot = 0

  Specifies the zero-based binding slot. By default 0.
- std::uint32_t arraySize = 1

  Specifies the number of binding slots for an array resource. By default 1.

10.9.1 Detailed Description

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

See also

PipelineLayoutDescriptor::bindings

10.9.2 Constructor & Destructor Documentation

10.9.2.1 LLGL::BindingDescriptor::BindingDescriptor ( ) [default]

10.9.2.2 LLGL::BindingDescriptor::BindingDescriptor (const BindingDescriptor &) [default]

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

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

10.9.2.4 LLGL::BindingDescriptor::BindingDescriptor (const std::string & *name*, ResourceType *type*, long *bindFlags*, long *stageFlags*, std::uint32_t *slot*, std::uint32_t *arraySize* = 1) [inline]

Constructors with all attributes.

10.9.3 Member Data Documentation

10.9.3.1 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).

10.9.3.2 long LLGL::BindingDescriptor::bindFlags = 0

Specifies to which kind of resource slot the resource will be bound. By default 0.

Remarks

Input and output binding flags cannot be used together when a resource is bound, e.g. a texture cannot be sampled while it is written to.
When a Buffer is bound to a constant buffer slot for instance, the binding flag BindFlags::ConstantBuffer is required. When a Texture is bound to a sampled texture slot, the binding flag BindFlags::Sampled is required and so on.

See also

BindFlags

10.9.3.3 std::string LLGL::BindingDescriptor::name

Optional name for shading languages that do not support binding slots within the shader.

Remarks

This is only used for the OpenGL backend, when the GLSL version does not support `explicit binding points`. If GLSL 420 or later is supported, this can be ignored and the binding points can be specified like this:

```
#version 420
layout(binding = 1) uniform sampler2D mySampler;
```

Otherwise, the name of the resource must be included in this binding descriptor, e.g. `"mySampler"`.

10.9.3.4 std::uint32_t LLGL::BindingDescriptor::slot = 0

Specifies the zero-based binding slot. By default 0.

Note

For Vulkan, each binding slot of all layout bindings must have a different value within a pipeline layout.

10.9.3.5 long LLGL::BindingDescriptor::stageFlags = 0

Specifies which shader stages are affected by this layout binding. By default 0.

Remarks

This can be a bitwise OR combination of the StageFlags bitmasks.

See also

StageFlags

10.9.3.6 ResourceType LLGL::BindingDescriptor::type = ResourceType::Undefined

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

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

- PipelineLayoutFlags.h

10.10 LLGL::BlendDescriptor Struct Reference

Blending state descriptor structure.

`#include <PipelineStateFlags.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.
- std::uint32_t sampleMask = ~0u

  Specifies the sample bitmask if alpha coverage is enabled. By default `0xFFFFFFFF`.
- 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).
- bool blendFactorDynamic = false

  Specifies whether the blend factor will be set dynamically with the command buffer. By default false.
- BlendTargetDescriptor targets [LLGL_MAX_NUM_COLOR_ATTACHMENTS]

  Render-target blend states for the respective color attachments. A maximum of 8 targets is supported.

10.10.1 Detailed Description

Blending state descriptor structure.

See also

GraphicsPipelineDescriptor::blend

10.10.2 Member Data Documentation

10.10.2.1 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).

See also

sampleMask

10.10.2.2 ColorRGBAf LLGL::BlendDescriptor::blendFactor = { 0.0f, 0.0f, 0.0f, 0.0f }

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

Remarks

This is only used if any blending operations of any blending target is either BlendOp::BlendFactor or BlendOp::InvBlendFactor.
If `blendFactorDynamic` is set to true, this member is ignored.

See also

BlendOp::BlendFactor
BlendOp::InvBlendFactor
CommandBuffer::SetBlendFactor

10.10.2.3 bool LLGL::BlendDescriptor::blendFactorDynamic = false

Specifies whether the blend factor will be set dynamically with the command buffer. By default false.

Remarks

If this is true, `blendFactor` is ignored and the blending factors must be set with the `SetBlendFactor` function everytime the graphics pipeline is set.

See also

CommandBuffer::SetBlendFactor

10.10.2.4 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

10.10.2.5 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

10.10.2.6 std::uint32_t LLGL::BlendDescriptor::sampleMask = ~0u

Specifies the sample bitmask if alpha coverage is enabled. By default `0xFFFFFFFF`.

Remarks

If `alphaToCoverageEnabled` is false, this field is ignored.

See also

alphaToCoverageEnabled

10.10.2.7 BlendTargetDescriptor LLGL::BlendDescriptor::targets[LLGL_MAX_NUM_COLOR_ATTACHMENTS]

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:

- PipelineStateFlags.h

10.11 LLGL::BlendTargetDescriptor Struct Reference

Blend target state descriptor structure.

```
#include <PipelineStateFlags.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)`.

10.11.1 Detailed Description

Blend target state descriptor structure.

See also

BlendDescriptor::targets

10.11.2 Member Data Documentation

10.11.2.1 BlendArithmetic LLGL::BlendTargetDescriptor::alphaArithmetic = BlendArithmetic::Add

Alpha blending arithmetic. By default BlendArithmetic::Add.

10.11.2.2 bool LLGL::BlendTargetDescriptor::blendEnabled = false

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

10.11.2.3 BlendArithmetic LLGL::BlendTargetDescriptor::colorArithmetic = BlendArithmetic::Add

Color blending arithmetic. By default BlendArithmetic::Add.

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

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

Remarks

If no pixel shader is used in the graphics pipeline, the color mask **must** be set to `false` for all components. Otherwise, the behavior is undefined.

10.11.2.5 BlendOp LLGL::BlendTargetDescriptor::dstAlpha = BlendOp::InvSrcAlpha

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

10.11.2.6 BlendOp LLGL::BlendTargetDescriptor::dstColor = BlendOp::InvSrcAlpha

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

10.11.2.7 BlendOp LLGL::BlendTargetDescriptor::srcAlpha = BlendOp::SrcAlpha

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

10.11.2.8 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:

- PipelineStateFlags.h

10.12 LLGL::Blob Class Reference

CPU read-only buffer of arbitrary size.

```
#include <Blob.h>
```

Inheritance diagram for LLGL::Blob:

Public Member Functions

- virtual const void * GetData () const =0

  Returns a constant pointer to the internal buffer.
- virtual std::size_t GetSize () const =0

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

Static Public Member Functions

- static std::unique_ptr< Blob > CreateCopy (const void *data, std::size_t size)

  Creates a new Blob instance with a copy of the specified data.
- static std::unique_ptr< Blob > CreateWeakRef (const void *data, std::size_t size)

  Creates a new Blob instance with a weak reference to the specified data.
- static std::unique_ptr< Blob > CreateStrongRef (std::vector< std::int8_t > &&container)

  Creates a new Blob instance with a strong reference to the specified vector container.
- static std::unique_ptr< Blob > CreateStrongRef (std::string &&container)

  Creates a new Blob instance with a strong reference to the specified string container.
- static std::unique_ptr< Blob > CreateFromFile (const char *filename)

  Creates a new Blob instance with the data read from the specified binary file.
- static std::unique_ptr< Blob > CreateFromFile (const std::string &filename)

  Creates a new Blob instance with the data read from the specified binary file.

Additional Inherited Members

10.12.1 Detailed Description

CPU read-only buffer of arbitrary size.

See also

RenderSystem::CreatePipelineState

10.12.2 Member Function Documentation

10.12.2.1 static std::unique_ptr<Blob> LLGL::Blob::CreateCopy (const void * *data*, std::size_t *size*) [static]

Creates a new Blob instance with a copy of the specified data.

Parameters

in	*data*	Pointer to the data that is to be copied and managed by this blob.
in	*size*	Specifies the size (in bytes) of the data.

Returns

New instance of Blob that manages the memory that is copied.

10.12.2.2 static std::unique_ptr<Blob> LLGL::Blob::CreateFromFile (const char * *filename*) [static]

Creates a new Blob instance with the data read from the specified binary file.

Parameters

in	*filename*	Specifies the file that is to be read.

Returns

New instance of Blob that manages the memory of a conent copy from the specified file or null if the file could not be read.

10.12.2.3 static std::unique_ptr<Blob> LLGL::Blob::CreateFromFile (const std::string & *filename*) [static]

Creates a new Blob instance with the data read from the specified binary file.

Parameters

in	*filename*	Specifies the file that is to be read.

Returns

New instance of Blob that manages the memory of a conent copy from the specified file or null if the file could not be read.

10.12.2.4 static std::unique_ptr<Blob> LLGL::Blob::CreateStrongRef (std::vector< std::int8_t > && *container*) [static]

Creates a new Blob instance with a strong reference to the specified vector container.

Parameters

in	*container*	Specifies the container whose data is to be moved into this Blob instance.

Returns

New instance of Blob that manages the specified container.

10.12.2.5 static std::unique_ptr<Blob> LLGL::Blob::CreateStrongRef (std::string && *container*) [static]

Creates a new Blob instance with a strong reference to the specified string container.

Parameters

in	*container*	Specifies the container whose data is to be moved into this Blob instance.

Returns

New instance of Blob that manages the specified container.

10.12.2.6 static std::unique_ptr<Blob> LLGL::Blob::CreateWeakRef (const void * *data*, std::size_t *size*) [static]

Creates a new Blob instance with a weak reference to the specified data.

Parameters

in	*data*	Pointer to the data that is to be referenced. This pointer must remain valid for the lifetime of this Blob instance.
in	*size*	Specifies the size (in bytes) of the data.

Returns

New instance of Blob that refers to the specified memory.

10.12.2.7 virtual const void* LLGL::Blob::GetData ( ) const [pure virtual]

Returns a constant pointer to the internal buffer.

10.12.2.8 virtual std::size_t LLGL::Blob::GetSize ( ) const [pure virtual]

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

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

- Blob.h

10.13 LLGL::Buffer Class Reference

Hardware buffer interface.

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

Inheritance diagram for LLGL::Buffer:

Public Member Functions

- ResourceType GetResourceType () const override final

  Returns ResourceType::Buffer.
- long GetBindFlags () const

  Returns the binding flags this buffer was created with.
- virtual BufferDescriptor GetDesc () const =0

  Queries a descriptor of this buffer.

Protected Member Functions

- Buffer (long bindFlags)

10.13.1 Detailed Description

Hardware buffer interface.

See also

RenderSystem::CreateBuffer

10.13.2 Constructor & Destructor Documentation

10.13.2.1 LLGL::Buffer::Buffer (long *bindFlags*) [protected]

10.13.3 Member Function Documentation

10.13.3.1 long LLGL::Buffer::GetBindFlags ( ) const [inline]

Returns the binding flags this buffer was created with.

See also

BufferDescriptor::bindFlags

10.13.3.2 virtual BufferDescriptor LLGL::Buffer::GetDesc ( ) const [pure virtual]

Queries a descriptor of this buffer.

Remarks

This function only queries the following attributes:

- `size`
- `bindFlags`
- `cpuAccessFlags`
- `miscFlags`

All other attributes (such as `vertexBuffer` etc.) cannot be queried by this function. Those attributes are set to the default value specified in BufferDescriptor.
The returned flags (such as `cpuAccessFlags` etc.) are not necessarily the same that were specified when the resource was created. They reflect the capabilities of the actual hardware buffer. For example, a buffer created with CPUAccessFlags::Read might return CPUAccessFlags::ReadWrite, if the renderer backend does not distinguish between different CPU access flags.

See also

BufferDescriptor
Texture::GetDesc

10.13.3.3 ResourceType LLGL::Buffer::GetResourceType ( ) const [final], [override], [virtual]

Returns ResourceType::Buffer.

Implements LLGL::Resource.

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

- Buffer.h

10.14 LLGL::BufferArray Class Reference

Hardware buffer container interface.

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

Inheritance diagram for LLGL::BufferArray:

Public Member Functions

- long GetBindFlags () const

  Returns a bitwise-OR combination of the binding flags of all sub-buffers.

Protected Member Functions

- BufferArray (long bindFlags)

10.14.1 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

10.14.2 Constructor & Destructor Documentation

10.14.2.1 LLGL::BufferArray::BufferArray (long *bindFlags*) [protected]

10.14.3 Member Function Documentation

10.14.3.1 long LLGL::BufferArray::GetBindFlags ( ) const [inline]

Returns a bitwise-OR combination of the binding flags of all sub-buffers.

See also

BufferDescriptor::bindFlags

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

- BufferArray.h

10.15 LLGL::BufferDescriptor Struct Reference

Hardware buffer descriptor structure.

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

Public Attributes

- std::uint64_t size = 0

  Buffer size (in bytes). This must not be larger than RenderingLimits::maxBufferSize. By default 0.
- std::uint32_t stride = 0

  Optional stride for structured buffers. By default 0.
- Format format = Format::Undefined

  Optioanl hardware buffer format. By default Format::Undefined.
- long bindFlags = 0

  These flags describe to which resource slots the buffer can be bound. By default 0.
- long cpuAccessFlags = 0

  CPU read/write access flags. By default 0.
- long miscFlags = 0

  Miscellaneous buffer flags. By default 0.
- ArrayView< VertexAttribute > vertexAttribs

  Specifies the list of vertex attributes.

10.15.1 Detailed Description

Hardware buffer descriptor structure.

See also

RenderSystem::CreateBuffer

10.15.2 Member Data Documentation

10.15.2.1 long LLGL::BufferDescriptor::bindFlags = 0

These flags describe to which resource slots the buffer can be bound. By default 0.

Remarks

When the buffer will be bound to a vertex buffer slot for instance, the BindFlags::VertexBuffer flag is required.

See also

BindFlags

10.15.2.2 long LLGL::BufferDescriptor::cpuAccessFlags = 0

CPU read/write access flags. By default 0.

Remarks

If this is 0 the buffer cannot be mapped between GPU and CPU memory space.

See also

CPUAccessFlags
RenderSystem::MapBuffer
RenderSystem::ReadBuffer
RenderSystem::WriteBuffer

10.15.2.3 Format LLGL::BufferDescriptor::format = Format::Undefined

Optioanl hardware buffer format. By default Format::Undefined.

Remarks

This is used for index buffers, typed buffers (e.g. `Buffer<uint4>` and `RWBuffer<float4>` in HLSL), and byte address buffers (i.e. `ByteAddressBuffer` and `RWByteAddressBuffer` in HLSL).
This field is ignored if the binding flags do not contain at least one of the following bits: BindFlags::IndexBuffer, BindFlags::Sampled, or BindFlags::Storage.
If the BindFlags::IndexBuffer bit is set, this must be either Format::R16UInt, Format::R32UInt, or Format::Undefined.
If Format::Undefined is specified and the BindFlags::IndexBuffer bit is set, only the secondary `SetIndexBuffer` function can be used in the CommandBuffer interface.
If Format::Undefined is specified and `stride` is zero, sampled and storage buffers will be interpreted as byte address buffers.

See also

BindFlags::IndexBuffer
CommandBuffer::SetIndexBuffer(Buffer&)

10.15.2.4 long LLGL::BufferDescriptor::miscFlags = 0

Miscellaneous buffer flags. By default 0.

Remarks

This can be used as a hint for the renderer how frequently the buffer will be updated.

See also

MiscFlags

10.15.2.5 std::uint64_t LLGL::BufferDescriptor::size = 0

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

Remarks

If `stride` is greater than zero, then `size` **must** be a multiple of `stride`.

See also

RenderingLimits::maxBufferSize
bindFlags

10.15.2.6 std::uint32_t LLGL::BufferDescriptor::stride = 0

Optional stride for structured buffers. By default 0.

Remarks

This is only used for Direct3D structured buffer, i.e. `StructuredBuffer`, `RWStructuredBuffer`, `AppendStructuredBuffer`, and `ConsumeStructuredBuffer` in HLSL.
If this is non-zero, the `format` attribute is ignored for sampled and storage buffers, i.e. buffers with the binding flags BindFlags::Sampled or BindFlags::Storage.

Note

If the buffer has the binding flag BindFlags::IndirectBuffer, this **must** be 0.
Only supported with: Direct3D 11, Direct3D 12.

See also

MiscFlags::Append
MiscFlags::Counter

10.15.2.7 ArrayView<VertexAttribute> LLGL::BufferDescriptor::vertexAttribs

Specifies the list of vertex attributes.

Remarks

This is only used for vertex buffers and ignored if `bindFlags` does not contain the BindFlags::VertexBuffer bit.

See also

BindFlags::VertexBuffer
VertexShaderAttributes::inputAttribs

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

- BufferFlags.h

10.16 LLGL::BufferViewDescriptor Struct Reference

Buffer view descriptor structure.

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

Public Member Functions

- BufferViewDescriptor ()=default
- BufferViewDescriptor (Format format, std::uint64_t offset=0, std::uint64_t size=Constants::wholeSize)

  Initializes the descriptor with all or only some of the components.

Public Attributes

- Format format = Format::Undefined

  Specifies the format of the buffer view. By default Format::Undefined.
- std::uint64_t offset = 0

  Specifies the memory offset (in bytes) into the buffer resource. By default 0.
- std::uint64_t size = Constants::wholeSize

  Specifies the memory offset (in bytes) into the buffer resource. By default `Constants::wholeSize`.

10.16.1 Detailed Description

Buffer view descriptor structure.

Remarks

Contains all information about format and memory range to create a buffer view that shares the data of another buffer resource.

See also

ResourceViewDescriptor::bufferView
RenderingFeatures::hasBufferViews

10.16.2 Constructor & Destructor Documentation

10.16.2.1 LLGL::BufferViewDescriptor::BufferViewDescriptor ( ) [default]

10.16.2.2 LLGL::BufferViewDescriptor::BufferViewDescriptor (Format *format*, std::uint64_t *offset* = 0, std::uint64_t *size* = Constants::wholeSize) [inline]

Initializes the descriptor with all or only some of the components.

10.16.3 Member Data Documentation

10.16.3.1 Format LLGL::BufferViewDescriptor::format = Format::Undefined

Specifies the format of the buffer view. By default Format::Undefined.

Remarks

If the buffer resource was created with a `stride` greater than zero, this must be Format::Undefined.

See also

BufferDescriptor::format
BufferDescriptor::stride

10.16.3.2 std::uint64_t LLGL::BufferViewDescriptor::offset = 0

Specifies the memory offset (in bytes) into the buffer resource. By default 0.

Remarks

If `size` is equal to `Constants::wholeSize`, the offset is ignored and the entire buffer resource will be occupied by this buffer view.
If `format` is Format::Undefined, this **must** be aligned to the `stride` the buffer resource was created with.
If `format` is *not* Format::Undefined, this **must** be aligned to the size of `format`.

See also

RenderingLimits::minConstantBufferAlignment
RenderingLimits::minSampledBufferAlignment
RenderingLimits::minStorageBufferAlignment
GetFormatAttribs

10.16.3.3 std::uint64_t LLGL::BufferViewDescriptor::size = Constants::wholeSize

Specifies the memory offset (in bytes) into the buffer resource. By default `Constants::wholeSize`.

Remarks

If `size` is `Constants::wholeSize`, then `offset` is ignored and the whole buffer range will be used.
If `size` is *not* `Constants::wholeSize` and `format` is Format::Undefined, this **must** be aligned to the `stride` the buffer resource was created with.
If `size` is *not* `Constants::wholeSize` and `format` is *not* Format::Undefined, this **must** be aligned to the size of `format`.

See also

RenderingLimits::minConstantBufferAlignment
RenderingLimits::minSampledBufferAlignment
RenderingLimits::minStorageBufferAlignment
GetFormatAttribs

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

- BufferFlags.h

10.17 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

- ~Canvas ()

  Releases the internal data.
- virtual void SetTitle (const UTF8String &title)=0

  Sets the canvas title as UTF-8 string.
- virtual UTF8String GetTitle () const =0

  Returns the canvas title as UTF16 string.
- virtual bool HasQuit () const

  Returns true if this canvas is in the 'Quit' state.
- bool AdaptForVideoMode (Extent2D *resolution, bool *fullscreen) override

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

  Processes the events for this canvas (i.e. touch input, key presses etc.).
- Display * FindResidentDisplay () const override final

  Always returns Display::GetPrimary.
- void AddEventListener (const std::shared_ptr< EventListener > &eventListener)

  Adds a new event listener to this canvas.
- void RemoveEventListener (const EventListener *eventListener)

  Removes the specified event listener from this canvas.
- void PostQuit ()

  Posts a 'Quit' event to all event listeners.

Static Public Member Functions

- static std::unique_ptr< Canvas > Create (const CanvasDescriptor &desc)

  Creates a platform specific instance of the Canvas interface.

Protected Member Functions

- Canvas ()

  Allocates the internal data.
- virtual void OnProcessEvents ()=0

  Called inside the "ProcessEvents" function after all event listeners received the same event.

10.17.1 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

10.17.2 Constructor & Destructor Documentation

10.17.2.1 LLGL::Canvas::~Canvas ( )

Releases the internal data.

10.17.2.2 LLGL::Canvas::Canvas ( ) [protected]

Allocates the internal data.

10.17.3 Member Function Documentation

10.17.3.1 bool LLGL::Canvas::AdaptForVideoMode (Extent2D * *resolution,* bool * *fullscreen*) [override], [virtual]

This default implementation ignores the video mode descriptor completely and always return false.

Implements LLGL::Surface.

10.17.3.2 void LLGL::Canvas::AddEventListener (const std::shared_ptr< EventListener > & *eventListener*)

Adds a new event listener to this canvas.

10.17.3.3 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

10.17.3.4 Display* LLGL::Canvas::FindResidentDisplay ( ) const [final], [override], [virtual]

Always returns Display::GetPrimary.

Implements LLGL::Surface.

10.17.3.5 virtual UTF8String LLGL::Canvas::GetTitle ( ) const [pure virtual]

Returns the canvas title as UTF16 string.

10.17.3.6 virtual bool LLGL::Canvas::HasQuit ( ) const [virtual]

Returns true if this canvas is in the 'Quit' state.

See also

PostQuit
ProcessEvents

10.17.3.7 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

10.17.3.8 void LLGL::Canvas::PostQuit ( )

Posts a 'Quit' event to all event listeners.

Remarks

If at least one event listener returns false within the `OnQuit` callback, the canvas will not quit. If all event listeners return true within the `OnQuit` callback, `ProcessEvents` will return false from now on.

See also

EventListener::OnQuit
ProcessEvents
HasQuit

10.17.3.9 bool LLGL::Canvas::ProcessEvents ( ) [final], [override], [virtual]

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

Returns

True, as long as the window can process events. Once the `PostQuit` function has set this canvas to the 'Quit' state, this function returns false.

See also

PostQuit

Implements LLGL::Surface.

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

Removes the specified event listener from this canvas.

10.17.3.11 virtual void LLGL::Canvas::SetTitle (const UTF8String & *title*) [pure virtual]

Sets the canvas title as UTF-8 string.

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

- Canvas.h

10.18 LLGL::CanvasDescriptor Struct Reference

Canvas descriptor structure.

```
#include <CanvasFlags.h>
```

Public Attributes

- UTF8String title

  Canvas title as UTF16 string.
- bool borderless = false

  Specifies whether the canvas is borderless. This is required for a fullscreen swap-chain.

10.18.1 Detailed Description

Canvas descriptor structure.

10.18.2 Member Data Documentation

10.18.2.1 bool LLGL::CanvasDescriptor::borderless = false

Specifies whether the canvas is borderless. This is required for a fullscreen swap-chain.

10.18.2.2 UTF8String LLGL::CanvasDescriptor::title

Canvas title as UTF16 string.

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

- CanvasFlags.h

10.19 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) }

10.19.1 Detailed Description

Command buffer clear flags.

See also

CommandBuffer::Clear

10.19.2 Member Enumeration Documentation

10.19.2.1 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

10.20 LLGL::ClearValue Struct Reference

Clear value structure for color, depth, and stencil clear operations.

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

Public Member Functions

- ClearValue ()=default
- ClearValue (const ClearValue &)=default
- ClearValue & operator= (const ClearValue &)=default
- ClearValue (const ColorRGBAf &color, float depth=1.0f, std::uint32_t stencil=0)

  Constructor for color, depth, and stencil values.
- ClearValue (float depth)

  Constructor for the depth value only.
- ClearValue (std::uint32_t stencil)

  Constructor for the stencil value only.
- ClearValue (float depth, std::uint32_t stencil)

  Constructor for the depth and stencil values.

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.

10.20.1 Detailed Description

Clear value structure for color, depth, and stencil clear operations.

See also

AttachmentClear::clearValue
TextureDescriptor::clearValue

10.20.2 Constructor & Destructor Documentation

10.20.2.1 LLGL::ClearValue::ClearValue ( ) [default]

10.20.2.2 LLGL::ClearValue::ClearValue (const ClearValue &) [default]

10.20.2.3 LLGL::ClearValue::ClearValue (const ColorRGBAf & *color*, float *depth* = 1.0f, std::uint32_t *stencil* = 0) [inline]

Constructor for color, depth, and stencil values.

10.20.2.4 LLGL::ClearValue::ClearValue (float *depth*) [inline]

Constructor for the depth value only.

10.20.2.5 LLGL::ClearValue::ClearValue (std::uint32_t *stencil*) [inline]

Constructor for the stencil value only.

10.20.2.6 LLGL::ClearValue::ClearValue (float *depth,* std::uint32_t *stencil*) [inline]

Constructor for the depth and stencil values.

10.20.3 Member Function Documentation

10.20.3.1 ClearValue& LLGL::ClearValue::operator= (const ClearValue &) [default]

10.20.4 Member Data Documentation

10.20.4.1 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).

10.20.4.2 float LLGL::ClearValue::depth = 1.0f

Specifies the clear value to clear a depth attachment. By default 1.0.

10.20.4.3 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

10.21 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 constexpr std::size_t components = N

  Specifies the number of vector components.

10.21.1 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.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 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

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

Copy constructor.

10.21.2.3 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.

10.21.2.4 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.

10.21.3 Member Function Documentation

10.21.3.1 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.

10.21.3.2 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.

10.21.3.3 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.

10.21.3.4 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.

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

Returns the negation of this color.

10.21.3.6 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.

10.21.3.7 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.

10.21.3.8 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.

10.21.3.9 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).

10.21.3.10 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).

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

Returns a pointer to the first element of this vector.

10.21.3.12 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.

10.21.4 Member Data Documentation

10.21.4.1 template<typename T, std::size_t N> constexpr 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

10.22 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 constexpr std::size_t components = 3

  Specifies the number of color components.

10.22.1 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.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 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

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

Copy constructor.

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

Constructs all attributes with the specified scalar value.

10.22.2.4 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).

10.22.2.5 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.

10.22.3 Member Function Documentation

10.22.3.1 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.

10.22.3.2 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.

10.22.3.3 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.

10.22.3.4 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.

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

Returns the negation of this color.

10.22.3.6 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.

10.22.3.7 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.

10.22.3.8 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.

10.22.3.9 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).

10.22.3.10 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).

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

Returns a pointer to the first element of this color.

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

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

10.22.3.13 template<typename T > Color<T, 4> LLGL::Color< T, 3u >::ToRGBA ( ) const [inline]

Returns this RGB color as RGBA color.

10.22.4 Member Data Documentation

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

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

Specifies the number of color components.

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

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

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

- ColorRGB.h

10.23 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 constexpr std::size_t components = 4

  Specifies the number of color components.

10.23.1 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.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 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

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

Copy constructor.

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

Constructs all attributes with the specified scalar value.

10.23.2.4 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

10.23.2.5 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

10.23.2.6 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).

10.23.2.7 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.

10.23.3 Member Function Documentation

10.23.3.1 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.

10.23.3.2 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.

10.23.3.3 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.

10.23.3.4 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.

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

Returns the negation of this color.

10.23.3.6 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.

10.23.3.7 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.

10.23.3.8 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.

10.23.3.9 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).

10.23.3.10 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).

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

Returns a pointer to the first element of this color.

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

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

10.23.3.13 template<typename T > Color<T, 3> LLGL::Color< T, 4u >::ToRGB ( ) const [inline]

Returns this RGBA color as RGB color.

10.23.4 Member Data Documentation

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

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

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

Specifies the number of color components.

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

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

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

- ColorRGBA.h

10.24 LLGL::CommandBuffer Class Reference

Command buffer interface used for storing and encoding GPU commands.

```
#include <CommandBuffer.h>
```

Inheritance diagram for LLGL::CommandBuffer:

Public Member Functions

- virtual void Begin ()=0

  Begins with the encoding (also referred to as "recording") of this command buffer.
- virtual void End ()=0

  Ends the encoding (also referred to as "recording") of this command buffer.
- virtual void Execute (CommandBuffer &deferredCommandBuffer)=0

  Executes the specified deferred command buffer.
- virtual void UpdateBuffer (Buffer &dstBuffer, std::uint64_t dstOffset, const void *data, std::uint16_t 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 CopyBufferFromTexture (Buffer &dstBuffer, std::uint64_t dstOffset, Texture &srcTexture, const TextureRegion &srcRegion, std::uint32_t rowStride=0, std::uint32_t layerStride=0)=0

  Encodes a buffer copy command that blits data from a source texture.
- virtual void FillBuffer (Buffer &dstBuffer, std::uint64_t dstOffset, std::uint32_t value, std::uint64_t fill↩Size=Constants::wholeSize)=0

  Fills the destination buffer with copies of the specified 32-bit value.
- virtual void CopyTexture (Texture &dstTexture, const TextureLocation &dstLocation, Texture &srcTexture, const TextureLocation &srcLocation, const Extent3D &extent)=0

  Encodes a texture copy command for the specified texture regions.

- virtual void CopyTextureFromBuffer (Texture &dstTexture, const TextureRegion &dstRegion, Buffer &srcBuffer, std::uint64_t srcOffset, std::uint32_t rowStride=0, std::uint32_t layerStride=0)=0

  Encodes a texture copy command that blits data from a source buffer.
- virtual void GenerateMips (Texture &texture)=0

  Generates all MIP-maps for the specified texture.
- virtual void GenerateMips (Texture &texture, const TextureSubresource &subresource)=0

  Generates a range of MIP-maps for the specified texture.
- virtual void SetViewport (const Viewport &viewport)=0

  Sets a single viewport.
- virtual void SetViewports (std::uint32_t numViewports, const Viewport *viewports)=0

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

  Sets a single scissor rectangle.
- virtual void SetScissors (std::uint32_t numScissors, const Scissor *scissors)=0

  Sets an array of scissor rectangles, but only if the scissor test was enabled in the previously set graphics pipeline (otherwise, this function has no effect).
- virtual void SetVertexBuffer (Buffer &buffer)=0

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

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

  Sets the active index buffer for subsequent drawing operations.
- virtual void SetIndexBuffer (Buffer &buffer, const Format format, std::uint64_t offset=0)=0

  Sets the active index buffer for subsequent drawing operations with a dynamic format and optional buffer offset.
- virtual void SetResourceHeap (ResourceHeap &resourceHeap, std::uint32_t descriptorSet=0, const PipelineBindPoint bindPoint=PipelineBindPoint::Undefined)=0

  Binds the specified resource heap to the respective pipeline.
- virtual void SetResource (Resource &resource, std::uint32_t slot, long bindFlags, long stageFlags=StageFlags::AllStages)=0

  Sets the specified resource to a binding slot.
- virtual void ResetResourceSlots (const ResourceType resourceType, std::uint32_t firstSlot, std::uint32_t numSlots, long bindFlags, long stageFlags=StageFlags::AllStages)=0

  Resets the binding slots for the specified resources.
- virtual void BeginRenderPass (RenderTarget &renderTarget, const RenderPass *renderPass=nullptr, std::uint32_t numClearValues=0, const ClearValue *clearValues=nullptr)=0

  Begins with a new render pass.
- virtual void EndRenderPass ()=0

  Ends the current render pass.
- virtual void Clear (long flags, const ClearValue &clearValue={})=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 SetPipelineState (PipelineState &pipelineState)=0

  Sets the active graphics or compute pipeline state.
- virtual void SetBlendFactor (const ColorRGBAf &color)=0

  Sets the dynamic pipeline state for blending factors.
- virtual void SetStencilReference (std::uint32_t reference, const StencilFace stencilFace=StencilFace::FrontAndBack)=0

  Sets the dynamic pipeline state for stencil reference values.
- virtual void SetUniform (UniformLocation location, const void *data, std::uint32_t dataSize)=0

  Sets the value of a single uniform (a.k.a. shader constant) in the shader program that is currently bound.

- virtual void SetUniforms (UniformLocation location, std::uint32_t count, const void *data, std::uint32_t dataSize)=0

  Sets the value of multiple uniforms (a.k.a. shader constants) in the shader program that is currently bound.
- virtual void BeginQuery (QueryHeap &queryHeap, std::uint32_t query=0)=0

  Begins a query of the specified query heap.
- virtual void EndQuery (QueryHeap &queryHeap, std::uint32_t query=0)=0

  Ends the specified query.
- virtual void BeginRenderCondition (QueryHeap &queryHeap, std::uint32_t query=0, const RenderConditionMode mode=RenderConditionMode::Wait)=0

  Begins conditional rendering with the specified query object.
- virtual void EndRenderCondition ()=0

  Ends the current render condition.
- virtual void BeginStreamOutput (std::uint32_t numBuffers, Buffer *const *buffers)=0

  Begins a stream-output section for subsequent draw calls.
- virtual void EndStreamOutput ()=0

  Ends the current stream-output.
- virtual void Draw (std::uint32_t numVertices, std::uint32_t firstVertex)=0

  Draws the specified amount of primitives from the currently set vertex buffer.
- virtual void DrawIndexed (std::uint32_t numIndices, std::uint32_t firstIndex)=0
- virtual void DrawIndexed (std::uint32_t numIndices, std::uint32_t firstIndex, std::int32_t vertexOffset)=0

  Draws the specified amount of primitives from the currently set vertex- and index buffers.
- virtual void DrawInstanced (std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances)=0
- virtual void DrawInstanced (std::uint32_t numVertices, std::uint32_t firstVertex, std::uint32_t numInstances, std::uint32_t firstInstance)=0

  Draws the specified amount of instances of primitives from the currently set vertex buffer.
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset)=0
- virtual void DrawIndexedInstanced (std::uint32_t numIndices, std::uint32_t numInstances, std::uint32_t firstIndex, std::int32_t vertexOffset, std::uint32_t firstInstance)=0

  Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.
- virtual void DrawIndirect (Buffer &buffer, std::uint64_t offset)=0

  Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.
- virtual void DrawIndirect (Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0

  Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.
- virtual void DrawIndexedIndirect (Buffer &buffer, std::uint64_t offset)=0

  Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.
- virtual void DrawIndexedIndirect (Buffer &buffer, std::uint64_t offset, std::uint32_t numCommands, std::uint32_t stride)=0

  Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.
- virtual void Dispatch (std::uint32_t numWorkGroupsX, std::uint32_t numWorkGroupsY, std::uint32_t numWorkGroupsZ)=0

  Dispatches a compute command.
- virtual void DispatchIndirect (Buffer &buffer, std::uint64_t offset)=0

  Dispatches a compute command with an unknown amount of thread grounds.
- virtual void PushDebugGroup (const char *name)=0

  Pushes the specified name onto a stack of group strings that is used for debug reports.
- virtual void PopDebugGroup ()=0
- virtual void SetGraphicsAPIDependentState (const void *stateDesc, std::size_t stateDescSize)=0

  Sets a few low-level graphics API dependent states.

Protected Member Functions

- CommandBuffer ()=default

10.24.1 Detailed Description

Command buffer interface used for storing and encoding GPU commands.

Remarks

This is the main interface to encode graphics, compute, and blit commands to be submitted to the GPU. All states that can be changed with a setter function are not persistent across several encoding sections. Before any command can be encoded, the command buffer must be set into encode mode, which is done by the CommandBuffer::Begin function.

See also

RenderSystem::CreateCommandBuffer

10.24.2 Constructor & Destructor Documentation

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

10.24.3 Member Function Documentation

10.24.3.1 virtual void LLGL::CommandBuffer::Begin ( ) [pure virtual]

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

Remarks

All functions of the CommandBuffer interface must be used between a call to `Begin` and `End`. This function also resets all previously encoded commands.

See also

End

10.24.3.2 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

10.24.3.3 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

10.24.3.4 virtual void LLGL::CommandBuffer::BeginRenderPass (RenderTarget & *renderTarget*, const RenderPass * *renderPass* = nullptr, std::uint32_t *numClearValues* = 0, const ClearValue * *clearValues* = nullptr) [pure virtual]

Begins with a new render pass.

Parameters

in	*renderTarget*	Specifies the render target in which the subsequent draw operations will be stored.

Parameters

in	*renderPass*	Specifies an optional render pass object. If this is null, the default render pass for the specified render target will be used. This render pass object must be compatible with the render pass object the specified render target was created with.
in	*numClearValues*	Specifies the number of clear values that are specified in the `clearValues` parameter. This *should* be greater than or equal to the number of render pass attachments whose load operation (i.e. AttachmentFormatDescriptor::loadOp) is set to AttachmentLoadOp::Clear. Otherwise, the following default values are used: (0, 0, 0, 0) for color, 1 for depth, 0 for stencil.
in	*clearValues*	Optional pointer to the array of clear values. If `numClearValues` is not zero, this must be a valid pointer to an array of at least `numClearValues` entries. Each entry in the array is used to clear the attachment whose load operation is set to AttachmentLoadOp::Clear, where the depth attachment (i.e. RenderPassDescriptor::depthAttachment) and the stencil attachment (i.e. RenderPassDescriptor::stencilAttachment) are combined and appear as the last entry.

Remarks

This function starts a new render pass section and must be ended with the `EndRenderPass` function. The following example shows how to use a render pass to clear a render target with two color attachments and a depth-stencil attachment:

```
LLGL::ClearValue myClearValues[3];

// Set clear values for color attachments 0 and 1
myClearValues[0].color = { 1, 0, 0, 1 };
myClearValues[1].color = { 0, 1, 0, 1 };

// Set clear values for depth-stencil attachment
myClearValues[2].depth   = 1.0f;
myClearValues[2].stencil = 0;

// Begin render pass and clear render target
myCmdBuffer->BeginRenderPass(*myRenderTarget, *myRenderPass, 3, myClearValues);
{
    // Draw scene ...
}
myCmdBuffer->EndRenderPass();
```

The following commands **must** only be used **inside** a render pass section:

- Drawing commands (i.e. `Draw`, `DrawInstanced`, `DrawIndexed`, `DrawIndexedInstanced`, `DrawIndirect` and `DrawIndexedIndirect`).
- Clear attachment commands (i.e. `Clear` and `ClearAttachments`).
- Query block (i.e. `BeginQuery` and `EndQuery`).
- Conditional render block (i.e. `BeginRenderCondition` and `EndRenderCondition`).
- Stream-output block (i.e. `BeginStreamOutput` and `EndStreamOutput`).

The following commands **must** only be used **outside** a render pass section:

- Dispatch compute commands (i.e. `Dispatch` and `DispatchIndirect`).

The following commands *can* be used both inside and outside a render pass section but are *recommended* to be used only **outside** a render pass section to avoid potential performace penalties:

- Copy commands (i.e. `UpdateBuffer`, `CopyBuffer`, and `CopyTexture`).
- MIP-map generation commands (i.e. `GenerateMips`).

See also

RenderSystem::CreateRenderPass
RenderSystem::CreateRenderTarget
RenderTargetDescriptor::renderPass
AttachmentFormatDescriptor::loadOp
EndRenderPass

10.24.3.5 virtual void LLGL::CommandBuffer::BeginStreamOutput (std::uint32_t *numBuffers*, Buffer *const * *buffers*) [pure virtual]

Begins a stream-output section for subsequent draw calls.

Parameters

in	*numBuffers*	Specifies the number of stream-output buffers. This must be in the range [1, RenderingLimits::maxStreamOutputs].
in	*buffers*	Array to the stream-output buffers. This must be a valid pointer to an array of numBuffers buffer objects. Each of these buffers must have been created with the binding flag BindFlags::StreamOutputBuffer.

Remarks

This must only be called if a graphics pipeline is currently bound.

See also

EndStreamOutput
SetGraphicsPipeline
RenderingFeatures::hasStreamOutputs
RenderingLimits::maxStreamOutputs

10.24.3.6 virtual void LLGL::CommandBuffer::Clear (long *flags*, const ClearValue & *clearValue* = { }) [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 specified by clearValue.
in	*clearValue*	Specifies the value to which the attachments will be cleared.

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
ClearAttachments
BeginRenderPass

10.24.3.7 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

10.24.3.8 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,out	*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 the source offset (in bytes) at which the source buffer is to be read from. This offset plus the size (i.e. `srcOffset + size`) must be less than or equal to the size of the source buffer.
in	*size*	Specifies the size of the buffer region to copy.

Remarks

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

10.24.3.9 virtual void LLGL::CommandBuffer::CopyBufferFromTexture (Buffer & *dstBuffer*, std::uint64_t *dstOffset*, Texture & *srcTexture*, const TextureRegion & *srcRegion*, std::uint32_t *rowStride* = 0, std::uint32_t *layerStride* = 0) [pure virtual]

Encodes a buffer copy command that blits data from a source texture.

Parameters

in,out	*dstBuffer*	Specifies the destination buffer whose data is to be updated. This buffer must have been created with the binding flag BindFlags::CopyDst.
in	*dstOffset*	Specifies the destination offset (in bytes) at which the source buffer is to be updated. This **must** be a multiple of 4.
in	*srcTexture*	Specifies the source texture whose data is to be read from. This texture must have been created with the binding flag BindFlags::CopySrc and its format **must not** be compressed (see FormatFlags::IsCompressed) or packed (see FormatFlags::IsPacked).
in	*srcRegion*	Specifies the source region where the texture is to be read from. Note that the `numMipLevels` attribute of this parameter **must** be 1.
in	*rowStride*	Specifies an optional stride (in bytes) per row in the source buffer. By default 0.
in	*layerStride*	Specifies an optional stride (in bytes) per layer in the source buffer. This **must** be a multiple of `rowStride`. If `rowStride` is zero, then `layerStride` must also be zero. By default 0.

Remarks

This is called "copy buffer from texture" instead of "copy texture to buffer" to be uniform with the notation `buffer := texture`, or `memcpy(destination, source, size)`.
For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.
Further performance penalties can be introduced if `rowStride` is not aligned to the respective rendering API restrictions:

- Direct3D 12: `rowStride` **should** be a multiple of 256.
- Metal: `rowStride` **should** be less than or equal to 32767 multiplied by the source texture's format size.

If `rowStride` is 0, the source data is considered to be tightly packed for each array layer and the required alignment is managed automatically.
If `rowStride` is not 0, it **must** be greater than or equal to the size (in bytes) of each row in the texture region with respect to the texture's format.
The same rules of `rowStride` also apply to `layerStride`.

See also

CopyTextureFromBuffer
GetMemoryFootprint

10.24.3.10 virtual void LLGL::CommandBuffer::CopyTexture (Texture & *dstTexture*, const TextureLocation & *dstLocation*, Texture & *srcTexture*, const TextureLocation & *srcLocation*, const Extent3D & *extent*) [pure virtual]

Encodes a texture copy command for the specified texture regions.

Parameters

in,out	*dstTexture*	Specifies the destination texture whose data is to be updated.
in	*dstLocation*	Specifies the destination location, including MIP-map level and offset. Its offset plus the extent (i.e. `dstLocation.offset + extent`) must be less than or equal to the size of the destination texture.
in	*srcTexture*	Specifies the source texture whose data is to be read from.

Parameters

in	*srcLocation*	Specifies the source location, including MIP-map level and offset. Its offset plus the extent (i.e. `srcLocation.offset + extent`) must be less than or equal to the size of the source texture.
in	*extent*	Specifies the extent of the texture region to copy (see TextureDescriptor::extent). For this function, the extent also includes the array layers, i.e. `y` component for 1D arrays (TextureType::Texture1DArray), and `z` component for 2D and cube arrays (TextureType::Texture2DArray and TextureType::TextureCubeArray).

Remarks

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

10.24.3.11 virtual void LLGL::CommandBuffer::CopyTextureFromBuffer (Texture & *dstTexture*, const TextureRegion & *dstRegion*, Buffer & *srcBuffer*, std::uint64_t *srcOffset*, std::uint32_t *rowStride* = `0`, std::uint32_t *layerStride* = `0`) `[pure virtual]`

Encodes a texture copy command that blits data from a source buffer.

Parameters

in,out	*dstTexture*	Specifies the destination texture whose data is to be updated. This texture must have been created with the binding flag BindFlags::CopyDst and its format **must not** be compressed (see FormatFlags::IsCompressed) or packed (see FormatFlags::IsPacked).
in	*dstRegion*	Specifies the destination region where the texture is to be updated. Note that the `numMipLevels` attribute of this parameter **must** be 1.
in	*srcBuffer*	Specifies the source buffer whose data is to be read from. This buffer must have been created with the binding flag BindFlags::CopySrc.
in	*srcOffset*	Specifies the source offset (in bytes) at which the source buffer is to be read from. This **must** be a multiple of 4.
in	*rowStride*	Specifies an optional stride (in bytes) per row in the source buffer. By default 0.
in	*layerStride*	Specifies an optional stride (in bytes) per layer in the source buffer. This **must** be a multiple of `rowStride`. If `rowStride` is zero, then `layerStride` must also be zero. By default 0.

Remarks

This is called "copy texture from buffer" instead of "copy buffer to texture" to be uniform with the notation `texture := buffer`, or `memcpy(destination, source, size)`.
For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.
Further performance penalties can be introduced if `rowStride` is not aligned to the respective rendering API restrictions:

- Direct3D 12: `rowStride` **should** be a multiple of 256.
- Metal: `rowStride` **should** be less than or equal to 32767 multiplied by the destination texture's format size.

If `rowStride` is 0, the source data is considered to be tightly packed for each array layer and the required alignment is managed automatically.

If `rowStride` is not 0, it **must** be greater than or equal to the size (in bytes) of each row in the texture region with respect to the texture's format.
The same rules of `rowStride` also apply to `layerStride`.

See also

CopyBufferFromTexture
GetMemoryFootprint

10.24.3.12 virtual void LLGL::CommandBuffer::Dispatch (std::uint32_t *numWorkGroupsX*, std::uint32_t *numWorkGroupsY*, std::uint32_t *numWorkGroupsZ*) [pure virtual]

Dispatches a compute command.

Parameters

in	*numWorkGroupsX*	Specifies the number of worker thread groups in the X-dimension.
in	*numWorkGroupsY*	Specifies the number of worker thread groups in the Y-dimension.
in	*numWorkGroupsZ*	Specifies the number of worker thread groups in the Z-dimension.

See also

SetComputePipeline
RenderingLimits::maxComputeShaderWorkGroups

10.24.3.13 virtual void LLGL::CommandBuffer::DispatchIndirect (Buffer & *buffer*, std::uint64_t *offset*) [pure virtual]

Dispatches a compute command with an unknown amount of thread grounds.

Parameters

in	*buffer*	Specifies the buffer from which the dispatch command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag.
in	*offset*	Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4.

See also

DispatchIndirectArguments

10.24.3.14 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`.

10.24.3.15 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)

10.24.3.16 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.

Note

For the Metal renderer, the parameter `vertexOffset` is ignored when tessellation is enabled.

10.24.3.17 virtual void LLGL::CommandBuffer::DrawIndexedIndirect (Buffer & *buffer*, std::uint64_t *offset*) [pure virtual]

Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.

Parameters

in	*buffer*	Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag.
in	*offset*	Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4.

See also

DrawIndexedIndirectArguments

10.24.3.18 virtual void LLGL::CommandBuffer::DrawIndexedIndirect (Buffer & *buffer*, std::uint64_t *offset*, std::uint32_t *numCommands*, std::uint32_t *stride*) [pure virtual]

Draws an unknown amount of instances of primitives whose indexed draw command arguments are taken from a buffer object.

Parameters

in	*buffer*	Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag.
in	*offset*	Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4.
in	*numCommands*	Specifies the number of draw commands that are to be taken from the argument buffer.
in	*stride*	Specifies the stride (in bytes) betweeen consecutive sets of arguments, which is commonly greater than or euqal to `sizeof(DrawIndexedIndirectArguments)`. This stride must be a multiple of 4.

Remarks

This is also known as a "multi draw command" which is only natively supported by OpenGL and Vulkan. For other rendering APIs, the recording of multiple draw commands is emulated with a simple loop, which is equivalent to the following example:

```
while (numCommands-- > 0)
{
    DrawIndexedIndirect(buffer, offset);
    offset += stride;
}
```

See also

DrawIndexedIndirectArguments

10.24.3.19 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)

10.24.3.20 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)

10.24.3.21 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`.
For the Metal renderer, the parameter `vertexOffset` is ignored when tessellation is enabled.

See also

RenderingFeatures::hasInstancing
RenderingFeatures::hasOffsetInstancing

10.24.3.22 virtual void LLGL::CommandBuffer::DrawIndirect (Buffer & *buffer*, std::uint64_t *offset*) [pure virtual]

Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.

Parameters

in	*buffer*	Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag.
in	*offset*	Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4.

See also

DrawIndirectArguments
RenderingFeatures::hasIndirectDrawing

10.24.3.23 virtual void LLGL::CommandBuffer::DrawIndirect (Buffer & *buffer*, std::uint64_t *offset*, std::uint32_t *numCommands*, std::uint32_t *stride*) [pure virtual]

Draws an unknown amount of instances of primitives whose draw command arguments are taken from a buffer object.

Parameters

in	*buffer*	Specifies the buffer from which the draw command arguments are taken. This buffer must have been created with the BufferFlags::IndirectBinding flag.
in	*offset*	Specifies an offset within the argument buffer from which the arguments are to be taken. This offset must be a multiple of 4.
in	*numCommands*	Specifies the number of draw commands that are to be taken from the argument buffer.
in	*stride*	Specifies the stride (in bytes) betweeen consecutive sets of arguments, which is commonly greater than or euqal to `sizeof(DrawIndirectArguments)`. This stride must be a multiple of 4.

Remarks

This is also known as a "multi draw command" which is only natively supported by OpenGL and Vulkan. For other rendering APIs, the recording of multiple draw commands is emulated with a simple loop, which is equivalent to the following example:

```
while (numCommands-- > 0)
{
    DrawIndirect(buffer, offset);
    offset += stride;
}
```

See also

DrawIndirectArguments
RenderingFeatures::hasIndirectDrawing

10.24.3.24 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)

10.24.3.25 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).

See also

RenderingFeatures::hasInstancing
RenderingFeatures::hasOffsetInstancing

10.24.3.26 virtual void LLGL::CommandBuffer::End ( ) [pure virtual]

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

Remarks

After this call, the command buffer can be submitted to the CommandQueue or executed by a primary command buffer.

See also

Begin
Execute
CommandQueue::Submit(CommandBuffer&)

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

Ends the specified query.

See also

BeginQuery

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

Ends the current render condition.

See also

BeginRenderCondition

10.24.3.29 virtual void LLGL::CommandBuffer::EndRenderPass ( ) [pure virtual]

Ends the current render pass.

See also

BeginRenderPass

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

Ends the current stream-output.

See also

BeginStreamOutput

10.24.3.31 virtual void LLGL::CommandBuffer::Execute (CommandBuffer & *deferredCommandBuffer*) [pure virtual]

Executes the specified deferred command buffer.

Parameters

in	*deferredCommandBuffer*	Specifies the deferred command buffer which is meant to be executed. This command buffer must have been created with the CommandBufferFlags::Secondary flag.

Remarks

This function can only be used by primary command buffers, i.e. command buffers that have not been created with the flag CommandBufferFlags::Secondary.

See also

CommandBufferFlags

Todo Incomplete for: D3D12, Vulkan, Metal.

10.24.3.32 virtual void LLGL::CommandBuffer::FillBuffer (Buffer & *dstBuffer*, std::uint64_t *dstOffset*, std::uint32_t *value*, std::uint64_t *fillSize* = Constants::wholeSize) [pure virtual]

Fills the destination buffer with copies of the specified 32-bit value.

Parameters

in,out	*dstBuffer*	Specifies the destination buffer whose data is to be updated. This command works with all kinds of buffers, but for performance reasons it is recommended to create this buffer with the binding flag BindFlags::Storage. Otherwise, an intermediate buffer might be created and copied by LLGL.
in	*dstOffset*	Specifies the destination offset (in bytes) at which the destination buffer is to be updated.
in	*value*	Specifies the 32-bit value to fill the buffer with.
in	*fillSize*	Specifies the fill size (in bytes) of the buffer region. This **must** be a multiple of 4. By default Constants::wholeSize. If this is equal to `Constants::wholeSize`, `dstOffset` is ignored and the entire buffer will be filled.

Remarks

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

10.24.3.33 virtual void LLGL::CommandBuffer::GenerateMips (Texture & *texture*) [pure virtual]

Generates all MIP-maps for the specified texture.

Parameters

in,out	*texture*	Specifies the texture whose MIP-maps are to be generated. This texture must have been created with the binding flags BindFlags::Sampled and BindFlags::ColorAttachment.

Remarks

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

See also

GenerateMips(Texture&, const TextureSubresource&)

10.24.3.34 virtual void LLGL::CommandBuffer::GenerateMips (Texture & *texture*, const TextureSubresource & *subresource*) [pure virtual]

Generates a range of MIP-maps for the specified texture.

Parameters

in,out	*texture*	Specifies the texture whose MIP-maps are to be generated. This texture must have been created with the binding flags BindFlags::Sampled and BindFlags::ColorAttachment.
in	*subresource*	Specifies the texture subresource, i.e. the range of MIP-maps that are to be updated.

Remarks

For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.
This function guarantees to generate only the MIP-maps in the specified range (specified by `subresource`). However, this function *may* introduce a performance penalty compared to generating the full MIP chain if texture views are not natively supported by the backend. It is therefore recommended to use this function only if the range of MIP-maps is significantly smaller than the entire MIP chain, e.g. only a single slice of a large 2D array texture, and use the primary `GenerateMips` function otherwise.

See also

GenerateMips(Texture&)
RenderingFeatures::hasTextureViews

10.24.3.35 virtual void LLGL::CommandBuffer::PopDebugGroup ( ) [pure virtual]

See also

PushDebugGroup

10.24.3.36 virtual void LLGL::CommandBuffer::PushDebugGroup (const char * *name*) [pure virtual]

Pushes the specified name onto a stack of group strings that is used for debug reports.

Parameters

in	*name*	Pointer to a null terminated string that specifies the name. This **must not** be null!

Remarks

Here is a usage example:

```
myCmdBuffer->PushDebugGroup("Shadow Map Pass");
myCmdBuffer->BeginRenderPass(...);
// render shadow map ...
myCmdBuffer->EndRenderPass();
myCmdBuffer->PopDebugGroup();

myCmdBuffer->PushDebugGroup("Final Scene Pass");
myCmdBuffer->BeginRenderPass(...);
// render final scene ...
myCmdBuffer->EndRenderPass();
myCmdBuffer->PopDebugGroup();
```

Note

Only supported in debug mode or when the debug layer is enabled. Otherwise, the function has no effect.

See also

PopDebugGroup
RenderSystemChild::SetName

10.24.3.37 virtual void LLGL::CommandBuffer::ResetResourceSlots (const ResourceType *resourceType*, std::uint32_t *firstSlot*, std::uint32_t *numSlots*, long *bindFlags*, long *stageFlags* = StageFlags::AllStages) [pure virtual]

Resets the binding slots for the specified resources.

Remarks

This should be called when a resource is currently bound as shader output and will be bound as shader input for the next draw or compute commands.

Parameters

in	*resourceType*	Specifies the type of resources to unbind.
in	*firstSlot*	Specifies the first binding slot beginning with zero. This must be zero for the following resource types: ResourceType::IndexBuffer, ResourceType::StreamOutputBuffer.
in	*numSlots*	Specifies the number of binding slots to reset. If this is zero, the function has no effect.
in	*bindFlags*	Specifies which kind of binding slots to reset. This can be a bitwise OR combinations of the BindFlags entries. To reset a vertex buffer slot for instance, it must contain the BindFlags::VertexBuffer flag.
in	*stageFlags*	Specifies which shader stages are affected. This can be a bitwise OR combination of the StageFlags entries. By default StageFlags::AllStages.

Remarks

If direct resource binding is not supported by the render system, this function has no effect.

Note

Only supported with: OpenGL, Direct3D 11, Metal.

See also

BindFlags
StageFlags
RenderingFeatures::hasDirectResourceBinding

10.24.3.38 virtual void LLGL::CommandBuffer::SetBlendFactor (const ColorRGBAf & *color*) [pure virtual]

Sets the dynamic pipeline state for blending factors.

Parameters

in	*color*	Specifies the blending factors for each color component. The default value is `{ 1, 1, 1, 1 }`.

Remarks

This is only used for the following blending operations:

- BlendOp::BlendFactor
- BlendOp::InvBlendFactor

This must only be used if the currently bound graphics pipeline state was created with `blendFactor↩Dynamic` set to true. Otherwise, the behavior is undefined.

See also

BlendDescriptor::blendFactorDynamic

10.24.3.39 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 internal buffer binding slots. Here is a usage example:

```
LLGL::MetalDependentStateDescriptor myMetalStateDesc;
myMetalStateDesc.tessFactorBufferSlot = 1;
myCmdBuffer->SetGraphicsAPIDependentState(&myMetalStateDesc, sizeof(myMetalStateDesc));
```

Note

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

See also

MetalDependentStateDescriptor

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

Sets the active index buffer for subsequent drawing operations.

Parameters

in	*buffer*	Specifies the index buffer to set. This buffer must have been created with the binding flag BindFlags::IndexBuffer and its content **must not** be uninitialized. For this version of `SetIndexBuffer`, the index buffer must also be created with `indexFormat` set to either Format::R16UInt or Format::R32UInt.

Remarks

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

See also

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
DrawIndexed
DrawIndexedInstanced
BufferDescriptor::indexFormat

10.24.3.41 virtual void LLGL::CommandBuffer::SetIndexBuffer (Buffer & *buffer*, const Format *format*, std::uint64_t *offset* = 0) [pure virtual]

Sets the active index buffer for subsequent drawing operations with a dynamic format and optional buffer offset.

Parameters

in	*buffer*	Specifies the index buffer to set. This buffer must have been created with the binding flag BindFlags::IndexBuffer and its content **must not** be uninitialized.
in	*format*	Specifies the format of each index in the buffer. This must be either Format::R16UInt or Format::R32UInt.
in	*offset*	Specifies an optional offset (in bytes) where to start reading the index buffer. By default 0. This has the same effect as setting the `firstIndex` argument in any `DrawIndexed` or `DrawIndexedInstanced` function, except that this offset is byte aligned.

Remarks

The alternative version of this function uses merely the index format that was specified when the buffer was created.

See also

BufferDescriptor::IndexBuffer::format
SetIndexBuffer(Buffer&)

10.24.3.42 virtual void LLGL::CommandBuffer::SetPipelineState (PipelineState & *pipelineState*) [pure virtual]

Sets the active graphics or compute pipeline state.

Parameters

in	*pipelineState*	Specifies the pipeline state which is to be bound for subsequent draw or compute commands.

Remarks

A **graphics pipeline state** will set all blending-, rasterizer-, depth-, stencil-, and shader states. A valid graphics pipeline state must always be set before any drawing operation can be performed, and a graphics pipeline state **can** be set **inside** and **outside** a render pass section.
A **compute pipeline state** will set shader states for dispatch compute commands. A valid compute pipeline state must always be set before any dispatch compute operation cam ne performed, and a compute pipeline state **must** be set **outside** a render pass section.

```
// Set compute pipeline state and perform compute commands
myCmdBuffer->SetPipelineState(*myComputePipeline);
myCmdBuffer->Dispatch(...);

// Start render pass section
myCmdBuffer->BeginRenderPass(...);
{
    // Set graphics pipeline state and perform drawing operations
    myCmdBuffer->SetPipelineState(*myGraphicsPipeline);
    myCmdBuffer->Draw(...);
}
myCmdBuffer->EndRenderPass();
```

See also

RenderSystem::CreatePipelineState

10.24.3.43 virtual void LLGL::CommandBuffer::SetResource (Resource & *resource*, std::uint32_t *slot*, long *bindFlags*, long *stageFlags* = StageFlags::AllStages) [pure virtual]

Sets the specified resource to a binding slot.

Parameters

in	*resource*	Specifies the resource to set.
in	*slot*	Specifies the slot index where to put the resource.
in	*bindFlags*	Specifies to types of binding points for this resource. This can be a bitwise OR combination of the BindFlags entries. Input and output binding flags cannot be used together when a resource is bound, e.g. a texture cannot be sampled while it is written to.
in	*stageFlags*	Specifies at which shader stages the resource is to be set. This can be a bitwise OR combinations of the StageFlags entries. By default StageFlags::AllStages.

Remarks

This function is only supported with the older graphics APIs and only available for convenience. For best performance and complete renderer independence use `SetResourceHeap`.
The following example binds a constant buffer and texture resource to the fragment shader stage:

```
myCmdBuffer->SetResource(*myConstantBuffer, 1, LLGL::BindFlags::ConstantBuffer
      , LLGL::StageFlags::FragmentStage);
myCmdBuffer->SetResource(*myTexture,        2, LLGL::BindFlags::Sampled,
      LLGL::StageFlags::FragmentStage);
```

If direct resource binding is not supported by the render system, this function has no effect.

Note

Only supported with: OpenGL, Direct3D 11, Metal.

See also

RenderingFeatures::hasDirectResourceBinding
SetResourceHeap

10.24.3.44 virtual void LLGL::CommandBuffer::SetResourceHeap (ResourceHeap & *resourceHeap*, std::uint32_t *descriptorSet* = 0, const PipelineBindPoint *bindPoint* = PipelineBindPoint::Undefined) [pure virtual]

Binds the specified resource heap to the respective pipeline.

Parameters

in	*resourceHeap*	Specifies the resource heap that contains all shader resources that will be bound to the shader pipeline.
in	*descriptorSet*	Specifies the zero-based index of the first set of resource descriptors. This **must** be in the half-open range [0, ResourceHeap::GetNumDescriptorSets). By default 0.
in	*bindPoint*	Specifies to which pipeline the resource heap is meant be bound. By default PipelineBindPoint::Undefined. If this is PipelineBindPoint::Undefined, the resource heap is automatically bound to the graphics and/or compute pipeline. Use this parameter if a resource heap has one or more resources that are in both the graphics and compute pipeline to avoid unnecessary bindings.

Remarks

Any previous resource bindings are invalid after this call.

See also

ResourceHeap::GetNumDescriptorSets

10.24.3.45 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.
This must only be used if the currently bound graphics pipeline state was created with `scissors` being empty. Otherwise, the behavior is undefined.

See also

SetScissors
GraphicsPipelineDescriptor::scissors

10.24.3.46 virtual void LLGL::CommandBuffer::SetScissors (std::uint32_t *numScissors*, const Scissor * *scissors*) [pure virtual]

Sets an array of scissor rectangles, but only if the scissor test was enabled in the previously set graphics pipeline (otherwise, this function has no effect).

Parameters

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

Remarks

This function behaves differently on the OpenGL render system, depending on the state configured with the `SetGraphicsAPIDependentState` function. If `stateOpenGL.screenSpaceOriginLower↩Left` is `false`, the origin of each scissor rectangle is on the upper-left (like for all other render systems). If `stateOpenGL.screenSpaceOriginLowerLeft` is `true`, the origin of each scissor rectangle is on the lower-left.
This must only be used if the currently bound graphics pipeline state was created with `scissors` being empty. Otherwise, the behavior is undefined.

See also

GraphicsPipelineDescriptor::scissors
SetGraphicsAPIDependentState
RasterizerDescriptor::scissorTestEnabled

10.24.3.47 virtual void LLGL::CommandBuffer::SetStencilReference (std::uint32_t *reference*, const StencilFace *stencilFace* = StencilFace::FrontAndBack) [pure virtual]

Sets the dynamic pipeline state for stencil reference values.

Parameters

in	*reference*	Specifies the reference value.
in	*stencilFace*	Specifies the faces that will be affected by this reference value. For Direct3D renderers, this must be StencilFace::FrontAndBack, which is the default value.

Remarks

This must only be used if the currently bound graphics pipeline state was created with `referenceDynamic` set to true. Otherwise, the behavior is undefined.

See also

StencilDescriptor::referenceDynamic

10.24.3.48 virtual void LLGL::CommandBuffer::SetUniform (UniformLocation *location*, const void * *data*, std::uint32_t *dataSize*) [pure virtual]

Sets the value of a single uniform (a.k.a. shader constant) in the shader program that is currently bound.

Parameters

in	*location*	Specifies the location of the uniform.
in	*data*	Raw pointer to the data that is to be copied to the uniform.
in	*dataSize*	Specifies the size (in bytes) of the input buffer `data`. This must be a multiple of 4.

Remarks

This function must only be called after a graphics or compute pipeline has been set.

Note

Only supported with: OpenGL, Vulkan, Direct3D 12.

See also

Shader::FindUniformLocation
Shader::Reflect
SetGraphicsPipeline
SetComputePipeline

10.24.3.49 virtual void LLGL::CommandBuffer::SetUniforms (UniformLocation *location*, std::uint32_t *count*, const void * *data*, std::uint32_t *dataSize*) [pure virtual]

Sets the value of multiple uniforms (a.k.a. shader constants) in the shader program that is currently bound.

Parameters

in	*location*	Specifies the location of the first uniform.
in	*count*	Specifies the number of uniforms that are meant to be updated.
in	*data*	Raw pointer to the data that is to be copied to the uniforms.
in	*dataSize*	Specifies the size (in bytes) of the input buffer `data`. This must be a multiple of 4.

Remarks

This function must only be called after a graphics or compute pipeline has been set. The order of uniforms that come after the first one can be determined by the ShaderReflection::uniform container returned by Shader↩ ::Reflect.

Note

Only supported with: OpenGL, Vulkan, Direct3D 12.

See also

Shader::FindUniformLocation
Shader::Reflect
SetGraphicsPipeline
SetComputePipeline

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

Sets the specified vertex buffer for subsequent drawing operations.

Parameters

in	*buffer*	Specifies the vertex buffer to set. This buffer must have been created with the binding flag BindFlags::VertexBuffer and its content **must not** be uninitialized.

See also

RenderSystem::CreateBuffer
RenderSystem::WriteBuffer
SetVertexBufferArray

10.24.3.51 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

10.24.3.52 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.
This must only be used if the currently bound graphics pipeline state was created with `viewports` being empty. Otherwise, the behavior is undefined.

See also

SetViewports
GraphicsPipelineDescriptor::viewports

10.24.3.53 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 must only be used if the currently bound graphics pipeline state was created with `viewports` being empty. Otherwise, the behavior is undefined.

See also

GraphicsPipelineDescriptor::viewports
SetGraphicsAPIDependentState
RenderingLimits::maxViewports

10.24.3.54 virtual void LLGL::CommandBuffer::UpdateBuffer (Buffer & *dstBuffer*, std::uint64_t *dstOffset*, const void * *data*, std::uint16_t *dataSize*) [pure virtual]

Updates the data of the specified buffer during encoding the command buffer.

Parameters

in	*dstBuffer*	Specifies the destination buffer whose data is to be updated.
in	*dstOffset*	Specifies the destination offset (in bytes) at which the buffer is to be updated. This offset plus the data block size (i.e. `dstOffset + dataSize`) must be less than or equal to the size of the buffer.
in	*data*	Raw pointer to the data with which the buffer is to be updated. This **must not** be null!
in	*dataSize*	Specifies the size (in bytes) of the data block which is to be updated. This is limited to 2^16 = 65536 bytes, because it may be written to the command buffer itself before it is copied to the destination buffer (depending on the backend).

Remarks

To update buffers larger than 65536 bytes, use RenderSystem::WriteBuffer or RenderSystem::MapBuffer. For performance reasons, it is recommended to encode this command outside of a render pass. Otherwise, render pass interruptions might be inserted by LLGL.

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

- CommandBuffer.h

10.25 LLGL::CommandBufferDescriptor Struct Reference

Command buffer descriptor structure.

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

Public Member Functions

- CommandBufferDescriptor ()=default
- CommandBufferDescriptor (const CommandBufferDescriptor &)=default
- CommandBufferDescriptor & operator= (const CommandBufferDescriptor &)=default
- CommandBufferDescriptor (long flags)

  Constructs the command buffer descriptor with the specified flags.
- CommandBufferDescriptor (long flags, std::uint32_t numNativeBuffers)

  Constructs the command buffer descriptor with the specified flags and number of native buffers.

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.

10.25.1 Detailed Description

Command buffer descriptor structure.

See also

RenderSystem::CreateCommandBuffer

10.25.2 Constructor & Destructor Documentation

10.25.2.1 LLGL::CommandBufferDescriptor::CommandBufferDescriptor ( ) [default]

10.25.2.2 LLGL::CommandBufferDescriptor::CommandBufferDescriptor (const CommandBufferDescriptor &) [default]

10.25.2.3 LLGL::CommandBufferDescriptor::CommandBufferDescriptor (long *flags*) [inline]

Constructs the command buffer descriptor with the specified flags.

10.25.2.4 LLGL::CommandBufferDescriptor::CommandBufferDescriptor (long *flags*, std::uint32_t *numNativeBuffers*) [inline]

Constructs the command buffer descriptor with the specified flags and number of native buffers.

10.25.3 Member Function Documentation

10.25.3.1 CommandBufferDescriptor& LLGL::CommandBufferDescriptor::operator= (const CommandBufferDescriptor &) [default]

10.25.4 Member Data Documentation

10.25.4.1 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

10.25.4.2 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

10.26 LLGL::CommandBufferFlags Struct Reference

Command buffer creation flags.

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

Public Types

- enum { Secondary = (1 << 0), MultiSubmit = (1 << 1), ImmediateSubmit = (1 << 2) }

10.26.1 Detailed Description

Command buffer creation flags.

Remarks

A default command buffer is a primary command buffer (No `Secondary` flag) that has to be submitted explicitly (No `ImmediateSubmit` flag) and can only be submitted once until it is encoded again (No `MultiSubmit` flag).

See also

CommandBufferDescriptor::flags

10.26.2 Member Enumeration Documentation

10.26.2.1 anonymous enum

Enumerator

Secondary Specifies that the encoded command buffer will be submitted as a secondary command buffer.

Remarks

If this is specified, the command buffer must be submitted using the `Execute` function of a primary command buffer. This cannot be used in combination with the `ImmediateSubmit` flag.

See also

CommandBuffer::Execute

MultiSubmit Specifies that the encoded command buffer can be submitted multiple times.

Remarks

If this is not specified, the command buffer must be encoded again after it has been submitted to the command queue. This cannot be used in combination with the `ImmediateSubmit` flag.

See also

CommandQueue::Submit(CommandBuffer&)

ImmediateSubmit Specifies that the encoded command buffer is an immediate command buffer.

Remarks

If this specified, the command buffer is submitted immediately after encoding is done and calling CommandQueue::Submit on such a command buffer has no effect.
This cannot be used in combination with the `Secondary` or `MultiSubmit` flags.

See also

CommandBuffer::End

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

- CommandBufferFlags.h

10.27 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

10.27.1 Detailed Description

Command queue interface.

Remarks

This class is used to submit one or more command buffers (aka. command lists) into the command queue as well as CPU/GPU synchronization objects (aka. fences). For immediate command buffers, the CommandQueue::Submit function has no effect.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 LLGL::CommandQueue::CommandQueue ( ) [protected],[default]

10.27.3 Member Function Documentation

10.27.3.1 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 undefined.
in	*dataSize*	Specifies the size (in bytes) of the output data. This must be equal to `numQueries * sizeof(T)` where `T` is the type of the query entries described above.

Returns

True, if all results are available. Otherwise, the results are (partially) unavailable and the content of the output data is undefined.

Remarks

Here is a usage example:

```
// Get results of 10 occlusion queries
std::uint64_t occlusionQueryResults[10] = {};
myCmdQueue->QueryResult(*myOcclusionQuery, 0, 10, occlusionQueryResults, sizeof(occlusionQueryResults));

// Get result of a pipeline statistics query
LLGL::QueryPipelineStatistics stats;
myCmdQueue->QueryResult(*myPipelineStatsQuery, 0, 1, &stats, sizeof(stats));
```

10.27.3.2 virtual void LLGL::CommandQueue::Submit (CommandBuffer & *commandBuffer*) [pure virtual]

Submits the specified command buffer to the command queue.

Parameters

in	*commandBuffer*	Specifies the command buffer that is to be submitted. If the command buffer was created with the CommandBufferFlags::ImmediateSubmit flag, this function has no effect.

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 ∗)

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

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

10.27.3.4 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

10.27.3.5 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

10.28 LLGL::ComputePipelineDescriptor Struct Reference

Compute pipeline state descriptor structure.

```
#include <PipelineStateFlags.h>
```

Public Attributes

- const PipelineLayout * pipelineLayout = nullptr

  Pointer to an optional pipeline layout for the graphics pipeline.
- Shader * computeShader = nullptr

  Specifies the compute shader.

10.28.1 Detailed Description

Compute pipeline state descriptor structure.

See also

RenderSystem::CreatePipelineState

10.28.2 Member Data Documentation

10.28.2.1 Shader* LLGL::ComputePipelineDescriptor::computeShader = nullptr

Specifies the compute shader.

Remarks

This must never be null when a compute PSO is created.

10.28.2.2 const PipelineLayout* LLGL::ComputePipelineDescriptor::pipelineLayout = nullptr

Pointer to an optional pipeline layout for the graphics pipeline.

Remarks

This layout determines at which slots buffer resources can be bound. This is ignored by render systems which do not support pipeline layouts.

Note

Only supported with: Vulkan, Direct3D 12

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

- PipelineStateFlags.h

10.29 LLGL::ComputeShaderAttributes Struct Reference

Compute shader specific descriptor structure.

```
#include <ShaderFlags.h>
```

Public Attributes

- Extent3D workGroupSize = { 1, 1, 1 }

  Specifies the number of threads per threadgroup in X, Y, and Z direction. By default (1, 1, 1).

10.29.1 Detailed Description

Compute shader specific descriptor structure.

See also

ShaderDescriptor::compute
ShaderReflection::compute

10.29.2 Member Data Documentation

10.29.2.1 Extent3D LLGL::ComputeShaderAttributes::workGroupSize = { 1, 1, 1 }

Specifies the number of threads per threadgroup in X, Y, and Z direction. By default (1, 1, 1).

Remarks

Each component must be greater than zero.
Only the Metal backend supports dispatch compute kernels with dynamic work group sizes. If not used for shader reflection, all other renderers need to specified the workgroup size within the shader code:

- For GLSL: `layout(local_size_x = X, local_size_y = Y, local_size_z = Z)`
- For HLSL: `[numthreads(X, Y, Z)]`

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

- ShaderFlags.h

10.30 LLGL::CPUAccessFlags Struct Reference

CPU read/write access flag enumeration for mapping GPU into CPU memory space.

```
#include <ResourceFlags.h>
```

Public Types

- enum { Read = (1 << 0), Write = (1 << 1), ReadWrite = (Read | Write) }

10.30.1 Detailed Description

CPU read/write access flag enumeration for mapping GPU into CPU memory space.

See also

BufferDescriptor::cpuAccessFlags
RenderSystem::MapBuffer
RenderSystem::UnmapBuffer

10.30.2 Member Enumeration Documentation

10.30.2.1 anonymous enum

Enumerator

Read Resource mapping with CPU read access is required.

See also

CPUAccess::ReadOnly
CPUAccess::ReadWrite

Write Resource mapping with CPU write access is required.

See also

CPUAccess::WriteOnly
CPUAccess::WriteDiscard
CPUAccess::ReadWrite

ReadWrite

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

- ResourceFlags.h

10.31 LLGL::DepthBiasDescriptor Struct Reference

Depth bias descriptor structure to control fragment depth values.

```
#include <PipelineStateFlags.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.

10.31.1 Detailed Description

Depth bias descriptor structure to control fragment depth values.

10.31.2 Member Data Documentation

10.31.2.1 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).

10.31.2.2 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.

10.31.2.3 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:

- PipelineStateFlags.h

10.32 LLGL::DepthDescriptor Struct Reference

Depth state descriptor structure.

```
#include <PipelineStateFlags.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.

10.32.1 Detailed Description

Depth state descriptor structure.

See also

GraphicsPipelineDescriptor::depth

10.32.2 Member Data Documentation

10.32.2.1 CompareOp LLGL::DepthDescriptor::compareOp = CompareOp::Less

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

10.32.2.2 bool LLGL::DepthDescriptor::testEnabled = false

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

10.32.2.3 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:

- PipelineStateFlags.h

10.33 LLGL::DispatchIndirectArguments Struct Reference

Format structure for the arguments of an indirect compute command.

```
#include <IndirectArguments.h>
```

Public Attributes

- std::uint32_t numThreadGroups [3]

  Number of thread groups in X, Y, and Z dimension.

10.33.1 Detailed Description

Format structure for the arguments of an indirect compute command.

Remarks

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

Note

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

See also

CommandBuffer::DispatchIndirect
OpenGL counterpart `DispatchIndirectCommand`: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glDispatchComputeIndirect.xhtml
Vulkan counterpart `VkDispatchIndirectCommand`: https://www.khronos.org/registry/vulkan/specs1-extensions/man/html/VkDispatchIndirectCommand.html
Direct3D11 counterpart: N/A
Direct3D12 counterpart `D3D12_DISPATCH_ARGUMENTS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d12/ns-d3d12-d3d12_dispatch_arguments
Metal counterpart `MTLDispatchThreadgroupsIndirectArguments`: https://developer.apple.com/documentation/metal/mtldispatchthreadgroupsindirectarguments?language=objc

10.33.2 Member Data Documentation

10.33.2.1 std::uint32_t LLGL::DispatchIndirectArguments::numThreadGroups[3]

Number of thread groups in X, Y, and Z dimension.

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

- IndirectArguments.h

10.34 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 UTF8String GetDeviceName () const =0

  Returns the device name of this display in UTF-8 encoding. This may also be empty, if the platform does not support display names.
- virtual Offset2D GetOffset () const =0

  Returns the 2D offset relative to the primary display.
- virtual bool ResetDisplayMode ()=0

  Resets the display mode to its default value depending on the host system configuration.
- virtual bool SetDisplayMode (const DisplayModeDescriptor &displayModeDesc)=0

  Sets the new display mode for this display.
- virtual DisplayModeDescriptor GetDisplayMode () const =0

  Returns the current display mode of this display.
- virtual std::vector< DisplayModeDescriptor > GetSupportedDisplayModes () const =0

  Returns a list of all modes that are supported by this display.

Static Public Member Functions

- static std::size_t Count ()

  Returns the number of available displays.
- static Display *const * GetList ()

  Returns a null-terminated array of all displays.
- static Display * Get (std::size_t index)

  Returns the specified display or null if the index is out of bounds.
- static Display * GetPrimary ()

  Returns the primary display or null if no display can be found.
- 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 bool SetCursorPosition (const Offset2D &position)

  Sets the cursor to the specified screen coordinate and returns true on success. Otherwise, cursor relocation is not supported.
- static Offset2D GetCursorPosition ()

  Returns the screen coordiante of the cursor.

Static Protected Member Functions

- static void FinalizeDisplayModes (std::vector< DisplayModeDescriptor > &displayModeDescs)

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

Additional Inherited Members

10.34.1 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:

```
for (Display* const * myDisplayList = LLGL::Display::GetList(); Display* myDisplay =
      *myDisplayList; ++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;
}
```

10.34.2 Member Function Documentation

10.34.2.1 static std::size_t LLGL::Display::Count ( ) [static]

Returns the number of available displays.

Remarks

This function always checks for updates in the display list.

See also

Get

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

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

See also

GetSupportedDisplayModes

10.34.2.3 static Display* LLGL::Display::Get (std::size_t *index*) [static]

Returns the specified display or null if the index is out of bounds.

Parameters

in	*index*	Specifies the zero-based index for the display to be returned. This should be in the half-open range [0, Count) or null will be returned.

Remarks

This function always checks for updates in the display list.

See also

Count
GetPrimary

10.34.2.4 static Offset2D LLGL::Display::GetCursorPosition ( ) [static]

Returns the screen coordiante of the cursor.

10.34.2.5 virtual UTF8String LLGL::Display::GetDeviceName ( ) const [pure virtual]

Returns the device name of this display in UTF-8 encoding. This may also be empty, if the platform does not support display names.

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

Returns the current display mode of this display.

See also

SetDisplayMode

10.34.2.7 static Display* const* LLGL::Display::GetList ( ) [static]

Returns a null-terminated array of all displays.

Returns

Pointer to an array of Display pointers with Count()+1 elements and the last element being null.

Remarks

This function always checks for updates in the display list.

See also

Count

10.34.2.8 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

10.34.2.9 static Display* LLGL::Display::GetPrimary ( ) [static]

Returns the primary display or null if no display can be found.

See also

Get

10.34.2.10 virtual std::vector<DisplayModeDescriptor> LLGL::Display::GetSupportedDisplayModes ( ) const [pure virtual]

Returns a list of all modes that are supported by this display.

Remarks

This list is always sorted in the following manner: The first sorting criterion is the number of pixels (resolution width times resolution height) in ascending order, and the second sorting criterion is the refresh rate in ascending order. To get only the currently active display mode, use GetDisplayMode.

See also

GetDisplayMode

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

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

See also

ShowCursor

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

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

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

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

See also

SetDisplayMode

10.34.2.14 static bool LLGL::Display::SetCursorPosition (const Offset2D & *position*) [static]

Sets the cursor to the specified screen coordinate and returns true on success. Otherwise, cursor relocation is not supported.

10.34.2.15 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

10.34.2.16 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

10.35 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).

10.35.1 Detailed Description

Display mode descriptor structure.

Remarks

Describes the attributes of a physical display.

See also

VideoOutputDescriptor::displayModes
SwapChainDescriptor

10.35.2 Member Data Documentation

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

Display refresh rate (in Hz).

10.35.2.2 Extent2D LLGL::DisplayModeDescriptor::resolution

Display resolution (in pixels).

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

- DisplayFlags.h

10.36 LLGL::DrawIndexedIndirectArguments Struct Reference

Format structure for the arguments of an indirect indexed draw command.

```
#include <IndirectArguments.h>
```

Public Attributes

- std::uint32_t numIndices

  Specifies the number of indices per instance.
- std::uint32_t numInstances

  Specifies the number of instances to draw.
- std::uint32_t firstIndex

  Specifies the zero-based offset of the first index from the index buffer.
- std::int32_t vertexOffset

  Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.
- std::uint32_t firstInstance

  Specifies the zero-based offset of the first instance.

10.36.1 Detailed Description

Format structure for the arguments of an indirect indexed draw command.

Remarks

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

Note

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

See also

CommandBuffer::DrawIndexedIndirect
OpenGL counterpart `DrawElementsIndirectCommand`: https://www.khronos.org/registry/↩OpenGL/extensions/ARB/ARB_draw_indirect.txt
Vulkan counterpart `VkDrawIndexedIndirectCommand`: https://www.khronos.org/registry/vulkan/sp1-extensions/man/html/VkDrawIndexedIndirectCommand.html
Direct3D11 counterpart `D3D11_DRAW_INDEXED_INSTANCED_INDIRECT_ARGS`: https↩://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_↩draw_indexed_instanced_indirect_args
Direct3D12 counterpart `D3D12_DRAW_INDEXED_ARGUMENTS`: https://docs.microsoft.↩com/en-us/windows/desktop/api/D3D12/ns-d3d12-d3d12_draw_indexed_arguments
Metal counterpart `MTLDrawIndexedPrimitivesIndirectArguments`: https://developer.↩apple.com/documentation/metal/mtldrawindexedprimitivesindirectarguments?language=obj

10.36.2 Member Data Documentation

10.36.2.1 std::uint32_t LLGL::DrawIndexedIndirectArguments::firstIndex

Specifies the zero-based offset of the first index from the index buffer.

10.36.2.2 std::uint32_t LLGL::DrawIndexedIndirectArguments::firstInstance

Specifies the zero-based offset of the first instance.

Note

This parameter modifies the instance ID within the shader pipeline differently for `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan), due to rendering API differences. The system value `SV_InstanceID` in HLSL will always start with zero, but the system value `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan) will start with the value of `firstInstance`.

10.36.2.3 std::uint32_t LLGL::DrawIndexedIndirectArguments::numIndices

Specifies the number of indices per instance.

10.36.2.4 std::uint32_t LLGL::DrawIndexedIndirectArguments::numInstances

Specifies the number of instances to draw.

10.36.2.5 std::int32_t LLGL::DrawIndexedIndirectArguments::vertexOffset

Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.

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

- IndirectArguments.h

10.37 LLGL::DrawIndirectArguments Struct Reference

Format structure for the arguments of an indirect draw command.

```
#include <IndirectArguments.h>
```

Public Attributes

- std::uint32_t numVertices

  Specifies the number of vertices per instance.
- std::uint32_t numInstances

  Specifies the number of instances to draw.
- std::uint32_t firstVertex

  Specifies the zero-based offset of the first vertex from the vertex buffer.
- std::uint32_t firstInstance

  Specifies the zero-based offset of the first instance.

10.37.1 Detailed Description

Format structure for the arguments of an indirect draw command.

Remarks

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

Note

This is a plain-old-data (POD) structure, so it has no default constructor to make it easily compatible with the GPU memory space.

See also

CommandBuffer::DrawIndirect
OpenGL counterpart `DrawArraysIndirectCommand`: https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_draw_indirect.txt
Vulkan counterpart `VkDrawIndirectCommand`: https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/VkDrawIndirectCommand.html
Direct3D11 counterpart `D3D11_DRAW_INSTANCED_INDIRECT_ARGS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_draw_instanced_indirect_args
Direct3D12 counterpart `D3D12_DRAW_ARGUMENTS`: https://docs.microsoft.com/en-us/windows/desktop/D3D12/ns-d3d12-d3d12_draw_arguments
Metal counterpart `MTLDrawPrimitivesIndirectArguments`: https://developer.apple.com/documentation/metal/mtldrawprimitivesindirectarguments?language=objc

10.37.2 Member Data Documentation

10.37.2.1 std::uint32_t LLGL::DrawIndirectArguments::firstInstance

Specifies the zero-based offset of the first instance.

Note

This parameter modifies the instance ID within the shader pipeline differently for `SV_InstanceID` in HLSL and `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan), due to rendering API differences. The system value `SV_InstanceID` in HLSL will always start with zero, but the system value `gl_InstanceID` in GLSL (or `gl_InstanceIndex` for Vulkan) will start with the value of `firstInstance`.

10.37.2.2 std::uint32_t LLGL::DrawIndirectArguments::firstVertex

Specifies the zero-based offset of the first vertex from the vertex buffer.

Note

This parameter modifies the vertex ID within the shader pipeline differently for `SV_VertexID` in HLSL and `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan), due to rendering API differences. The system value `SV_VertexID` in HLSL will always start with zero, but the system value `gl_VertexID` in GLSL (or `gl_VertexIndex` for Vulkan) will start with the value of `firstVertex`.

10.37.2.3 std::uint32_t LLGL::DrawIndirectArguments::numInstances

Specifies the number of instances to draw.

10.37.2.4 std::uint32_t LLGL::DrawIndirectArguments::numVertices

Specifies the number of vertices per instance.

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

- IndirectArguments.h

10.38 LLGL::DrawPatchIndirectArguments Struct Reference

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

```
#include <IndirectArguments.h>
```

Public Attributes

- std::uint32_t numPatches

  Specifies the number of patches per instance.
- std::uint32_t numInstances

  Specifies the number of instances to draw.
- std::uint32_t firstPatch

  Specifies the patch start index.
- std::uint32_t firstInstance

  Specifies the first instance to draw.

10.38.1 Detailed Description

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

Remarks

The Metal API treats the arguments for rendering tessellated patches differently, so this structure is required to fill the buffer that is used for the arguments of an indirect draw command on the Metal backend.

Note

Only supported with: Metal.

See also

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect
Metal counterpart `MTLDrawPatchIndirectArguments:` https://developer.apple.com/documentation/metal/mtldrawpatchindirectarguments?language=objc

10.38.2 Member Data Documentation

10.38.2.1 std::uint32_t LLGL::DrawPatchIndirectArguments::firstInstance

Specifies the first instance to draw.

10.38.2.2 std::uint32_t LLGL::DrawPatchIndirectArguments::firstPatch

Specifies the patch start index.

10.38.2.3 std::uint32_t LLGL::DrawPatchIndirectArguments::numInstances

Specifies the number of instances to draw.

10.38.2.4 std::uint32_t LLGL::DrawPatchIndirectArguments::numPatches

Specifies the number of patches per instance.

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

- IndirectArguments.h

10.39 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.

10.39.1 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

10.39.2 Constructor & Destructor Documentation

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

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

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

Constructor to initialize all attributes.

10.39.3 Member Data Documentation

10.39.3.1 void* LLGL::DstImageDescriptor::data = nullptr

Pointer to the read/write image data.

10.39.3.2 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.

10.39.3.3 DataType LLGL::DstImageDescriptor::dataType = DataType::UInt8

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

10.39.3.4 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

10.40 LLGL::Window::EventListener Class Reference

Interface for all window event listeners.

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

Inheritance diagram for LLGL::Window::EventListener:

Protected Member Functions

- virtual void OnProcessEvents (Window &sender)

  Send when the window events are about to be polled. The event listeners receive this event before the window itself.
- virtual void OnQuit (Window &sender, bool &veto)

  Send when the window is about to quit.
- virtual void OnKeyDown (Window &sender, Key keyCode)

  Send when a key (from keyboard or mouse) has been pushed.
- virtual void OnKeyUp (Window &sender, Key keyCode)

  Send when a key (from keyboard or mouse) has been released.
- virtual void OnDoubleClick (Window &sender, Key keyCode)

  Send when a mouse button has been double clicked.
- virtual void OnChar (Window &sender, wchar_t chr)

Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.

- virtual void OnWheelMotion (Window &sender, int motion)

  Send when the mouse wheel has been moved on the sender window.
- virtual void OnLocalMotion (Window &sender, const Offset2D &position)

  Send when the mouse has been moved on the sender window.
- virtual void OnGlobalMotion (Window &sender, const Offset2D &motion)

  Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.
- virtual void OnResize (Window &sender, const Extent2D &clientAreaSize)

  Send when the window has been resized.
- virtual void OnGetFocus (Window &sender)

  Send when the window gets the keyboard focus.
- virtual void OnLostFocus (Window &sender)

  Send when the window lost the keyboard focus.
- virtual void OnTimer (Window &sender, std::uint32_t timerID)

  Send when the window received a timer event with the specified timer ID number.

Friends

- class Window

Additional Inherited Members

10.40.1 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

10.40.2 Member Function Documentation

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

Send when a character specific key has been typed on the sender window. This will repeat depending on the OS keyboard settings.

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

Send when a mouse button has been double clicked.

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

Send when the window gets the keyboard focus.

10.40.2.4 virtual void LLGL::Window::EventListener::OnGlobalMotion (Window & *sender,* const Offset2D & *motion*) [protected], [virtual]

Send when the global mouse position has changed. This is a raw input and independent of the screen resolution.

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

Send when a key (from keyboard or mouse) has been pushed.

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

Send when a key (from keyboard or mouse) has been released.

10.40.2.7 virtual void LLGL::Window::EventListener::OnLocalMotion (Window & *sender,* const Offset2D & *position*) [protected], [virtual]

Send when the mouse has been moved on the sender window.

10.40.2.8 virtual void LLGL::Window::EventListener::OnLostFocus (Window & *sender*) [protected], [virtual]

Send when the window lost the keyboard focus.

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

Send when the window events are about to be polled. The event listeners receive this event before the window itself.

See also

Window::OnProcessEvents

10.40.2.10 virtual void LLGL::Window::EventListener::OnQuit (Window & *sender,* bool & *veto*) [protected], [virtual]

Send when the window is about to quit.

Parameters

in	*sender*	Specifies the sender of this event.
out	*veto*	Specifies whether to cancel the quit event. If set to true, the call to `PostQuit` does not change the state `sender`, only the event listeners get informed. If no event listener sets this parameter to true, `sender` is set to the 'Quit' state and `ProcessEvents` returns false from then on.

See also

Window::ProcessEvents

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

Send when the window has been resized.

10.40.2.12 virtual void LLGL::Window::EventListener::OnTimer (Window & *sender,* std::uint32_t *timerID*) [protected], [virtual]

Send when the window received a timer event with the specified timer ID number.

Note

Only supported on: MS. Windows.

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

Send when the mouse wheel has been moved on the sender window.

10.40.3 Friends And Related Function Documentation

10.40.3.1 friend class Window [friend]

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

- Window.h

10.41 LLGL::Canvas::EventListener Class Reference

Interface for all canvas event listeners.

```
#include <Canvas.h>
```

Inheritance diagram for LLGL::Canvas::EventListener:

Protected Member Functions

- virtual void OnProcessEvents (Canvas &sender)

  Send when the canvas events are about to be polled. The event listeners receive this event before the canvas itself.
- virtual void OnQuit (Canvas &sender, bool &veto)

  Send when the canvas is about to quit.

Friends

- class Canvas

Additional Inherited Members

10.41.1 Detailed Description

Interface for all canvas event listeners.

10.41.2 Member Function Documentation

10.41.2.1 virtual void LLGL::Canvas::EventListener::OnProcessEvents (Canvas & *sender*) [protected], [virtual]

Send when the canvas events are about to be polled. The event listeners receive this event before the canvas itself.

Parameters

in	*sender*	Specifies the sender of this event.

See also

Canvas::OnProcessEvents

10.41.2.2 virtual void LLGL::Canvas::EventListener::OnQuit (Canvas & *sender,* bool & *veto*) [protected], [virtual]

Send when the canvas is about to quit.

Parameters

in	*sender*	Specifies the sender of this event.
out	*veto*	Specifies whether to cancel the quit event. If set to true, the call to `PostQuit` does not change the state `sender`, only the event listeners get informed. If no event listener sets this parameter to true, `sender` is set to the 'Quit' state and `ProcessEvents` returns false from then on.

See also

Canvas::ProcessEvents

10.41.3 Friends And Related Function Documentation

10.41.3.1 friend class Canvas [friend]

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

- Canvas.h

10.42 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.

10.42.1 Detailed Description

2-Dimensional extent structure.

Remarks

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

10.42.2 Constructor & Destructor Documentation

10.42.2.1 LLGL::Extent2D::Extent2D ( ) [default]

10.42.2.2 LLGL::Extent2D::Extent2D (const Extent2D &) [default]

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

10.42.3 Member Data Documentation

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

Extent Y axis, i.e. height.

10.42.3.2 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

10.43 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.

10.43.1 Detailed Description

3-Dimensional extent structure.

Remarks

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

10.43.2 Constructor & Destructor Documentation

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

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

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

10.43.3 Member Data Documentation

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

Extent Z axis, i.e. depth.

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

Extent Y axis, i.e. height.

10.43.3.3 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

10.44 LLGL::Fence Class Reference

Fence interface for CPU/GPU synchronization.

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

Inheritance diagram for LLGL::Fence:

Additional Inherited Members

10.44.1 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

10.45 LLGL::FormatAttributes Struct Reference

Vertex and texture format attributes structure.

```
#include <Format.h>
```

Public Attributes

- std::uint16_t bitSize

  Size (in bits) of the a pixel block or vertex attribute. This is 0 for invalid formats.
- std::uint8_t blockWidth

  Width of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.
- std::uint8_t blockHeight

  Height of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.
- std::uint8_t components

  Number of components. This is either 1, 2, 3, or 4. This is 0 for invalid formats.
- ImageFormat format

  Image pixel format. This is ignored for depth-stencil and compressed formats.
- DataType dataType

  Color component data type. This is ignored for depth-stencil and compressed formats.
- long flags

  Specifies the boolean attributes of this format descriptor. This can be bitwise OR combination of the FormatFlags entries.

10.45.1 Detailed Description

Vertex and texture format attributes structure.

Remarks

Describes all attributes of a hardware format, e.g. bit size, color components, compression etc.

Note

This descriptor structure has no default initializers to fulfill the requirements of a plain-old-data (POD) structure.

See also

GetFormatAttribs

10.45.2 Member Data Documentation

10.45.2.1 std::uint16_t LLGL::FormatAttributes::bitSize

Size (in bits) of the a pixel block or vertex attribute. This is 0 for invalid formats.

10.45.2.2 std::uint8_t LLGL::FormatAttributes::blockHeight

Height of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.

10.45.2.3 std::uint8_t LLGL::FormatAttributes::blockWidth

Width of a pixel block. This is 1 for all non-compressed formats. This is 0 for invalid formats.

10.45.2.4 std::uint8_t LLGL::FormatAttributes::components

Number of components. This is either 1, 2, 3, or 4. This is 0 for invalid formats.

10.45.2.5 DataType LLGL::FormatAttributes::dataType

Color component data type. This is ignored for depth-stencil and compressed formats.

Remarks

For depth-stencil formats, this data type always refers to the depth component, because stencil components are always in 8-bit unsigned integer format.

10.45.2.6 long LLGL::FormatAttributes::flags

Specifies the boolean attributes of this format descriptor. This can be bitwise OR combination of the FormatFlags entries.

See also

FormatFlags

10.45.2.7 ImageFormat LLGL::FormatAttributes::format

Image pixel format. This is ignored for depth-stencil and compressed formats.

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

- Format.h

10.46 LLGL::FormatFlags Struct Reference

Format attribute flags enumeration.

```
#include <Format.h>
```

Public Types

- enum {
  HasDepth = (1 << 0), HasStencil = (1 << 1), IsColorSpace_sRGB = (1 << 2), IsCompressed = (1 << 3), IsNormalized = (1 << 4), IsInteger = (1 << 5), IsUnsigned = (1 << 6), IsPacked = (1 << 7), SupportsRenderTarget = (1 << 8), SupportsMips = (1 << 9), SupportsGenerateMips = (1 << 10), SupportsTexture1D = (1 << 11), SupportsTexture2D = (1 << 12), SupportsTexture3D = (1 << 13), SupportsTextureCube = (1 << 14), SupportsVertex = (1 << 15), IsUnsignedInteger = (IsUnsigned | IsInteger), HasDepthStencil = (HasDepth | HasStencil) }

10.46.1 Detailed Description

Format attribute flags enumeration.

See also

FormatAttributes::flags

10.46.2 Member Enumeration Documentation

10.46.2.1 anonymous enum

Enumerator

HasDepth Specifies whether the format has a depth component (e.g. Format::D16UNorm or Format::D24↩UNormS8UInt).

HasStencil Specifies whether the format has a stencil component (e.g. Format::D24UNormS8UInt).

IsColorSpace_sRGB Specifies whether the format is in non-linear sRGB color space (e.g. Format::RGB↩A8UNorm_sRGB or Format::BC1UNorm_sRGB).

IsCompressed Specifies whether the format is compressed (e.g. Format::BC1UNorm or Format::BC5S↩Norm).

IsNormalized Specifies whether the format has normalized integer components (e.g. Format::R8UNorm or Format::R8SNorm).

IsInteger Specifies whether the format is an integer format (e.g. Format::R8UNorm or Format::R8SInt).

IsUnsigned Specifies whether the format is an unsigned format (e.g. Format::R8UInt or Format::RG11↩B10Float).

IsPacked Specifies whether the format is packed (e.g. Format::RGB10A2UNorm or Format::RGB9E5Float).

SupportsRenderTarget Specifies whether the format can be used for render targets or rather depth-stencil or color attachments.

See also

BindFlags::ColorAttachment
BindFlags::DepthStencilAttachment

SupportsMips Specifies whether the format can have more than one MIP-map.

See also

TextureDescriptor::mipLevels

SupportsGenerateMips Specifies whether the format supports to automatically generate MIP-maps.

Remarks

This implies that the flags `SupportsMips` and `SupportsRenderTarget` are also enabled.

See also

MiscFlgas::GenerateMips
CommandBuffer::GenerateMips

SupportsTexture1D Specifies whether the format can be used for 1D and 1D-array textures.

Remarks

For example, block compressions (e.g. Format::BC1UNorm) cannot be used for 1D textures, because they are made out of 4x4 blocks.

See also

TextureType::Texture1D
TextureType::Texture1DArray

SupportsTexture2D Specifies whether the format can be used for 2D and 2D-array textures (both regular and multisampled).

See also

TextureType::Texture2D
TextureType::Texture2DArray
TextureType::Texture2DMS
TextureType::Texture2DMSArray

SupportsTexture3D Specifies whether the format can be used for 3D textures.

See also

TextureType::Texture3D

SupportsTextureCube Specifies whether the format can be used for cube and cube-array textures.

See also

TextureType::TextureCube
TextureType::TextureCubeArray

SupportsVertex Specifies whether the format can be used for vertex attributes.

See also

VertexAttribute::format

IsUnsignedInteger Combines the format flags `IsInteger` and `IsUnsigned`.

HasDepthStencil Combines the format flags `HasDepth` and `HasStencil`.

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

- Format.h

10.47 LLGL::FragmentAttribute Struct Reference

Fragment output attribute structure.

```
#include <FragmentAttribute.h>
```

Public Member Functions

- FragmentAttribute ()=default
- FragmentAttribute (const FragmentAttribute &)=default
- FragmentAttribute & operator= (const FragmentAttribute &)=default
- FragmentAttribute (const StringView &name, std::uint32_t location=0)

  Constructor for minimal fragment attribute information.
- FragmentAttribute (const StringView &name, const Format format, std::uint32_t location=0, const SystemValue systemValue=SystemValue::Undefined)

  Constructor to initialize all members.

Public Attributes

- std::string name

  Fragment attribute name (for GLSL) or semantic name (for HLSL).
- Format format = Format::RGBA32Float

  Fragment attribute format. By default Format::RGBA32Float.
- std::uint32_t location = 0

  Fragment attribute location.
- SystemValue systemValue = SystemValue::Undefined

  Specifies the system value type for this fragment attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.

10.47.1 Detailed Description

Fragment output attribute structure.

Remarks

This is primarily used for shader reflection and to bind fragment output attributes for OpenGL 2.0.

See also

FragmentShaderAttributes::outputAttribs
VertexAttribute

10.47.2 Constructor & Destructor Documentation

10.47.2.1 LLGL::FragmentAttribute::FragmentAttribute ( ) [default]

10.47.2.2 LLGL::FragmentAttribute::FragmentAttribute (const FragmentAttribute &) [default]

10.47.2.3 LLGL::FragmentAttribute::FragmentAttribute (const StringView & *name*, std::uint32_t *location* = 0) [inline]

Constructor for minimal fragment attribute information.

10.47.2.4 LLGL::FragmentAttribute::FragmentAttribute (const StringView & *name*, const Format *format*, std::uint32_t *location* = 0, const SystemValue *systemValue* = SystemValue::Undefined) [inline]

Constructor to initialize all members.

10.47.3 Member Function Documentation

10.47.3.1 FragmentAttribute& LLGL::FragmentAttribute::operator= (const FragmentAttribute &) [default]

10.47.4 Member Data Documentation

10.47.4.1 Format LLGL::FragmentAttribute::format = Format::RGBA32Float

Fragment attribute format. By default Format::RGBA32Float.

10.47.4.2 std::uint32_t LLGL::FragmentAttribute::location = 0

Fragment attribute location.

Remarks

Here is an example of two fragment output attributes in GLSL with location 0 and 1:

```
layout(location = 0) out vec4 myBaseColor;
layout(location = 1) out vec4 mySecondaryColor;
```

Here is an example of two fragment output attributes in HLSL with location 0 and 1:

```
struct PixelShaderOutput
{
    float4 myBaseColor      : SV_Target0;
    float4 mySecondaryColor : SV_Target1;
};
```

Here is an example of two fragment output attributes in Metal with location 0 and 1:

```
struct FragmentFunctionOutput
{
    float4 myBaseColor      [[color(0)]];
    float4 mySecondaryColor [[color(1)]];
};
```

10.47.4.3 std::string LLGL::FragmentAttribute::name

Fragment attribute name (for GLSL) or semantic name (for HLSL).

Remarks

Semantic names in HLSL may contain an index as suffix. However, this name must not contain an index suffix, because it will be added automatically.

10.47.4.4 SystemValue LLGL::FragmentAttribute::systemValue = SystemValue::Undefined

Specifies the system value type for this fragment attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.

Remarks

This can only be one the following values:

- SystemValue::Undefined
- SystemValue::Color
- SystemValue::Depth
- SystemValue::DepthGreater
- SystemValue::DepthLess
- SystemValue::Stencil

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

- FragmentAttribute.h

10.48 LLGL::FragmentShaderAttributes Struct Reference

Fragment shader specific descriptor structure.

```
#include <ShaderFlags.h>
```

Public Attributes

- std::vector< FragmentAttribute > outputAttribs

  Fragment shader output attributes.

10.48.1 Detailed Description

Fragment shader specific descriptor structure.

See also

ShaderDescriptor::fragment
ShaderReflection::fragment

10.48.2 Member Data Documentation

10.48.2.1 std::vector<FragmentAttribute> LLGL::FragmentShaderAttributes::outputAttribs

Fragment shader output attributes.

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

- ShaderFlags.h

10.49 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 constantBufferBindings
      Counter for all individual constant buffer bindings.
    std::uint32_t sampledBufferBindings
      Counter for all sampled buffer bindings (i.e. with BindFlags::Sampled flag).
    std::uint32_t storageBufferBindings
      Counter for all storage buffer bindings (i.e. with BindFlags::Storage flag).
    std::uint32_t sampledTextureBindings
      Counter for all sampled texture bindings (i.e. with BindFlags::Sampled flag).
    std::uint32_t storageTextureBindings
      Counter for all sampled texture bindings (i.e. with BindFlags::Storage flag).
    std::uint32_t samplerBindings
      Counter for all sampler-state bindings.
    std::uint32_t resourceHeapBindings
      Counter for all resource heap bindings.
    std::uint32_t graphicsPipelineBindings
      Counter for all graphics pipeline state bindings.
    std::uint32_t computePipelineBindings
      Counter for all compute pipeline state 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 bufferFills
      Counter for all buffer fills 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 renderPassSections
      Counter for all command buffer sections that are enclosed by a call to `BeginRenderPass` *and* `EndRenderPass`*.*
    std::uint32_t streamOutputSections
      Counter for all command buffer sections that are enclosed by a call to `BeginStreamOutput` *and* `EndStreamOut`
    std::uint32_t querySections
      Counter for all command buffer sections that are enclosed by a call to `BeginQuery` *and* `EndQuery`*.*
    std::uint32_t renderConditionSections
      Counter for all command buffer sections that are enclosed by a call to `BeginRenderCondition` *and* `EndRender`
    std::uint32_t drawCommands
      Counter for all draw commands.
    std::uint32_t dispatchCommands

Counter for dispatch compute commands.

std::uint32_t commandBufferSubmittions

Counter for all command buffers that were submitted to the queue.

std::uint32_t commandBufferEncodings

Counter for all command buffer encodings that are enclosed by a call to `Begin` and `End`.

std::uint32_t fenceSubmissions

Counter for all fences that were submitted to the queue.

}

std::uint32_t values [33]

All proflile values as linear array.

};

- std::vector< ProfileTimeRecord > timeRecords

  List of all time records for this frame profile.

10.49.1 Detailed Description

Profile of a rendered frame.

See also

RenderingProfiler::NextFrame

10.49.2 Constructor & Destructor Documentation

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

Default constructor that initializes all counter values to zero.

10.49.3 Member Function Documentation

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

Accumulates the specified profile with this profile.

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

Clears all counter values.

10.49.4 Member Data Documentation

10.49.4.1 union { ... }

10.49.4.2 std::uint32_t LLGL::FrameProfile::attachmentClears

Counter for all framebuffer attachment clear operations.

See also

CommandBuffer::Clear
CommandBuffer::ClearAttachments

10.49.4.3 std::uint32_t LLGL::FrameProfile::bufferCopies

Counter for all buffer copies during command encoding.

See also

CommandBuffer::CopyBuffer

10.49.4.4 std::uint32_t LLGL::FrameProfile::bufferFills

Counter for all buffer fills during command encoding.

See also

CommandBuffer::FillBuffer

10.49.4.5 std::uint32_t LLGL::FrameProfile::bufferMappings

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

See also

RenderSystem::MapBuffer.
RenderSystem::UnmapBuffer.

10.49.4.6 std::uint32_t LLGL::FrameProfile::bufferReads

Counter for all buffer write operations outside of command encoding.

Todo Not available yet.

10.49.4.7 std::uint32_t LLGL::FrameProfile::bufferUpdates

Counter for all buffer updates during command encoding.

See also

CommandBuffer::UpdateBuffer

10.49.4.8 std::uint32_t LLGL::FrameProfile::bufferWrites

Counter for all buffer write operations outside of command encoding.

See also

RenderSystem::WriteBuffer

10.49.4.9 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

10.49.4.10 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 ∗)

10.49.4.11 std::uint32_t LLGL::FrameProfile::computePipelineBindings

Counter for all compute pipeline state bindings.

See also

CommandBuffer::SetPipelineState

10.49.4.12 std::uint32_t LLGL::FrameProfile::constantBufferBindings

Counter for all individual constant buffer bindings.

See also

CommandBuffer::SetResource

10.49.4.13 std::uint32_t LLGL::FrameProfile::dispatchCommands

Counter for dispatch compute commands.

See also

CommandBuffer::Dispatch

10.49.4.14 std::uint32_t LLGL::FrameProfile::drawCommands

Counter for all draw commands.

See also

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

10.49.4.15 std::uint32_t LLGL::FrameProfile::fenceSubmissions

Counter for all fences that were submitted to the queue.

See also

CommandQueue::Submit(Fence&)

10.49.4.16 std::uint32_t LLGL::FrameProfile::graphicsPipelineBindings

Counter for all graphics pipeline state bindings.

See also

CommandBuffer::SetPipelineState

10.49.4.17 std::uint32_t LLGL::FrameProfile::indexBufferBindings

Counter for all index buffer bindings.

See also

CommandBuffer::SetIndexBuffer

10.49.4.18 std::uint32_t LLGL::FrameProfile::mipMapsGenerations

Counter for all MIP-map generations.

See also

CommandBuffer::GenerateMips

10.49.4.19 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

10.49.4.20 std::uint32_t LLGL::FrameProfile::renderConditionSections

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

See also

CommandBuffer::BeginRenderCondition
CommandBuffer::EndRenderCondition

10.49.4.21 std::uint32_t LLGL::FrameProfile::renderPassSections

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

See also

CommandBuffer::BeginRenderPass
CommandBuffer::EndRenderPass

10.49.4.22 std::uint32_t LLGL::FrameProfile::resourceHeapBindings

Counter for all resource heap bindings.

See also

CommandBuffer::SetResourceHeap

10.49.4.23 std::uint32_t LLGL::FrameProfile::sampledBufferBindings

Counter for all sampled buffer bindings (i.e. with BindFlags::Sampled flag).

See also

CommandBuffer::SetResource

10.49.4.24 std::uint32_t LLGL::FrameProfile::sampledTextureBindings

Counter for all sampled texture bindings (i.e. with BindFlags::Sampled flag).

See also

CommandBuffer::SetResource

10.49.4.25 std::uint32_t LLGL::FrameProfile::samplerBindings

Counter for all sampler-state bindings.

See also

CommandBuffer::SetResource

10.49.4.26 std::uint32_t LLGL::FrameProfile::storageBufferBindings

Counter for all storage buffer bindings (i.e. with BindFlags::Storage flag).

See also

CommandBuffer::SetResource

10.49.4.27 std::uint32_t LLGL::FrameProfile::storageTextureBindings

Counter for all sampled texture bindings (i.e. with BindFlags::Storage flag).

See also

CommandBuffer::SetResource

10.49.4.28 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

10.49.4.29 std::uint32_t LLGL::FrameProfile::textureCopies

Counter for all texture copies during command encoding.

See also

CommandBuffer::CopyTexture.

10.49.4.30 std::uint32_t LLGL::FrameProfile::textureReads

Counter for all texture write operations outside of command encoding.

See also

RenderSystem::ReadTexture.

10.49.4.31 std::uint32_t LLGL::FrameProfile::textureWrites

Counter for all texture write operations outside of command encoding.

See also

RenderSystem::WriteTexture.

10.49.4.32 std::vector<ProfileTimeRecord> LLGL::FrameProfile::timeRecords

List of all time records for this frame profile.

See also

RenderingProfiler::timeRecordingEnabled

10.49.4.33 std::uint32_t LLGL::FrameProfile::values[33]

All proflile values as linear array.

10.49.4.34 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

10.50 LLGL::GraphicsPipelineDescriptor Struct Reference

Graphics pipeline state descriptor structure.

```
#include <PipelineStateFlags.h>
```

Public Attributes

- const PipelineLayout * pipelineLayout = nullptr

  Specifies an optional pipeline layout for the graphics pipeline. By default null.
- const RenderPass * renderPass = nullptr

  Specifies an optional render pass. By default null.
- 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").
- 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 static viewports. If empty, the viewports must be set dynamically with the command buffer.
- std::vector< Scissor > scissors

  Specifies an optional list of static 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.
- TessellationDescriptor tessellation

  Specifies the tessellation pipeline state.

10.50.1 Detailed Description

Graphics pipeline state descriptor structure.

Remarks

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

See also

RenderSystem::CreatePipelineState

10.50.2 Member Data Documentation

10.50.2.1 BlendDescriptor LLGL::GraphicsPipelineDescriptor::blend

Specifies the state descriptor for the blend stage.

10.50.2.2 DepthDescriptor LLGL::GraphicsPipelineDescriptor::depth

Specifies the depth state for the depth-stencil stage.

10.50.2.3 Shader* LLGL::GraphicsPipelineDescriptor::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.

10.50.2.4 Shader* LLGL::GraphicsPipelineDescriptor::geometryShader = nullptr

Specifies an optional geometry shader.

Remarks

This shader may also have a stream output.

10.50.2.5 const PipelineLayout* LLGL::GraphicsPipelineDescriptor::pipelineLayout = nullptr

Specifies an optional pipeline layout for the graphics pipeline. By default null.

Remarks

This layout determines at which slots buffer resources will be bound. If this is null, a default layout will be used that is only compatible with graphics pipelines that have no input and output vertex attributes and no binding points.

See also

RenderSystem::CreatePipelineLayout

10.50.2.6 PrimitiveTopology LLGL::GraphicsPipelineDescriptor::primitiveTopology = PrimitiveTopology::TriangleList

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

See also

PrimitiveTopology

10.50.2.7 RasterizerDescriptor LLGL::GraphicsPipelineDescriptor::rasterizer

Specifies the state for the rasterizer stage.

10.50.2.8 const RenderPass* LLGL::GraphicsPipelineDescriptor::renderPass = nullptr

Specifies an optional render pass. By default null.

Remarks

If this is null, the render pass of the SwapChain 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
RenderSystem::CreateRenderPass

10.50.2.9 std::vector<Scissor> LLGL::GraphicsPipelineDescriptor::scissors

Specifies an optional list of static 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

10.50.2.10 StencilDescriptor LLGL::GraphicsPipelineDescriptor::stencil

Specifies the stencil state for the depth-stencil stage.

10.50.2.11 Shader* LLGL::GraphicsPipelineDescriptor::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

10.50.2.12 TessellationDescriptor LLGL::GraphicsPipelineDescriptor::tessellation

Specifies the tessellation pipeline state.

Remarks

This is only used to configure a few tessellation states on the CPU side for the Metal backend. All other backends ignore this member silently.

Note

Only supported with: Metal.

10.50.2.13 Shader* LLGL::GraphicsPipelineDescriptor::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

10.50.2.14 Shader* LLGL::GraphicsPipelineDescriptor::vertexShader = nullptr

Specifies the vertex shader.

Remarks

Each graphics pipeline must have at least a vertex shader. Therefore, this must never be null when a graphics PSO is created. With OpenGL, this shader may also have a stream output.

10.50.2.15 std::vector<Viewport> LLGL::GraphicsPipelineDescriptor::viewports

Specifies an optional list of static 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:

- PipelineStateFlags.h

10.51 LLGL::GrowStrategyAddHalf Struct Reference

Container capacity grow strategy to increase the capacity to 150%.

```
#include <SmallVector.h>
```

Static Public Member Functions

- static std::size_t IncreaseToFit (std::size_t size)

  Returns the increased size by adding half of the input value, effectively returning 150% the input size. This is always greater than or equal to `size`.

10.51.1 Detailed Description

Container capacity grow strategy to increase the capacity to 150%.

Remarks

This is the default grow strategy for the SmallVector container.

See also

SmallVector

10.51.2 Member Function Documentation

10.51.2.1 static std::size_t LLGL::GrowStrategyAddHalf::IncreaseToFit (std::size_t *size*) [inline], [static]

Returns the increased size by adding half of the input value, effectively returning 150% the input size. This is always greater than or equal to `size`.

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

- SmallVector.h

10.52 LLGL::GrowStrategyDouble Struct Reference

Container capacity grow strategy to increase the capacity to 200%.

```
#include <SmallVector.h>
```

Static Public Member Functions

- static std::size_t IncreaseToFit (std::size_t size)

  Returns the increased size by multiplying by two, effectively doubling the input size. This is always greater than or equal to `size`.

10.52.1 Detailed Description

Container capacity grow strategy to increase the capacity to 200%.

See also

SmallVector

10.52.2 Member Function Documentation

10.52.2.1 static std::size_t LLGL::GrowStrategyDouble::IncreaseToFit (std::size_t *size*) [inline], [static]

Returns the increased size by multiplying by two, effectively doubling the input size. This is always greater than or equal to `size`.

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

- SmallVector.h

10.53 LLGL::GrowStrategyRoundUpPow2 Struct Reference

Container capacity grow strategy to increase the capacity to the next higher power of two.

```
#include <SmallVector.h>
```

Static Public Member Functions

- static unsigned int Round (unsigned int v)
- static unsigned long Round (unsigned long v)
- static unsigned long long Round (unsigned long long v)
- static std::size_t IncreaseToFit (std::size_t size)

  Returns the increased size by rounding up to the next power of two. This is always greater than or equal to `size`.

10.53.1 Detailed Description

Container capacity grow strategy to increase the capacity to the next higher power of two.

See also

SmallVector

10.53.2 Member Function Documentation

10.53.2.1 static std::size_t LLGL::GrowStrategyRoundUpPow2::IncreaseToFit (std::size_t *size*) [inline], [static]

Returns the increased size by rounding up to the next power of two. This is always greater than or equal to `size`.

10.53.2.2 static unsigned int LLGL::GrowStrategyRoundUpPow2::Round (unsigned int *v*) [inline], [static]

10.53.2.3 static unsigned long LLGL::GrowStrategyRoundUpPow2::Round (unsigned long *v*) [inline], [static]

10.53.2.4 static unsigned long long LLGL::GrowStrategyRoundUpPow2::Round (unsigned long long *v*) [inline], [static]

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

- SmallVector.h

10.54 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, unsigned threadCount=0)

  Converts the image format and data type.
- void Resize (const Extent3D &extent)

  Resizes the image and resets the image buffer.
- void Resize (const Extent3D &extent, const ColorRGBAd &fillColor)

  Resizes the image and initializes the new pixels with the specified color.
- void Resize (const Extent3D &extent, const ColorRGBAd &fillColor, const Offset3D &offset)

  Resizes the image, moves the previous pixels by an offset, and initializes the new pixels outside the previous extent with the specified color.
- void Resize (const Extent3D &extent, const SamplerFilter filter)

  Resizes the image and resamples the pixels from the previous image buffer.
- void Swap (Image &rhs)

Swaps all attributes with the specified image.

- void Reset ()

  Resets all image attributes to its default values.

- void Reset (const Extent3D &extent, const ImageFormat format, const DataType dataType, ByteBuffer &&data)

  Resets all image attributes to the specified values.

- ByteBuffer Release ()

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

- void Blit (Offset3D dstRegionOffset, const Image &srcImage, Offset3D srcRegionOffset, Extent3D srcRegionExtent)

  Copies a region of the specified source image into this image.

- void Fill (Offset3D offset, Extent3D extent, const ColorRGBAd &fillColor)

  Fills a region of this image by the specified color.

- void ReadPixels (const Offset3D &offset, const Extent3D &extent, const DstImageDescriptor &imageDesc, unsigned 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, unsigned threadCount=0)

  Writes a region of pixels to this image from the source image buffer specified by 'imageDesc'.

- void MirrorYZPlane ()

  Mirrors the image at the YZ plane.

- void MirrorXZPlane ()

  Mirrors the image at the XZ plane.

- void MirrorXYPlane ()

  Mirrors the image at the XY plane.

- SrcImageDescriptor GetSrcDesc () const

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

- DstImageDescriptor GetDstDesc ()

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

- const Extent3D & GetExtent () const

  Returns the extent of the image as 3D vector.

- ImageFormat GetFormat () const

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

- DataType GetDataType () const

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

- const void * GetData () const

  Returns the image data buffer as constant raw pointer.

- void * GetData ()

  Returns the image data buffer as raw pointer.

- std::uint32_t GetBytesPerPixel () const

  Returns the size (in bytes) for each pixel.

- std::uint32_t GetRowStride () const

  Returns the stride (in bytes) for each row.

- std::uint32_t GetDepthStride () const

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

- std::uint32_t GetNumPixels () const

  Returns the number of pixels this image has.

- std::uint32_t GetDataSize () const

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

- bool IsRegionInside (const Offset3D &offset, const Extent3D &extent) const

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

10.54.1 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

10.54.2 Constructor & Destructor Documentation

10.54.2.1 LLGL::Image::Image ( ) [default]

10.54.2.2 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

10.54.2.3 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

10.54.2.4 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&&)

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

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

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

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

10.54.3 Member Function Documentation

10.54.3.1 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

10.54.3.2 void LLGL::Image::Convert (const ImageFormat *format*, const DataType *dataType*, unsigned *threadCount* = 0)

Converts the image format and data type.

See also

ConvertImageBuffer

10.54.3.3 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.

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

Returns the size (in bytes) for each pixel.

See also

GetFormat
ImageFormatSize
GetDataType
DataTypeSize

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

Returns the image data buffer as constant raw pointer.

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

Returns the image data buffer as raw pointer.

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

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

See also

GetBytesPerPixel
GetNumPixels

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

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

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

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

10.54.3.10 DstImageDescriptor LLGL::Image::GetDstDesc ( )

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

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

Returns the extent of the image as 3D vector.

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

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

10.54.3.13 std::uint32_t LLGL::Image::GetNumPixels ( ) const

Returns the number of pixels this image has.

Remarks

This is equivalent to the following code example:

```
const auto& extent = myImage.GetExtent();
return extent.width * extent.height * extent.depth;
```

See also

GetExtent

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

Returns the stride (in bytes) for each row.

10.54.3.15 SrcImageDescriptor LLGL::Image::GetSrcDesc ( ) const

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

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

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

10.54.3.17 void LLGL::Image::MirrorXYPlane ( )

Mirrors the image at the XY plane.

Todo Not implemented yet

10.54.3.18 void LLGL::Image::MirrorXZPlane ( )

Mirrors the image at the XZ plane.

Todo Not implemented yet

10.54.3.19 void LLGL::Image::MirrorYZPlane ( )

Mirrors the image at the YZ plane.

Todo Not implemented yet

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

Copy operator which copies the entire image buffer and attributes.

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

Move operator which takes the ownership of the image buffer.

10.54.3.22 void LLGL::Image::ReadPixels (const Offset3D & *offset*, const Extent3D & *extent*, const DstImageDescriptor & *imageDesc*, unsigned *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

10.54.3.23 ByteBuffer LLGL::Image::Release ( )

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

10.54.3.24 void LLGL::Image::Reset ( )

Resets all image attributes to its default values.

10.54.3.25 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
AllocateByteBuffer

10.54.3.26 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!

10.54.3.27 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

10.54.3.28 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

10.54.3.29 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.

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

Swaps all attributes with the specified image.

10.54.3.31 void LLGL::Image::WritePixels (const Offset3D & *offset*, const Extent3D & *extent*, const SrcImageDescriptor & *imageDesc*, unsigned *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

10.55 LLGL::Input Class Reference

Default window event listener to receive user input.

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

Inheritance diagram for LLGL::Input:

Public Member Functions

- Input ()

  Default initializes the input handler without assigning to any surface.
- Input (Surface &surface)

  Assigns an event listener for this input handler to the specified surface.
- ~Input ()

  Releases the internal data.
- void Listen (Surface &surface)

  Adds an event listener for this input handler to the specified surface.
- void Drop (Surface &surface)

  Removes the event listener for this input handler from the specified surface.
- 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 UTF8String & GetEnteredChars () const

  Returns the entered characters.
- unsigned GetAnyKeyCount () const

  Returns the number of any keys being pressed.

Additional Inherited Members

10.55.1 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 ...
}
```

Todo Make Window::EventListener a member of Input instead of extending it. Also add a member of Canvas::EventListener and add touch event functions.

10.55.2 Constructor & Destructor Documentation

10.55.2.1 LLGL::Input::Input ( )

Default initializes the input handler without assigning to any surface.

10.55.2.2 LLGL::Input::Input (Surface & *surface*)

Assigns an event listener for this input handler to the specified surface.

10.55.2.3 LLGL::Input::~Input ( )

Releases the internal data.

10.55.3 Member Function Documentation

10.55.3.1 void LLGL::Input::Drop (Surface & *surface*)

Removes the event listener for this input handler from the specified surface.

10.55.3.2 unsigned LLGL::Input::GetAnyKeyCount ( ) const

Returns the number of any keys being pressed.

10.55.3.3 const UTF8String& LLGL::Input::GetEnteredChars ( ) const

Returns the entered characters.

10.55.3.4 const Offset2D& LLGL::Input::GetMouseMotion ( ) const

Returns the global mouse motion.

10.55.3.5 const Offset2D& LLGL::Input::GetMousePosition ( ) const

Returns the local mouse position.

10.55.3.6 int LLGL::Input::GetWheelMotion ( ) const

Returns the mouse wheel motion.

10.55.3.7 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.

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

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

10.55.3.9 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).

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

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

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

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

10.55.3.12 void LLGL::Input::Listen (Surface & *surface*)

Adds an event listener for this input handler to the specified surface.

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

- Input.h

10.56 LLGL::Interface Class Reference

Base class for all interfaces in LLGL.

```
#include <Interface.h>
```

Inheritance diagram for LLGL::Interface:

Public Member Functions

- virtual bool IsInstanceOf (int id) const

  Returns true if this object is an instance of the specified interface.

Additional Inherited Members

10.56.1 Detailed Description

Base class for all interfaces in LLGL.

See also

Display
RenderSystem
RenderSystemChild
Surface

10.56.2 Member Function Documentation

10.56.2.1 virtual bool LLGL::Interface::IsInstanceOf (int *id*) const [virtual]

Returns true if this object is an instance of the specified interface.

Remarks

Can be used for a more efficient run-time type information (RTTI) than the default C++ mechanism. This can be used like in the following example:

```
LLGL::Surface* mySurface = ...
if (mySurface->IsInstanceOf(LLGL::InterfaceID::Window))
{
    LLGL::Window* myWindow = static_cast<LLGL::Window*>(mySurface);
    ...
}
```

See also

LLGL::CastTo

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

- Interface.h

10.57 LLGL::InterfaceID Struct Reference

Enumeration of all interfaces in LLGL.

`#include <InterfaceID.h>`

Public Types

- enum {
  Interface, Display, Surface, Canvas,
  Canvas_EventListener, Window, Window_EventListener, Input,
  RenderSystem, RenderSystemChild, Report, Buffer,
  BufferArray, CommandBuffer, CommandQueue, Fence,
  PipelineLayout, PipelineState, QueryHeap, RenderPass,
  Resource, ResourceHeap, Sampler, Shader,
  Texture, RenderTarget, SwapChain, Reserved }

10.57.1 Detailed Description

Enumeration of all interfaces in LLGL.

Remarks

The ID for a base class must always have a smaller value than the ID of its sub classes. Therefore, all base classes must be declared before their sub classes in this enumeration.

See also

Interface::IsInstanceOf
Interface::AsInstanceOf

10.57.2 Member Enumeration Documentation

10.57.2.1 anonymous enum

Enumerator

Interface Base class of all public interfaces in LLGL.

See also

Interface

Display Extends Interface.

See also

Display

Surface Extends Interface.

See also

Surface

Canvas Extends Interface.

See also

Canvas

Canvas_EventListener Extends Interface.

See also

Canvas::EventListener

Window Extends Interface.

See also

Window

Window_EventListener Extends Interface.

See also

Window::EventListener

Input Extends Interface.

See also

Input

RenderSystem Extends Interface.

See also

RenderSystem

RenderSystemChild Extends Interface.

See also

RenderSystemChild

Report Extends Interface.

See also

Report

Buffer Extends RenderSystemChild.

See also

Buffer

BufferArray Extends RenderSystemChild.

See also

BufferArray

CommandBuffer Extends RenderSystemChild.

See also

CommandBuffer

CommandQueue Extends RenderSystemChild.

See also

CommandQueue

Fence Extends RenderSystemChild.

See also

Fence

PipelineLayout Extends RenderSystemChild.

See also

PipelineLayout

PipelineState Extends RenderSystemChild.

See also

PipelineState

QueryHeap Extends RenderSystemChild.

See also

QueryHeap

RenderPass Extends RenderSystemChild.

See also

RenderPass

Resource Extends RenderSystemChild.

See also

Resource

ResourceHeap Extends RenderSystemChild.

See also

ResourceHeap

Sampler Extends RenderSystemChild.

See also

Sampler

Shader Extends RenderSystemChild.

See also

Shader

Texture Extends RenderSystemChild.

See also

Texture

RenderTarget Extends RenderSystemChild.

See also

RenderTarget

SwapChain Extends RenderTarget.

See also

SwapChain

Reserved Maximum reserved ID for interfaces.

Remarks

Use this value to write custom sub classes of an LLGL interface:

```
class MyWindow : public LLGL::Window {
    LLGL_DECLARE_INTERFACE(LLGL::InterfaceID::Reserved + 1
      );
    ...
};
LLGL_IMPLEMENT_INTERFACE(MyWindow, LLGL::Window);
```

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

- InterfaceID.h

10.58 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 StringView &text, const StringView &source, const StringView &groupName)

  Initializes the message with text, source, and group name information.
- 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.
- UTF8String ToReportString (const StringView &reportTypeName) const

  Returns a report string for this message.
- const UTF8String & GetText () const

  Returns the message text.
- const UTF8String & GetSource () const

  Returns the source function where this message occurred.
- const UTF8String & GetGroupName () const

  Returns the debug group name where this message occured.
- std::size_t GetOccurrences () const

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

  Returns true if this message has already been blocked.

Protected Member Functions

- void IncOccurrence ()

Friends

- class RenderingDebugger

10.58.1 Detailed Description

Rendering debugger message class.

Todo Rename to "Report"

10.58.2 Constructor & Destructor Documentation

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

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

10.58.2.3 LLGL::RenderingDebugger::Message::Message (const StringView & *text,* const StringView & *source,* const StringView & *groupName*)

Initializes the message with text, source, and group name information.

10.58.3 Member Function Documentation

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

Blocks further occurrences of this message.

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

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

10.58.3.3 const UTF8String& LLGL::RenderingDebugger::Message::GetGroupName ( ) const [inline]

Returns the debug group name where this message occured.

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

Returns the number of occurrences of this message.

10.58.3.5 const UTF8String& LLGL::RenderingDebugger::Message::GetSource ( ) const [inline]

Returns the source function where this message occurred.

10.58.3.6 const UTF8String& LLGL::RenderingDebugger::Message::GetText ( ) const [inline]

Returns the message text.

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

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

Returns true if this message has already been blocked.

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

10.58.3.10 UTF8String LLGL::RenderingDebugger::Message::ToReportString (const StringView & *reportTypeName*) const

Returns a report string for this message.

Parameters

in	*reportTypeName*	Specifies the name of the report type (e.g. "ERROR").

Returns

Constructed report string containing all information of this message.

10.58.4 Friends And Related Function Documentation

10.58.4.1 friend class RenderingDebugger [friend]

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

- RenderingDebugger.h

10.59 LLGL::MetalDependentStateDescriptor Struct Reference

Graphics API dependent state descriptor for the Metal renderer.

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

Public Attributes

- std::uint32_t tessFactorBufferSlot = 30

  Specifies the buffer slot for the internal tessellation factor buffer. By default 30, which is the maximum buffer slot.

10.59.1 Detailed Description

Graphics API dependent state descriptor for the Metal renderer.

Remarks

This descriptor is used to compensate a few differences between Metal and the other rendering APIs.

See also

CommandBuffer::SetGraphicsAPIDependentState

Todo Move into namespace Metal

10.59.2 Member Data Documentation

10.59.2.1 std::uint32_t LLGL::MetalDependentStateDescriptor::tessFactorBufferSlot = 30

Specifies the buffer slot for the internal tessellation factor buffer. By default 30, which is the maximum buffer slot.

Remarks

In the respective Metal tessellation kernel, this must refer to a buffer of type `MTLTriangleTessellationFactorsHalf` or `MTLQuadTessellationFactorsHalf`.

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

- CommandBufferFlags.h

10.60 LLGL::MiscFlags Struct Reference

Miscellaneous resource flags.

```
#include <ResourceFlags.h>
```

Public Types

- enum {
  DynamicUsage = (1 << 0), FixedSamples = (1 << 1), GenerateMips = (1 << 2), NoInitialData = (1 << 3), Append = (1 << 4), Counter = (1 << 5) }

10.60.1 Detailed Description

Miscellaneous resource flags.

See also

BufferDescriptor::miscFlags
TextureDescriptor::miscFlags

10.60.2 Member Enumeration Documentation

10.60.2.1 anonymous enum

Enumerator

DynamicUsage Hint to the renderer that the resource will be frequently updated from the CPU.

Remarks

This is useful for a constant buffer for instance, that is updated by the host program every frame.

See also

RenderSystem::WriteBuffer
RenderSystem::WriteTexture

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

FixedSamples Multi-sampled Texture resource has fixed sample locations.

Remarks

This can only be used with multi-sampled Texture resources (i.e. TextureType::Texture2DMS, TextureType::Texture2DMSArray).

GenerateMips Generates MIP-maps at texture creation time with the initial image data (if specified).

Remarks

To generate MIP-maps, the texture must be created with the binding flags BindFlags::Sampled and BindFlags::ColorAttachment, which is the default.
This can be used to generate all MIP-maps when a new texture is created without explicitly encoding the CommandBuffer::GenerateMips function. The number of MIP-maps being generated depends on the `mipLevels` attribute in TextureDescriptor.

See also

TextureDescriptor::mipLevels
CommandBuffer::GenerateMips

NoInitialData Specifies to ignore resource data initialization.

Remarks

If this is specified, a texture or buffer resource will stay uninitialized during creation and the content is undefined.

Append Enables a storage buffer to be used for `AppendStructuredBuffer` and `Consume↩StructuredBuffer` in HLSL only.

Remarks

This can only be used with buffers that also have the binding flag BindFlags::Storage and a `stride` greater than zero.
This cannot be used together with the MiscFlags::Counter bit.

Note

Only supported with: Direct3D 11, Direct3D 12.

See also

BufferDescriptor::stride
ResourceViewDescriptor::initialCount
https://docs.microsoft.com/en-us/windows/win32/api/d3d11/ne-d3d11-d3d11↩_buffer_uav_flag

Counter Enables the hidden counter in a storage buffer to be used for `RWStructuredBuffer` in HLSL only.

Remarks

This can only be used with buffers that also have the binding flag BindFlags::Storage and a `stride` greater than zero.
This cannot be used together with the MiscFlags::Append bit.

Note

Only supported with Direct3D 11, Direct3D 12.

See also

BufferDescriptor::stride
ResourceViewDescriptor::initialCount
https://docs.microsoft.com/en-us/windows/win32/api/d3d11/ne-d3d11-d3d11↩_buffer_uav_flag

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

- ResourceFlags.h

10.61 LLGL::NativeContextHandle Struct Reference

Android native context handle structure.

```
#include <AndroidNativeHandle.h>
```

Public Attributes

- ANativeWindow * parentWindow
- UIView * parentView
- ::Display * display
- ::Window parentWindow
- ::XVisualInfo * visual
- ::Colormap colorMap
- int screen
- NSWindow * parentWindow
- HWND parentWindow

10.61.1 Detailed Description

Android native context handle structure.

Win32 native context handle structure.

MacOS native context handle structure.

Linux native context handle structure.

iOS native context handle structure.

Remarks

This must be a POD (Plain-Old-Data) structure, so no default initialization is provided!

10.61.2 Member Data Documentation

10.61.2.1 ::Colormap LLGL::NativeContextHandle::colorMap

10.61.2.2 ::Display* LLGL::NativeContextHandle::display

10.61.2.3 UIView* LLGL::NativeContextHandle::parentView

10.61.2.4 ANativeWindow* LLGL::NativeContextHandle::parentWindow

10.61.2.5 NSWindow* LLGL::NativeContextHandle::parentWindow

10.61.2.6 ::Window LLGL::NativeContextHandle::parentWindow

10.61.2.7 HWND LLGL::NativeContextHandle::parentWindow

10.61.2.8 int LLGL::NativeContextHandle::screen

10.61.2.9 ::XVisualInfo* LLGL::NativeContextHandle::visual

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

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

10.62 LLGL::NativeHandle Struct Reference

Android native handle structure.

```
#include <AndroidNativeHandle.h>
```

Public Attributes

- ANativeWindow * window
- UIView * view
- ::Display * display
- ::Window window
- ::XVisualInfo * visual
- NSWindow * window
- HWND window

10.62.1 Detailed Description

Android native handle structure.

Win32 native handle structure.

MacOS native handle structure.

Linux native handle structure.

iOS native handle structure.

Remarks

This must be a POD (Plain-Old-Data) structure, so no default initialization is provided!

10.62.2 Member Data Documentation

10.62.2.1 ::Display* LLGL::NativeHandle::display

10.62.2.2 UIView* LLGL::NativeHandle::view

10.62.2.3 ::XVisualInfo* LLGL::NativeHandle::visual

10.62.2.4 ANativeWindow* LLGL::NativeHandle::window

10.62.2.5 NSWindow* LLGL::NativeHandle::window

10.62.2.6 ::Window LLGL::NativeHandle::window

10.62.2.7 HWND LLGL::NativeHandle::window

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

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

10.63 LLGL::NonCopyable Class Reference

Base class for all classes with virtual function tables in LLGL.

```
#include <NonCopyable.h>
```

Inheritance diagram for LLGL::NonCopyable:

Public Member Functions

- NonCopyable (const NonCopyable &)=delete
- NonCopyable & operator= (const NonCopyable &)=delete
- virtual ~NonCopyable ()=default

Protected Member Functions

- NonCopyable ()=default

10.63.1 Detailed Description

Base class for all classes with virtual function tables in LLGL.

Remarks

Sub classes of this interface cannot be copied on its own, since its copy constructor and copy operator are deleted functions.

See also

Interface

10.63.2 Constructor & Destructor Documentation

10.63.2.1 LLGL::NonCopyable::NonCopyable (const NonCopyable &) [delete]

10.63.2.2 virtual LLGL::NonCopyable::~NonCopyable ( ) [virtual],[default]

10.63.2.3 LLGL::NonCopyable::NonCopyable ( ) [protected],[default]

10.63.3 Member Function Documentation

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

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

- NonCopyable.h

10.64 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.

10.64.1 Detailed Description

2-Dimensional offset structure.

Remarks

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

10.64.2 Constructor & Destructor Documentation

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

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

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

10.64.3 Member Data Documentation

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

Offset X axis.

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

Offset Y axis.

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

- Types.h

10.65 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.

10.65.1 Detailed Description

3-Dimensional offset structure.

Remarks

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

10.65.2 Constructor & Destructor Documentation

10.65.2.1 LLGL::Offset3D::Offset3D ( ) [default]

10.65.2.2 LLGL::Offset3D::Offset3D (const Offset3D &) [default]

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

10.65.3 Member Data Documentation

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

Offset X axis.

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

Offset Y axis.

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

Offset Z axis.

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

- Types.h

10.66 LLGL::PipelineLayout Class Reference

Pipeline layout interface.

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

Inheritance diagram for LLGL::PipelineLayout:

Public Member Functions

- virtual std::uint32_t GetNumBindings () const =0

  Returns the number of resource bindings in this pipeline layout.

Additional Inherited Members

10.66.1 Detailed Description

Pipeline layout interface.

Remarks

An instance of this interface provides a layout for resource binding in a graphics or compute pipeline.

See also

RenderSystem::CreatePipelineLayout
GraphicsPipelineDescriptor::pipelineLayout
ResourceHeapDescriptor::pipelineLayout

10.66.2 Member Function Documentation

10.66.2.1 virtual std::uint32_t LLGL::PipelineLayout::GetNumBindings ( ) const [pure virtual]

Returns the number of resource bindings in this pipeline layout.

See also

PipelineLayoutDescriptor::bindings
ResourceHeap::GetNumDescriptorSets

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

- PipelineLayout.h

10.67 LLGL::PipelineLayoutDescriptor Struct Reference

Pipeline layout descritpor structure.

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

Public Attributes

- std::vector< BindingDescriptor > bindings

  List of layout resource bindings.

10.67.1 Detailed Description

Pipeline layout descritpor structure.

Remarks

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

See also

RenderSystem::CreatePipelineLayout

10.67.2 Member Data Documentation

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

List of layout resource bindings.

See also

ResourceHeap::GetNumDescriptorSets

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

- PipelineLayoutFlags.h

10.68 LLGL::PipelineState Class Reference

Graphics and compute pipeline state interface.

```
#include <PipelineState.h>
```

Inheritance diagram for LLGL::PipelineState:

Public Member Functions

- virtual const Report * GetReport () const =0

  Returns a pointer to the report or null if there is none.

Additional Inherited Members

10.68.1 Detailed Description

Graphics and compute pipeline state interface.

See also

RenderSystem::CreatePipelineState
CommandBuffer::SetPipelineState

10.68.2 Member Function Documentation

10.68.2.1 virtual const Report* LLGL::PipelineState::GetReport ( ) const [pure virtual]

Returns a pointer to the report or null if there is none.

Remarks

If there is a report, it might contain warnings and/or errors from the PSO and shader compilation process.

See also

Report

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

- PipelineState.h

10.69 LLGL::ProfileTimeRecord Struct Reference

Structure with annotation and elapsed time for a timer profile.

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

Public Attributes

- const char * annotation = ""

  Time record annotation, e.g. function name that was recorded from the CommandBuffer.
- std::uint64_t elapsedTime = 0

  Elapsed time (in nanoseconds) to execute the respective command.

10.69.1 Detailed Description

Structure with annotation and elapsed time for a timer profile.

See also

FrameProfile::timeRecords

10.69.2 Member Data Documentation

10.69.2.1 const char* LLGL::ProfileTimeRecord::annotation = ""

Time record annotation, e.g. function name that was recorded from the CommandBuffer.

10.69.2.2 std::uint64_t LLGL::ProfileTimeRecord::elapsedTime = 0

Elapsed time (in nanoseconds) to execute the respective command.

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

- RenderingProfiler.h

10.70 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)

10.70.1 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

10.70.2 Constructor & Destructor Documentation

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

10.70.3 Member Function Documentation

10.70.3.1 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

10.71 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.

10.71.1 Detailed Description

Query heap descriptor structure.

See also

RenderSystem::CreateQueryHeap

10.71.2 Member Data Documentation

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

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

10.71.2.2 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.

10.71.2.3 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

10.72 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.

10.72.1 Detailed Description

Query data structure for pipeline statistics.

Remarks

This structure is byte aligned, i.e. it can be reinterpret casted to a buffer in CPU memory space.

See also

QueryType::PipelineStatistics
CommandQueue::QueryResult
RenderingFeatures::hasPipelineStatistics
Direct3D11 counterpart `D3D11_QUERY_DATA_PIPELINE_STATISTICS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d11/ns-d3d11-d3d11_query_data_pipeline_statistics
Direct3D12 counterpart `D3D12_QUERY_DATA_PIPELINE_STATISTICS`: https://docs.microsoft.com/en-us/windows/desktop/api/d3d12/ns-d3d12-d3d12_query_data_pipeline_statistics
Vulkan counterpart `VkQueryPipelineStatisticFlagBits`: https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/VkQueryPipelineStatisticFlagBits.html

10.72.2 Member Data Documentation

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

Number of primitives that reached the primitive clipping stage.

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

Number of primitives that passed the primitive clipping stage.

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

Number of compute shader invocations.

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

Number of fragment shader invocations.

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

Number of geometry shader invocations.

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

Number of primitives generated by the geometry shader.

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

Number of primitives submitted to the input-assembly.

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

Number of vertices submitted to the input-assembly.

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

Number of tessellation-control shader invocations.

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

Number of tessellation-evaluation shader invocations.

10.72.2.11 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

10.73 LLGL::RasterizerDescriptor Struct Reference

Rasterizer state descriptor structure.

```
#include <PipelineStateFlags.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.
- 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 multiSampleEnabled = false

  Specifies whether multi-sampling 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.

10.73.1 Detailed Description

Rasterizer state descriptor structure.

See also

GraphicsPipelineDescriptor::rasterizer

10.73.2 Member Data Documentation

10.73.2.1 bool LLGL::RasterizerDescriptor::antiAliasedLineEnabled = false

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

10.73.2.2 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)
- Vulkan (if the extension `VK_EXT_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
https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/VK_EXT_conservative_rasterization.html
RenderingFeatures::hasConservativeRasterization

10.73.2.3 CullMode LLGL::RasterizerDescriptor::cullMode = CullMode::Disabled

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

10.73.2.4 DepthBiasDescriptor LLGL::RasterizerDescriptor::depthBias

Specifies the parameters to bias fragment depth values.

10.73.2.5 bool LLGL::RasterizerDescriptor::depthClampEnabled = false

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

10.73.2.6 bool LLGL::RasterizerDescriptor::discardEnabled = false

If enabled, primitives are discarded after optional stream-outputs but before the rasterization stage. By default disabled.

Note

Only supported with: OpenGL, Vulkan, Metal.

10.73.2.7 bool LLGL::RasterizerDescriptor::frontCCW = false

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

10.73.2.8 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

10.73.2.9 bool LLGL::RasterizerDescriptor::multiSampleEnabled = false

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

10.73.2.10 PolygonMode LLGL::RasterizerDescriptor::polygonMode = PolygonMode::Fill

Polygon render mode. By default PolygonMode::Fill.

10.73.2.11 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:

- PipelineStateFlags.h

10.74 LLGL::RendererConfigurationOpenGL Struct Reference

OpenGL profile descriptor structure.

```
#include <RendererConfiguration.h>
```

Public Attributes

- OpenGLContextProfile contextProfile = OpenGLContextProfile::CoreProfile

  Specifies the requested OpenGL context profile. By default OpenGLContextProfile::CoreProfile.
- int majorVersion = 0

  Specifies the requested OpenGL context major version. By default 0.
- int minorVersion = 0

  Specifies the requested OpenGL context minor version. By default 0.

10.74.1 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.

10.74.2 Member Data Documentation

10.74.2.1 OpenGLContextProfile LLGL::RendererConfigurationOpenGL::contextProfile = OpenGLContextProfile::CoreProfile

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

10.74.2.2 int LLGL::RendererConfigurationOpenGL::majorVersion = 0

Specifies the requested OpenGL context major version. By default 0.

Remarks

If both `majorVersion` and `minorVersion` are 0, the highest OpenGL version that is available on the host system will be choosen.
This member is ignored if `contextProfile` is OpenGLContextProfile::CompatibilityProfile.

10.74.2.3 int LLGL::RendererConfigurationOpenGL::minorVersion = 0

Specifies the requested OpenGL context minor version. By default 0.

Remarks

If both `majorVersion` and `minorVersion` are 0, the highest OpenGL version that is available on the host system will be choosen.
This member is ignored if `contextProfile` is OpenGLContextProfile::CompatibilityProfile.

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

- RendererConfiguration.h

10.75 LLGL::RendererConfigurationOpenGLES3 Struct Reference

OpenGL ES 3 profile descriptor structure.

```
#include <RendererConfiguration.h>
```

Public Attributes

- int majorVersion = 3

  Specifies the requested OpenGL ES context major version. Must be 3. By default 3.
- int minorVersion = 0

  Specifies the requested OpenGL ES context minor version. Must be 0, 1, or 2. By default 0.

10.75.1 Detailed Description

OpenGL ES 3 profile descriptor structure.

Todo Replace with RendererConfigurationOpenGL and make use of OpenGLContextProfile::ESProfile.

10.75.2 Member Data Documentation

10.75.2.1 int LLGL::RendererConfigurationOpenGLES3::majorVersion = 3

Specifies the requested OpenGL ES context major version. Must be 3. By default 3.

10.75.2.2 int LLGL::RendererConfigurationOpenGLES3::minorVersion = 0

Specifies the requested OpenGL ES context minor version. Must be 0, 1, or 2. By default 0.

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

- RendererConfiguration.h

10.76 LLGL::RendererConfigurationVulkan Struct Reference

Structure for a Vulkan renderer specific configuration.

```
#include <RendererConfiguration.h>
```

Public Attributes

- ApplicationDescriptor application

  Application descriptor used when a Vulkan debug or validation layer is enabled.
- ArrayView< const char * > enabledLayers

  List of Vulkan layers to enable. The ones that are not supported, will be ignored.
- std::uint64_t minDeviceMemoryAllocationSize = 1024*1024

  *Minimal allocation size for a device memory chunk. By default 1024*1024, i.e. 1 MB of VRAM.*
- bool reduceDeviceMemoryFragmentation = false

  Specifies whether fragmentation of the device memory blocks shall be kept low. By default false.

10.76.1 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.

10.76.2 Member Data Documentation

10.76.2.1 ApplicationDescriptor LLGL::RendererConfigurationVulkan::application

Application descriptor used when a Vulkan debug or validation layer is enabled.

See also

ApplicationDescriptor

10.76.2.2 ArrayView<const char*> LLGL::RendererConfigurationVulkan::enabledLayers

List of Vulkan layers to enable. The ones that are not supported, will be ignored.

Remarks

For example, the layer "`VK_LAYER_KHRONOS_validation`" can be used for a stronger validation.

10.76.2.3 std::uint64_t LLGL::RendererConfigurationVulkan::minDeviceMemoryAllocationSize = 1024*1024

Minimal allocation size for a device memory chunk. By default 1024*1024, i.e. 1 MB of VRAM.

Remarks

Vulkan only allows a limited set of device memory objects (e.g. 4096 on a GPU with 8 GB of VRAM). This member specifies the minimum size used for hardware memory allocation of such a memory chunk. The Vulkan render system automatically manages sub-region allocation and defragmentation.

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

10.76.2.4 bool LLGL::RendererConfigurationVulkan::reduceDeviceMemoryFragmentation = false

Specifies whether fragmentation of the device memory blocks shall be kept low. By default false.

Remarks

If this is true, each buffer and image allocation first tries to find a reusable device memory block within a single VkDeviceMemory chunk (which might be potentially slower). Whenever a VkDeviceMemory chunk is full, the memory manager tries to reduce fragmentation anyways.

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

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

- RendererConfiguration.h

10.77 LLGL::RendererID Struct Reference

Renderer identification number enumeration.

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

Static Public Attributes

- static constexpr int Undefined = 0x00000000

  Undefined ID number.
- static constexpr int Null = 0x00000001

  ID number for a Null renderer. This renderer does not render anything but provides the same interface for debugging purposes.
- static constexpr int OpenGL = 0x00000002

  ID number for an OpenGL renderer.
- static constexpr int OpenGLES1 = 0x00000003

  ID number for an OpenGL ES 1 renderer.
- static constexpr int OpenGLES2 = 0x00000004

  ID number for an OpenGL ES 2 renderer.
- static constexpr int OpenGLES3 = 0x00000005

  ID number for an OpenGL ES 3 renderer.
- static constexpr int Direct3D9 = 0x00000006

  ID number for a Direct3D 9 renderer.
- static constexpr int Direct3D10 = 0x00000007

  ID number for a Direct3D 10 renderer.
- static constexpr int Direct3D11 = 0x00000008

  ID number for a Direct3D 11 renderer.
- static constexpr int Direct3D12 = 0x00000009

  ID number for a Direct3D 12 renderer.
- static constexpr int Vulkan = 0x0000000a

  ID number for a Vulkan renderer.
- static constexpr int Metal = 0x0000000b

  ID number for a Metal renderer.
- static constexpr int Reserved = 0x000000ff

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

10.77.1 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

10.77.2 Member Data Documentation

10.77.2.1 constexpr int LLGL::RendererID::Direct3D10 = 0x00000007 [static]

ID number for a Direct3D 10 renderer.

10.77.2.2 constexpr int LLGL::RendererID::Direct3D11 = 0x00000008 [static]

ID number for a Direct3D 11 renderer.

10.77.2.3 constexpr int LLGL::RendererID::Direct3D12 = 0x00000009 [static]

ID number for a Direct3D 12 renderer.

10.77.2.4 constexpr int LLGL::RendererID::Direct3D9 = 0x00000006 [static]

ID number for a Direct3D 9 renderer.

10.77.2.5 constexpr int LLGL::RendererID::Metal = 0x0000000b [static]

ID number for a Metal renderer.

10.77.2.6 constexpr int LLGL::RendererID::Null = 0x00000001 [static]

ID number for a Null renderer. This renderer does not render anything but provides the same interface for debugging purposes.

10.77.2.7 constexpr int LLGL::RendererID::OpenGL = 0x00000002 [static]

ID number for an OpenGL renderer.

10.77.2.8 constexpr int LLGL::RendererID::OpenGLES1 = 0x00000003 [static]

ID number for an OpenGL ES 1 renderer.

10.77.2.9 constexpr int LLGL::RendererID::OpenGLES2 = 0x00000004 [static]

ID number for an OpenGL ES 2 renderer.

10.77.2.10 constexpr int LLGL::RendererID::OpenGLES3 = 0x00000005 [static]

ID number for an OpenGL ES 3 renderer.

10.77.2.11 constexpr int LLGL::RendererID::Reserved = 0x000000ff [static]

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

10.77.2.12 constexpr int LLGL::RendererID::Undefined = 0x00000000 [static]

Undefined ID number.

10.77.2.13 constexpr int LLGL::RendererID::Vulkan = 0x0000000a [static]

ID number for a Vulkan renderer.

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

- RenderSystemFlags.h

10.78 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").
- std::vector< std::string > extensionNames

  List of enabled renderer extensions (e.g. "GL_ARB_direct_state_access" or "VK_EXT_conditional_rendering").

10.78.1 Detailed Description

Renderer basic information structure.

See also

RenderSystem::GetRendererInfo

10.78.2 Member Data Documentation

10.78.2.1 std::string LLGL::RendererInfo::deviceName

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

10.78.2.2 std::vector<std::string> LLGL::RendererInfo::extensionNames

List of enabled renderer extensions (e.g. "GL_ARB_direct_state_access" or "VK_EXT_conditional_rendering").

10.78.2.3 std::string LLGL::RendererInfo::rendererName

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

10.78.2.4 std::string LLGL::RendererInfo::shadingLanguageName

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

10.78.2.5 std::string LLGL::RendererInfo::vendorName

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

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

- RenderSystemFlags.h

10.79 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 hardware texture formats.
- RenderingFeatures features

  Specifies all supported hardware features.
- RenderingLimits limits

  Specifies all rendering limitations.

10.79.1 Detailed Description

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

See also

RenderSystem::GetRenderingCaps

10.79.2 Member Data Documentation

10.79.2.1 ClippingRange LLGL::RenderingCapabilities::clippingRange = ClippingRange::ZeroToOne

Specifies the clipping depth range.

10.79.2.2 RenderingFeatures LLGL::RenderingCapabilities::features

Specifies all supported hardware features.

Remarks

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

10.79.2.3 RenderingLimits LLGL::RenderingCapabilities::limits

Specifies all rendering limitations.

Remarks

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

10.79.2.4 ScreenOrigin LLGL::RenderingCapabilities::screenOrigin = ScreenOrigin::UpperLeft

Screen coordinate system origin.

Remarks

This determines the native coordinate space of viewports, scissors, and framebuffers. If the native screen origin is lower-left, LLGL emulates it to always maintain the upper-left as the screen origin.

10.79.2.5 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

10.79.2.6 std::vector<Format> LLGL::RenderingCapabilities::textureFormats

Specifies the list of supported hardware texture formats.

See also

Format

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

- RenderSystemFlags.h

10.80 LLGL::RenderingDebugger Class Reference

Rendering debugger interface.

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

Classes

- class Message

  Rendering debugger message class.

Public Member Functions

- RenderingDebugger ()

  Initializes the internal data.
- virtual ~RenderingDebugger ()

  Release the internal data.
- void SetSource (const char *source)

  Sets the new source function name.
- void SetDebugGroup (const char *name)

  Sets the new debug group name.
- void PostError (const ErrorType type, const StringView &message)

  Posts an error message.
- void PostWarning (const WarningType type, const StringView &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.

10.80.1 Detailed Description

Rendering debugger interface.

Remarks

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

10.80.2 Constructor & Destructor Documentation

10.80.2.1 LLGL::RenderingDebugger::RenderingDebugger ( )

Initializes the internal data.

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

Release the internal data.

10.80.3 Member Function Documentation

10.80.3.1 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

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

Callback function when a warning was posted.

See also

RenderingDebugger::PostWarning
OnError

10.80.3.3 void LLGL::RenderingDebugger::PostError (const ErrorType *type*, const StringView & *message*)

Posts an error message.

Parameters

in	*type*	Specifies the type of error.
in	*message*	Specifies the string which describes the failure.

10.80.3.4 void LLGL::RenderingDebugger::PostWarning (const WarningType *type*, const StringView & *message*)

Posts a warning message.

Parameters

in	*type*	Specifies the type of error.
in	*message*	Specifies the string which describes the warning.

10.80.3.5 void LLGL::RenderingDebugger::SetDebugGroup (const char * *name*)

Sets the new debug group name.

Parameters

in	*name*	Pointer to a null terminated string that specifies the name. If this is null, the debug group is disabled.

Note

This function only stores the pointer. Hence, the pointer must be valid until a new pointer or a null pointer is set.

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

Sets the new source function name.

Parameters

in	*source*	Pointer to a null terminated string that specifies the name. If this is null, the source is disabled.

Note

This function only stores the pointer. Hence, the pointer must be valid until a new pointer or a null pointer is set.

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

- RenderingDebugger.h

10.81 LLGL::RenderingFeatures Struct Reference

Contains the attributes for all supported rendering features.

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

Public Attributes

- bool hasDirectResourceBinding = false

  Specifies whether the render system supports binding individual resources directly instead of a resource heap. This does not exclude support of resource heaps, because these are always supported.
- bool hasRenderTargets = false

  Specifies whether render targets (also "framebuffer objects") are supported.
- bool has3DTextures = false

  Specifies whether 3D textures are supported.
- bool hasCubeTextures = false

  Specifies whether cube textures are supported.
- bool hasArrayTextures = false

  Specifies whether 1D- and 2D array textures are supported.
- bool hasCubeArrayTextures = false

  Specifies whether cube array textures are supported.
- bool hasMultiSampleTextures = false

  Specifies whether multi-sample textures are supported.
- bool hasTextureViews = false

  Specifies whether texture views are supported.
- bool hasTextureViewSwizzle = false

  Specifies whether texture views can have swizzling (a.k.a. component mapping).
- bool hasBufferViews = false

  Specifies whether buffer views are supported.
- bool hasSamplers = false
- bool hasConstantBuffers = false

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

  Specifies whether storage buffers (also "read/write buffers") are supported.
- bool hasUniforms = false

  Specifies whether individual shader uniforms are supported.
- bool hasGeometryShaders = false

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

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

  Specifies whether tessellator stage is supported.
- bool hasComputeShaders = false

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

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

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

  Specifies whether indirect draw commands are supported.
- bool hasViewportArrays = false

  Specifies whether multiple viewports, depth-ranges, and scissors at once are supported.
- bool hasConservativeRasterization = false

  Specifies whether conservative rasterization is supported.
- bool hasStreamOutputs = false

  Specifies whether stream-output is supported.
- bool hasLogicOp = false

Specifies whether logic fragment operations are supported.

- bool hasPipelineStatistics = false

  Specifies whether queries for pipeline statistics are supported.

- bool hasRenderCondition = false

  Specifies whether queries for conditional rendering are supported.

10.81.1 Detailed Description

Contains the attributes for all supported rendering features.

See also

RenderingCapabilities::features

10.81.2 Member Data Documentation

10.81.2.1 bool LLGL::RenderingFeatures::has3DTextures = false

Specifies whether 3D textures are supported.

See also

TextureType::Texture3D

10.81.2.2 bool LLGL::RenderingFeatures::hasArrayTextures = false

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

See also

TextureType::Texture1DArray
TextureType::Texture2DArray

10.81.2.3 bool LLGL::RenderingFeatures::hasBufferViews = false

Specifies whether buffer views are supported.

Remarks

Buffer views can shared their data with another buffer resource in a different range and format.

See also

BufferViewDescriptor

10.81.2.4 bool LLGL::RenderingFeatures::hasComputeShaders = false

Specifies whether compute shaders are supported.

See also

ShaderType::Compute
CommandBuffer::Dispatch
CommandBuffer::DispatchIndirect

10.81.2.5 bool LLGL::RenderingFeatures::hasConservativeRasterization = false

Specifies whether conservative rasterization is supported.

See also

RasterizerDescriptor::conservativeRasterization

10.81.2.6 bool LLGL::RenderingFeatures::hasConstantBuffers = false

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

See also

BindFlags::ConstantBuffer

10.81.2.7 bool LLGL::RenderingFeatures::hasCubeArrayTextures = false

Specifies whether cube array textures are supported.

See also

TextureType::TextureCubeArray

10.81.2.8 bool LLGL::RenderingFeatures::hasCubeTextures = false

Specifies whether cube textures are supported.

See also

TextureType::TextureCube

10.81.2.9 bool LLGL::RenderingFeatures::hasDirectResourceBinding = false

Specifies whether the render system supports binding individual resources directly instead of a resource heap. This does not exclude support of resource heaps, because these are always supported.

See also

CommandBuffer::SetResource
CommandBuffer::ResetResourceSlots

10.81.2.10 bool LLGL::RenderingFeatures::hasGeometryShaders = false

Specifies whether geometry shaders are supported.

See also

ShaderType::Geometry

10.81.2.11 bool LLGL::RenderingFeatures::hasIndirectDrawing = false

Specifies whether indirect draw commands are supported.

See also

CommandBuffer::DrawIndirect
CommandBuffer::DrawIndexedIndirect

10.81.2.12 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)

10.81.2.13 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

10.81.2.14 bool LLGL::RenderingFeatures::hasMultiSampleTextures = false

Specifies whether multi-sample textures are supported.

See also

TextureType::Texture2DMS
TextureType::Texture2DMSArray

10.81.2.15 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)

10.81.2.16 bool LLGL::RenderingFeatures::hasPipelineStatistics = false

Specifies whether queries for pipeline statistics are supported.

See also

QueryType::PipelineStatistics
QueryPipelineStatistics

10.81.2.17 bool LLGL::RenderingFeatures::hasRenderCondition = false

Specifies whether queries for conditional rendering are supported.

See also

QueryHeapDescriptor::renderCondition
CommandBuffer:BeginRenderCondition

10.81.2.18 bool LLGL::RenderingFeatures::hasRenderTargets = false

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

10.81.2.19 bool LLGL::RenderingFeatures::hasSamplers = false

Specifies whether samplers are supported.

Todo Rename to `hasNativeSamplerStates` or remove entirely.

10.81.2.20 bool LLGL::RenderingFeatures::hasStorageBuffers = false

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

See also

BindFlags::Sampled
BindFlags::Storage

10.81.2.21 bool LLGL::RenderingFeatures::hasStreamOutputs = false

Specifies whether stream-output is supported.

See also

VertexShaderAttributes::outputAttribs
CommandBuffer::BeginStreamOutput
RenderingLimits::maxStreamOutputs

10.81.2.22 bool LLGL::RenderingFeatures::hasTessellationShaders = false

Specifies whether tessellation shaders are supported.

Remarks

This feature implies that `hasTessellatorStage` is true.

Note

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

See also

ShaderType::TessControl
ShaderType::TessEvaluation

10.81.2.23 bool LLGL::RenderingFeatures::hasTessellatorStage = false

Specifies whether tessellator stage is supported.

Remarks

The Metal backend supports a tessellator stage but no dedicated tessellation shaders. The tessellation control shader and the tessellation evaluation shader are usually defined by a compute kernel and a post-tessellation vertex function respectively.

See also

TessellationDescriptor.

10.81.2.24 bool LLGL::RenderingFeatures::hasTextureViews = false

Specifies whether texture views are supported.

Remarks

Texture views can share their image data with another texture resource in a different range and format.

See also

TextureViewDescriptor

10.81.2.25 bool LLGL::RenderingFeatures::hasTextureViewSwizzle = false

Specifies whether texture views can have swizzling (a.k.a. component mapping).

Remarks

This feature implies that `hasTextureViews` is true.

Note

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

See also

TextureViewDescriptor::swizzle

10.81.2.26 bool LLGL::RenderingFeatures::hasUniforms = false

Specifies whether individual shader uniforms are supported.

Note

Only supported with: OpenGL.

See also

CommandBuffer::SetUniform
CommandBuffer::SetUniforms

10.81.2.27 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

10.82 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 the render system supports. Upper limit is specified by `LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS`.
- 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.
- std::uint32_t maxStreamOutputs = 0

  Specifies the maximum number of simultaneous stream-output buffers.
- std::uint32_t maxTessFactor = 0

  Specifies the maximum tessellation factor.
- std::uint64_t minConstantBufferAlignment = 0

  Specifies the minimum alignment (in bytes) for Constant Buffer Views (CBV).
- std::uint64_t minSampledBufferAlignment = 0

  Specifies the minimum alignment (in bytes) for sampled buffers, aka. Shader Resource Views (SRV).
- std::uint64_t minStorageBufferAlignment = 0

  Specifies the minimum alignment (in bytes) for storage buffers, aka. Unordered Access Views (UAV).

10.82.1 Detailed Description

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

See also

RenderingCapabilities::limits

10.82.2 Member Data Documentation

10.82.2.1 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

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

Specifies the maximum size of each 1D texture.

See also

TextureDescriptor::extent

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

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

See also

TextureDescriptor::extent

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

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

See also

TextureDescriptor::extent

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

Specifies the maximum anisotropy texture filter.

See also

SamplerDescriptor::maxAnisotropy

10.82.2.6 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

10.82.2.7 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

10.82.2.8 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

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

Specifies the maximum work group size in a compute shader.

10.82.2.10 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

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

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

See also

TextureDescriptor::extent

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

Specifies the maximum number of patch control points.

See also

PrimitiveTopology::Patches1
PrimitiveTopology::Patches32

10.82.2.13 std::uint32_t LLGL::RenderingLimits::maxStreamOutputs = 0

Specifies the maximum number of simultaneous stream-output buffers.

Remarks

This must not be larger than `LLGL_MAX_NUM_SO_BUFFERS` which is 4.

See also

CommandBuffer::BeginStreamOutput
RenderingFeatures::hasStreamOutputs

10.82.2.14 std::uint32_t LLGL::RenderingLimits::maxTessFactor = 0

Specifies the maximum tessellation factor.

Remarks

Metal for example supports 64 on macOS and 16 on iOS.

See also

TessellationDescriptor::maxTessFactor

10.82.2.15 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

10.82.2.16 std::uint32_t LLGL::RenderingLimits::maxViewports = 0

Specifies the maximum number of viewports and scissor rectangles the render system supports. Upper limit is specified by `LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS`.

See also

CommandBuffer::SetViewports
CommandBuffer::SetScissors
GraphicsPipelineDescriptor::viewports
GraphicsPipelineDescriptor::scissors
RenderingFeatures::hasViewportArrays

10.82.2.17 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

10.82.2.18 std::uint64_t LLGL::RenderingLimits::minConstantBufferAlignment = 0

Specifies the minimum alignment (in bytes) for Constant Buffer Views (CBV).

See also

BufferViewDescriptor::offset
BufferViewDescriptor::size

10.82.2.19 std::uint64_t LLGL::RenderingLimits::minSampledBufferAlignment = 0

Specifies the minimum alignment (in bytes) for sampled buffers, aka. Shader Resource Views (SRV).

See also

BufferViewDescriptor::offset
BufferViewDescriptor::size

10.82.2.20 std::uint64_t LLGL::RenderingLimits::minStorageBufferAlignment = 0

Specifies the minimum alignment (in bytes) for storage buffers, aka. Unordered Access Views (UAV).

See also

BufferViewDescriptor::offset
BufferViewDescriptor::size

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

- RenderSystemFlags.h

10.83 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.
- bool timeRecordingEnabled = false

  Specifis whether the command buffer time recording is enabled or disabled. By default disabled.

10.83.1 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.

10.83.2 Member Function Documentation

10.83.2.1 void LLGL::RenderingProfiler::Accumulate (const FrameProfile & *profile*) [inline]

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

10.83.2.2 void LLGL::RenderingProfiler::NextProfile (FrameProfile * *outputProfile* = nullptr) [inline]

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.

10.83.3 Member Data Documentation

10.83.3.1 FrameProfile LLGL::RenderingProfiler::frameProfile

Current frame profile with all counter values.

10.83.3.2 bool LLGL::RenderingProfiler::timeRecordingEnabled = false

Specifis whether the command buffer time recording is enabled or disabled. By default disabled.

See also

FrameProfile::timeRecords

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

- RenderingProfiler.h

10.84 LLGL::RenderPass Class Reference

Render pass interface.

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

Inheritance diagram for LLGL::RenderPass:

Additional Inherited Members

10.84.1 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

10.85 LLGL::RenderPassDescriptor Struct Reference

Render pass descriptor structure.

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

Public Attributes

- AttachmentFormatDescriptor colorAttachments [LLGL_MAX_NUM_COLOR_ATTACHMENTS]

  Specifies the color attachments used within the render pass.
- AttachmentFormatDescriptor depthAttachment

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

  Specifies the stencil attachment used within the render pass.
- std::uint32_t samples = 1

  Specifies the number of samples for the respective render target attachment. By default 1.

10.85.1 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

10.85.2 Member Data Documentation

10.85.2.1 AttachmentFormatDescriptor LLGL::RenderPassDescriptor::colorAttachments[LLGL_MAX_NUM_COLOR_ATTACHMENTS]

Specifies the color attachments used within the render pass.

Remarks

Each attachment with a `format` field set to Format::Undefined is disabled. Any other attachment after the first disabled attachment is also considered disabled as attachments must be enabled in consecutive order. A swap-chain usually uses a BGRA format instead of an RGBA format.

See also

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

10.85.2.2 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.

10.85.2.3 std::uint32_t LLGL::RenderPassDescriptor::samples = 1

Specifies the number of samples for the respective render target attachment. By default 1.

Remarks

This must be greater than 0. If this is 1, multi-sampling is disabled. All attachments and the respective render target must have the same number of samples.

See also

TextureDescriptor::samples

10.85.2.4 AttachmentFormatDescriptor LLGL::RenderPassDescriptor::stencilAttachment

Specifies the stencil attachment used within the render pass.

Remarks

The depth attachment and stencil attachment usually share the same format (e.g. Format::D24UNormS8UInt). They are separated here to specify different load and store operations.

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

- RenderPassFlags.h

10.86 LLGL::RenderSystem Class Reference

Render system interface.

```
#include <RenderSystem.h>
```

Inheritance diagram for LLGL::RenderSystem:

Public Member Functions

- ~RenderSystem ()

  Releases the internal data.
- int GetRendererID () const

  Rendering API identification number.
- const char * 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 SwapChain * CreateSwapChain (const SwapChainDescriptor &swapChainDesc, const std::shared↩_ptr< Surface > &surface={})=0

  Creates a new swap-chain. At least one swap-chain is required to render into an output surface.
- virtual void Release (SwapChain &swapChain)=0

  Releases the specified swap-chain. After this call, the specified object must no longer be used.
- virtual CommandQueue * GetCommandQueue ()=0

  Returns the single instance of the command queue.
- virtual CommandBuffer * CreateCommandBuffer (const CommandBufferDescriptor &commandBuffer↩Desc={})=0

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

  Releases the specified command buffer. After this call, the specified object must no longer be used.
- virtual Buffer * CreateBuffer (const BufferDescriptor &bufferDesc, 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 &buffer, std::uint64_t offset, const void *data, std::uint64_t dataSize)=0

  Updates the data of the specified buffer.
- virtual void ReadBuffer (Buffer &buffer, std::uint64_t offset, void *data, std::uint64_t dataSize)=0

  Reads the data from the specified buffer.
- virtual void * MapBuffer (Buffer &buffer, const CPUAccess access)=0

  Maps the specified buffer from GPU to CPU memory space.
- virtual void * MapBuffer (Buffer &buffer, const CPUAccess access, std::uint64_t offset, std::uint64_t length)=0

  Maps the specified buffer range 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 (Texture &texture, const TextureRegion &textureRegion, const DstImageDescriptor &imageDesc)=0

Reads the image data from the specified texture.

- virtual Sampler * CreateSampler (const SamplerDescriptor &samplerDesc)=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 &resourceHeapDesc, const ArrayView< ResourceViewDescriptor > &initialResourceViews={})=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 std::uint32_t WriteResourceHeap (ResourceHeap &resourceHeap, std::uint32_t firstDescriptor, const ArrayView< ResourceViewDescriptor > &resourceViews)=0

  Writes new resource view descriptors into the specified resource heap.

- virtual RenderPass * CreateRenderPass (const RenderPassDescriptor &renderPassDesc)=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 &renderTargetDesc)=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 &shaderDesc)=0

  Creates a new and Shader object and compiles the specified source.

- virtual void Release (Shader &shader)=0

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

- virtual PipelineLayout * CreatePipelineLayout (const PipelineLayoutDescriptor &pipelineLayoutDesc)=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 PipelineState * CreatePipelineState (const Blob &serializedCache)=0

  Creates a new graphics or compute pipeline state object (PSO) from the specified cache.

- virtual PipelineState * CreatePipelineState (const GraphicsPipelineDescriptor &pipelineStateDesc, std↩::unique_ptr< Blob > *serializedCache=nullptr)=0

  Creates a new graphics pipeline state object (PSO).

- virtual PipelineState * CreatePipelineState (const ComputePipelineDescriptor &pipelineStateDesc, std↩::unique_ptr< Blob > *serializedCache=nullptr)=0

  Creates a new compute pipeline state object (PSO).

- virtual void Release (PipelineState &pipelineState)=0

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

- virtual QueryHeap * CreateQueryHeap (const QueryHeapDescriptor &queryHeapDesc)=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 RenderSystemPtr Load (const RenderSystemDescriptor &renderSystemDesc, RenderingProfiler *profiler=nullptr, RenderingDebugger *debugger=nullptr)

  Loads a new render system from the specified module.
- static void Unload (RenderSystemPtr &&renderSystem)

  Unloads the specified render system and the internal module.

Protected Member Functions

- RenderSystem ()

  Allocates the internal data.
- void SetRendererInfo (const RendererInfo &info)

  Sets the renderer information.
- void SetRenderingCaps (const RenderingCapabilities &caps)

  Sets the rendering capabilities.
- void AssertCreateBuffer (const BufferDescriptor &bufferDesc, 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 &shaderDesc)

  Validates the specified shader descriptor.
- void AssertCreateRenderTarget (const RenderTargetDescriptor &renderTargetDesc)

  Validates the specified render target 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).
- void CopyTextureImageData (const DstImageDescriptor &dstImageDesc, const Extent3D &extent, const Format format, const void *data, std::size_t rowStride=0)

  Copies the specified source data (i.e. `data`) to the destination image.

10.86.1 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);
```

10.86.2 Constructor & Destructor Documentation

10.86.2.1 LLGL::RenderSystem::~RenderSystem ( )

Releases the internal data.

10.86.2.2 LLGL::RenderSystem::RenderSystem ( ) [protected]

Allocates the internal data.

10.86.3 Member Function Documentation

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

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

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

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

10.86.3.3 void LLGL::RenderSystem::AssertCreateRenderTarget (const RenderTargetDescriptor & *renderTargetDesc*) [protected]

Validates the specified render target descriptor.

10.86.3.4 void LLGL::RenderSystem::AssertCreateShader (const ShaderDescriptor & *shaderDesc*) [protected]

Validates the specified shader descriptor.

10.86.3.5 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).

10.86.3.6 void LLGL::RenderSystem::CopyTextureImageData (const DstImageDescriptor & *dstImageDesc*, const Extent3D & *extent*, const Format *format*, const void * *data*, std::size_t *rowStride* = 0) [protected]

Copies the specified source data (i.e. `data`) to the destination image.

Remarks

This function also performs image conversion if there is a mismatch between source and destination format.

See also

ConvertImageBuffer

10.86.3.7 virtual Buffer* LLGL::RenderSystem::CreateBuffer (const BufferDescriptor & *bufferDesc*, const void * *initialData* = nullptr) [pure virtual]

Creates a new generic hardware buffer.

Parameters

in	*bufferDesc*	Specifies the 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

10.86.3.8 virtual BufferArray* LLGL::RenderSystem::CreateBufferArray (std::uint32_t *numBuffers*, Buffer *const * *bufferArray*) [pure virtual]

Creates a new buffer array.

Parameters

in	*numBuffers*	Specifies the number of buffers in the array. This must be greater than 0.
in	*bufferArray*	Pointer to an array of Buffer object pointers. This must not be null.

Remarks

All buffers within this array must have the same binding flags. The buffers inside this array must persist as long as this buffer array is used, and the individual buffers are still required to read and write its data from and to the GPU.

Exceptions

std::invalid_argument	If `numBuffers` is 0.
std::invalid_argument	If `bufferArray` is null.
std::invalid_argument	If any of the pointers in the array are null.
std::invalid_argument	If not all buffers have the same binding flags.

See also

BufferDescriptor::bindFlags

10.86.3.9 virtual CommandBuffer* LLGL::RenderSystem::CreateCommandBuffer (const CommandBufferDescriptor & *commandBufferDesc* = { }) [pure virtual]

Creates a new command buffer.

Parameters

in	*commandBufferDesc*	Specifies an optional command buffer descriptor.

Remarks

Each render system can create multiple command buffers, but especially the legacy graphics APIs such as OpenGL and Direct3D 11 don't provide a performance benefit with that feature.

10.86.3.10 virtual Fence* LLGL::RenderSystem::CreateFence ( ) [pure virtual]

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

See also

CommandBuffer::SubmitFence
CommandBuffer::WaitFence

10.86.3.11 virtual PipelineLayout* LLGL::RenderSystem::CreatePipelineLayout (const PipelineLayoutDescriptor & *pipelineLayoutDesc*) [pure virtual]

Creates a new and initialized pipeline layout object, if and only if the renderer supports pipeline layouts.

Parameters

in	*pipelineLayoutDesc*	Specifies the pipeline layout descriptor with all layout bindings.

Remarks

A pipeline layout is required in combination with a ResourceHeap to bind multiple resources at once. For modern graphics APIs (i.e. Direct3D 12 and Vulkan), this is only way to bind shader resources. For legacy graphics APIs (i.e. Direct3D 11 and OpenGL), shader resources can also be bound individually with the extended command buffer.

Returns

Pointer to the new PipelineLayout object or null if the renderer does not support pipeline layouts.

See also

CreateResourceHeap
PipelineLayoutDesc

10.86.3.12 virtual PipelineState* LLGL::RenderSystem::CreatePipelineState (const Blob & *serializedCache*) [pure virtual]

Creates a new graphics or compute pipeline state object (PSO) from the specified cache.

Parameters

in	*serializedCache*	Specifies the serialized cache that was created from the same pipeline state. This is usually created during a previous application run, stored to file, and restored on a later run.

Remarks

Here is an example how to use pipeline state caches:

```
// Try to read PSO cache file
if (auto myCache = LLGL::Blob::CreateFromFile("MyPSOCacheFile.bin"))
{
    // Create PSO from cache
    myPipelineState = myRenderer->CreatePipelineState(*myCache);
}
else
{
    // Setup initial pipeline state
    LLGL::ComputePipelineDescritpor myPipelineDesc;
    myPipelineDesc.pipelineLayout = myPipelineLayout;
    myPipelineDesc.computeShader  = myComputeShader;

    // Create new PSO
    std::unique_ptr<LLGL::Blob> myCache;
    myRenderer->CreatePipelineState(myPipelineDesc, &myCache);

    // Store PSO to file
    std::ofstream myCacheFile{ "MyPSOCacheFile.bin", std::ios::out | std::ios::binary };
    myCacheFile.write(
        reinterpret_cast<const char*>(myCache->GetData()),
        static_cast<std::streamsize>(myCache->GetSize())
    );
}
```

See also

CreatePipelineState(const GraphicsPipelineDescriptor&, std::unique_ptr<Blob>*)
CreatePipelineState(const ComputePipelineDescriptor&, std::unique_ptr<Blob>*)

10.86.3.13 virtual PipelineState* LLGL::RenderSystem::CreatePipelineState (const GraphicsPipelineDescriptor & *pipelineStateDesc*, std::unique_ptr< Blob > * *serializedCache* = nullptr) [pure virtual]

Creates a new graphics pipeline state object (PSO).

Parameters

in	*pipelineStateDesc*	Specifies the graphics PSO descriptor. This will describe the entire pipeline state, i.e. the blending-, rasterizer-, depth-, stencil- and shader states. The `vertexShader` member of the descriptor must never be null!
out	*serializedCache*	Optional pointer to a unique Blob instance. If this is not null, the renderer returns the pipeline state as serialized cache. This cache may be unique to the respective hardware and driver the application is running on. The behavior is undefined if this cache is used in a different software environment. It can be used to faster restore a pipeline state on next application run.

See also

GraphicsPipelineDescriptor
CreatePipelineState(const Blob&)

10.86.3.14 virtual PipelineState* LLGL::RenderSystem::CreatePipelineState (const ComputePipelineDescriptor & *pipelineStateDesc*, std::unique_ptr< Blob > * *serializedCache* = nullptr) [pure virtual]

Creates a new compute pipeline state object (PSO).

Parameters

in	*pipelineStateDesc*	Specifies the compute PSO descriptor. This will describe the entire pipeline state. The computeShader member of the descriptor must never be null!
out	*serializedCache*	Optional pointer to a unique Blob instance. If this is not null, the renderer returns the pipeline state as serialized cache. This cache may be unique to the respective hardware and driver the application is running on. The behavior is undefined if this cache is used in a different software environment. It can be used to faster restore a pipeline state on next application run.

See also

ComputePipelineDescriptor
CreatePipelineState(const Blob&)

10.86.3.15 virtual QueryHeap* LLGL::RenderSystem::CreateQueryHeap (const QueryHeapDescriptor & *queryHeapDesc*) [pure virtual]

Creates a new query heap.

10.86.3.16 virtual RenderPass* LLGL::RenderSystem::CreateRenderPass (const RenderPassDescriptor & *renderPassDesc*) [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

10.86.3.17 virtual RenderTarget* LLGL::RenderSystem::CreateRenderTarget (const RenderTargetDescriptor & *renderTargetDesc*) [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).

10.86.3.18 virtual ResourceHeap* LLGL::RenderSystem::CreateResourceHeap (const ResourceHeapDescriptor & *resourceHeapDesc*, const ArrayView< ResourceViewDescriptor > & *initialResourceViews* = { }) [pure virtual]

Creates a new resource heap.

Parameters

in	*resourceHeapDesc*	Specifies the descriptor for the resource heap. If the `numResourceViews` field is zero, the `initialResourceViews` parameter will determine the number of resources, it must *not* be empty and it **must** be a multiple of the number of bindings in the piepline layout.
in	*initialResourceViews*	Specifies an optional array of initial resource views. If this is non-null, the array pointed to must have enough elements to initialze the entire resource heap. Uninitialized resource views must be written with a call to WriteResourceHeap before the resource heap can be used in a command buffer.

Remarks

Resource 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::SetResourceHeap
WriteResourceHeap
ResourceHeapDescriptor::numResourceViews

10.86.3.19 virtual Sampler* LLGL::RenderSystem::CreateSampler (const SamplerDescriptor & *samplerDesc*) [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

10.86.3.20 virtual Shader* LLGL::RenderSystem::CreateShader (const ShaderDescriptor & *shaderDesc*) [pure virtual]

Creates a new and Shader object and compiles the specified source.

Remarks

To check whether the compilation was successful or not, use the `HasErrors` and `GetReport` functions of the Shader interface.

See also

Shader::HasErrors
Shader::GetReport
ShaderDescriptor
ShaderDescFromFile

10.86.3.21 virtual SwapChain* LLGL::RenderSystem::CreateSwapChain (const SwapChainDescriptor & *swapChainDesc,* const std::shared_ptr< Surface > & *surface* = { }) [pure virtual]

Creates a new swap-chain. At least one swap-chain is required to render into an output surface.

Parameters

in	*swapChainDesc*	Specifies the swap-chain descriptor, which contains resolution, bit depth, multi-sampling settings etc.
in	*surface*	Optional shared pointer to a surface for the swap-chain. If this is null, the swap-chain will create its own platform specific surface, which can be accessed by SwapChain::GetSurface. The default surface on desktop platforms (i.e. Window interface) is not shown automatically, i.e. the Window::Show function has to be invoked to show the surface.

See also

SwapChain::GetSurface
Window::Show

10.86.3.22 virtual Texture* LLGL::RenderSystem::CreateTexture (const TextureDescriptor & *textureDesc,* const SrcImageDescriptor * *imageDesc* = `nullptr`) [pure virtual]

Creates a new texture.

Parameters

in	*textureDesc*	Specifies the texture descriptor.
in	*imageDesc*	Optional pointer to the image data descriptor. If this is null, the texture will be initialized with the currently configured default image color (if this feature is enabled). If this is non-null, it is used to initialize the texture data. This parameter will be ignored if the texture type is a multi-sampled texture (i.e. TextureType::Texture2DMS or TextureType::Texture2DMSArray).

See also

WriteTexture

10.86.3.23 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 `{ "Direct3D12", "Direct3D11", "OpenGL" }`, but on MacOS it might be `{ "Metal", "OpenGL" }`.

10.86.3.24 virtual CommandQueue* LLGL::RenderSystem::GetCommandQueue ( ) [pure virtual]

Returns the single instance of the command queue.

10.86.3.25 const char* LLGL::RenderSystem::GetName ( ) const

Returns the name of this render system.

10.86.3.26 int LLGL::RenderSystem::GetRendererID ( ) const

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

10.86.3.27 const RendererInfo& LLGL::RenderSystem::GetRendererInfo ( ) const

Returns basic renderer information.

Remarks

The validity of these information is only guaranteed if this function is called after a valid swap-chain has been created. Otherwise the behavior is undefined!

10.86.3.28 const RenderingCapabilities& LLGL::RenderSystem::GetRenderingCaps ( ) const

Returns the rendering capabilities.

Remarks

The validity of these information is only guaranteed if this function is called after a valid swap-chain has been created. Otherwise the behavior is undefined!

10.86.3.29 static RenderSystemPtr 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

10.86.3.30 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. Depending on the CPU access type (see `access` parameter), this buffer must have been created with the corresponding CPU access flag, i.e. CPUAccessFlags::Read and/or CPUAccessFlags::Write.
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 in CPU memory space or null if the operation failed.

Remarks

Memory that is written back from CPU to GPU becomes visible in the GPU after a corresponding UnmapBuffer operation.

See also

UnmapBuffer

10.86.3.31 virtual void* LLGL::RenderSystem::MapBuffer (Buffer & *buffer*, const CPUAccess *access*, std::uint64_t *offset*, std::uint64_t *length*) [pure virtual]

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

Parameters

in	*buffer*	Specifies the buffer which is to be mapped. Depending on the CPU access type (see `access` parameter), this buffer must have been created with the corresponding CPU access flag, i.e. CPUAccessFlags::Read and/or CPUAccessFlags::Write.
in	*access*	Specifies the CPU buffer access requirement, i.e. if the CPU can read and/or write the mapped memory.
in	*offset*	Specifies the memory offset (in bytes) from the GPU buffer.
in	*length*	Specifies the length of the memory block (in bytes) that is to be mapped.

Returns

Raw pointer to the mapped memory block in CPU memory space or null if the operation failed.

Remarks

Memory that is written back from CPU to GPU becomes visible in the GPU after a corresponding UnmapBuffer operation.

See also

UnmapBuffer

10.86.3.32 virtual void LLGL::RenderSystem::ReadBuffer (Buffer & *buffer*, std::uint64_t *offset*, void * *data*, std::uint64_t *dataSize*) [pure virtual]

Reads the data from the specified buffer.

Parameters

in	*buffer*	Specifies the buffer which is to be read.
in	*offset*	Specifies the offset (in bytes) at which the buffer is to be read.
out	*data*	Raw pointer to a memory block in CPU memory space where the data will be written to.
in	*dataSize*	Specifies the size (in bytes) of the data block given by the `data` parameter.

See also

WriteBuffer

10.86.3.33 virtual void LLGL::RenderSystem::ReadTexture (Texture & *texture*, const TextureRegion & *textureRegion*, const DstImageDescriptor & *imageDesc*) [pure virtual]

Reads the image data from the specified texture.

Parameters

in	*texture*	Specifies the texture object to read from.
in	*textureRegion*	Specifies the region where the texture data is to be read.
out	*imageDesc*	Specifies the destination image descriptor to write the texture data to.

Remarks

The required size for a successful texture read operation depends on the image format, data type, and texture size. The Texture::GetDesc or Texture::GetMipExtent functions can be used to determine the texture dimensions.

```
// Query texture size attribute
auto myTextureExtent = myTexture->GetMipExtent(0);

// Allocate image buffer with elements in all dimensions
std::vector<LLGL::ColorRGBAub> myImage(myTextureExtent.width * myTextureExtent.height * myTextureExtent.
      depth);

// Initialize destination image descriptor
const DstImageDescriptor myImageDesc {
    LLGL::ImageFormat::RGBA,                    // RGBA image format, since we used
       LLGL::ColorRGBAub
    LLGL::DataType::UInt8,                      // 8-bit unsigned integral data type:
       <std::uint8_t> or <unsigned char>
    myImage.data(),                             // Output image buffer
    myImage.size() * sizeof(LLGL::ColorRGBAub)  // Image buffer size: number of color
       elements and size of each color element
};

// Read texture data from first MIP-map level (index 0)
myRenderSystem->ReadTexture(*myTexture, 0, myImageDesc);
```

Note

The behavior is undefined if `imageDesc.data` points to an invalid buffer, or `imageDesc.data` points to a buffer that is smaller than specified by `imageDesc.dataSize`, or `imageDesc.dataSize` is less than the required size.

Exceptions

std::invalid_argument	If `imageDesc.data` is null.

See also

Texture::GetDesc
Texture::GetMipExtent

10.86.3.34 virtual void LLGL::RenderSystem::Release (SwapChain & *swapChain*) [pure virtual]

Releases the specified swap-chain. After this call, the specified object must no longer be used.

See also

CreateSwapChain

10.86.3.35 virtual void LLGL::RenderSystem::Release (CommandBuffer & *commandBuffer*) [pure virtual]

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

See also

CreateCommandBuffer

10.86.3.36 virtual void LLGL::RenderSystem::Release (Buffer & *buffer*) [pure virtual]

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

10.86.3.37 virtual void LLGL::RenderSystem::Release (BufferArray & *bufferArray*) [pure virtual]

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

10.86.3.38 virtual void LLGL::RenderSystem::Release (Texture & *texture*) [pure virtual]

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

10.86.3.39 virtual void LLGL::RenderSystem::Release (Sampler & *sampler*) [pure virtual]

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

10.86.3.40 virtual void LLGL::RenderSystem::Release (ResourceHeap & *resourceHeap*) [pure virtual]

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

10.86.3.41 virtual void LLGL::RenderSystem::Release (RenderPass & *renderPass*) [pure virtual]

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

10.86.3.42 virtual void LLGL::RenderSystem::Release (RenderTarget & *renderTarget*) [pure virtual]

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

10.86.3.43 virtual void LLGL::RenderSystem::Release (Shader & *shader*) [pure virtual]

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

10.86.3.44 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.

10.86.3.45 virtual void LLGL::RenderSystem::Release (PipelineState & *pipelineState*) [pure virtual]

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

10.86.3.46 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.

10.86.3.47 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.

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

Sets the renderer information.

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

Sets the rendering capabilities.

10.86.3.50 static void LLGL::RenderSystem::Unload (RenderSystemPtr && *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!

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

Unmaps the specified buffer.

Remarks

This must be called on a buffer that was previously mapped into CPU memory space. The following example illustrates how to map and unmap a buffer from GPU into CPU memory sapce:

```
if (void* data = myRenderer->MapBuffer(*myBuffer, LLGL::CPUAccess::Write))
{
    // Write to 'data' ...
    myRenderer->UnmapBuffer(*myBuffer);
}
```

See also

MapBuffer

10.86.3.52 virtual void LLGL::RenderSystem::WriteBuffer (Buffer & *buffer*, std::uint64_t *offset*, const void * *data*, std::uint64_t *dataSize*) [pure virtual]

Updates the data of the specified buffer.

Parameters

in	*buffer*	Specifies the destination buffer whose data is to be updated.
in	*offset*	Specifies the offset (in bytes) at which the buffer is to be updated. This offset plus the data block size (i.e. `offset + dataSize`) must be less than or equal to the size of the buffer.
in	*data*	Raw pointer to the data with which the buffer is to be updated. This must not be null!
in	*dataSize*	Specifies the size (in bytes) of the data block which is to be updated. This must be less then or equal to the size of the buffer.

Remarks

To update a small buffer (maximum of 65536 bytes) during encoding a command buffer, use CommandBuffer::UpdateBuffer.

See also

ReadBuffer

10.86.3.53 virtual std::uint32_t LLGL::RenderSystem::WriteResourceHeap (ResourceHeap & *resourceHeap*, std::uint32_t *firstDescriptor*, const ArrayView< ResourceViewDescriptor > & *resourceViews*) [pure virtual]

Writes new resource view descriptors into the specified resource heap.

Parameters

in	*resourceHeap*	Specifies the resource heap that is to be updated.
in	*firstDescriptor*	Zero-based index to the first descriptor that is to be updated. This must be less than the number of bindings in the resource heap's pipeline layout (PipelineLayout::GetNumBindings) times the number of descriptor sets in this resource heap (ResourceHeap::GetNumDescriptorSets).
in	*resourceViews*	Array of resource view descriptors.

Remarks

The type of a resource view, i.e. whether it's a buffer, texture, or sampler, must not be changed with this function.

Returns

Number of resource views that have been updated by this call. Any resource view descriptor with a `resource` field that is null will be ignored silently.

See also

ResourceHeap::GetNumDescriptorSets
PipelineLayout::GetNumBindings

10.86.3.54 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 region where the texture is to be updated. The field TextureRegion::numMipLevels **must** be 1.
in	*imageDesc*	Specifies the image data descriptor. Its `data` member must not be null!

Remarks

This function can only be used for non-multi-sample textures, i.e. from types other than TextureType::Texture2DMS and TextureType::Texture2DMSArray.

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

- RenderSystem.h

10.87 LLGL::RenderSystemChild Class Reference

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

```
#include <RenderSystemChild.h>
```

Inheritance diagram for LLGL::RenderSystemChild:

Public Member Functions

- virtual void SetName (const char *name)

  Sets the name of this class instance for debugging purposes.

Additional Inherited Members

10.87.1 Detailed Description

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

10.87.2 Member Function Documentation

10.87.2.1 virtual void LLGL::RenderSystemChild::SetName (const char * *name*) [virtual]

Sets the name of this class instance for debugging purposes.

Parameters

in	*name*	Pointer to a null terminated string that specifies the new name of this instance. Specifying a null pointer effectively removes the name from the object. The implementation of this function may alter the actual name depending on how many internal objects need to be labeled.

Remarks

This is used for debugging purposes only and the implementation is undefined, i.e. if the respective render system does not support debug labels this function call will be ignored silently.
This function is especially useful in conjunction with diagnistoc tools (such as `RenderDoc`) to better identify objects in the event history.
The default implementation has no effect.

See also

CommandBuffer::PushDebugGroup
CommandBuffer::PopDebugGroup
RenderSystemDescriptor::debugCallback

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

- RenderSystemChild.h

10.88 LLGL::RenderSystemDeleter Class Reference

Delegate to delete an instance of the RenderSystem interface.

```
#include <RenderSystem.h>
```

Public Types

- typedef void(∗ RenderSystemDeleterFuncPtr) (void ∗)

Public Member Functions

- RenderSystemDeleter () noexcept=default
- RenderSystemDeleter (const RenderSystemDeleter &) noexcept=default
- RenderSystemDeleter & operator= (const RenderSystemDeleter &) noexcept=default
- RenderSystemDeleter (RenderSystemDeleterFuncPtr deleterFuncPtr)

  Constructs the deleter with the actual deleter function pointer.
- void operator() (RenderSystem ∗ptr) const

10.88.1 Detailed Description

Delegate to delete an instance of the RenderSystem interface.

Remarks

This deleter keeps a function pointer to the actual deleter from the renderer module. If no function pointer is provided, the deleter uses the C++ `delete` operator by default.

See also

RenderSystem::Load
RenderSystem::Unload

10.88.2 Member Typedef Documentation

10.88.2.1 typedef void(* LLGL::RenderSystemDeleter::RenderSystemDeleterFuncPtr) (void *)

10.88.3 Constructor & Destructor Documentation

10.88.3.1 LLGL::RenderSystemDeleter::RenderSystemDeleter ( ) [default], [noexcept]

10.88.3.2 LLGL::RenderSystemDeleter::RenderSystemDeleter (const RenderSystemDeleter &) [default], [noexcept]

10.88.3.3 LLGL::RenderSystemDeleter::RenderSystemDeleter (RenderSystemDeleterFuncPtr *deleterFuncPtr*) [inline]

Constructs the deleter with the actual deleter function pointer.

10.88.4 Member Function Documentation

10.88.4.1 void LLGL::RenderSystemDeleter::operator() (RenderSystem * *ptr*) const [inline]

Deletes the specified render system using the function pointer this deleter was initialized with.

Remarks

If no function pointer was provided, the deleter uses the C++ `delete` operator by default.

10.88.4.2 RenderSystemDeleter& LLGL::RenderSystemDeleter::operator= (const RenderSystemDeleter &) [default], [noexcept]

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

- RenderSystem.h

10.89 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 from an std::string.
- RenderSystemDescriptor (const char *moduleName)

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

Public Attributes

- std::string moduleName

  Specifies the name from which the new render system is to be loaded.
- DebugCallback debugCallback

  Debuging callback function object.
- const void * rendererConfig = nullptr

  Optional raw pointer to a renderer specific configuration structure.
- std::size_t rendererConfigSize = 0

  Specifies the size (in bytes) of the structure where the `rendererConfig` member points to (use `sizeof` with the respective structure). By default 0.

10.89.1 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

10.89.2 Constructor & Destructor Documentation

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

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

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

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

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

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

10.89.3 Member Function Documentation

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

10.89.4 Member Data Documentation

10.89.4.1 DebugCallback LLGL::RenderSystemDescriptor::debugCallback

Debuging callback function object.

10.89.4.2 std::string LLGL::RenderSystemDescriptor::moduleName

Specifies the name from which the new render system is to be loaded.

Remarks

This denotes a shared library (*.dll-files on Windows, *.so-files on Unix systems). If compiled in debug mode, the postfix "D" is appended to the module name. Moreover, the platform dependent file extension is always added automatically as well as the prefix "LLGL_", i.e. a module name "OpenGL" will be translated to "LLGL_OpenGLD.dll", if compiled on Windows in Debug mode. If LLGL was built with the `LLGL_BUILD_STATIC_LIB` option, this member is ignored.

10.89.4.3 const void* LLGL::RenderSystemDescriptor::rendererConfig = nullptr

Optional raw pointer to a renderer specific configuration structure.

Remarks

This can be used to pass some refinement configurations to the render system when the module is loaded. Example usage (for Vulkan renderer):

```
// Initialize Vulkan specific configurations (e.g. always allocate at least 1GB of VRAM for each device
      memory chunk).
LLGL::RendererConfigurationVulkan config;
config.minDeviceMemoryAllocationSize = 1024*1024*1024;

// Initialize render system descriptor
LLGL::RenderSystemDescriptor rendererDesc;
rendererDesc.moduleName         = "Vulkan";
rendererDesc.rendererConfig     = &config;
rendererDesc.rendererConfigSize = sizeof(config);

// Load Vulkan render system
auto renderer = LLGL::RenderSystem::Load(rendererDesc);
```

See also

rendererConfigSize
RendererConfigurationVulkan
RendererConfigurationOpenGL
RendererConfigurationOpenGLES3

10.89.4.4 std::size_t LLGL::RenderSystemDescriptor::rendererConfigSize = 0

Specifies the size (in bytes) of the structure where the `rendererConfig` member points to (use `sizeof` with the respective structure). By default 0.

Remarks

If `rendererConfig` is null then this member is ignored.

See also

rendererConfig

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

- RenderSystemFlags.h

10.90 LLGL::RenderTarget Class Reference

Render target interface.

`#include <RenderTarget.h>`

Inheritance diagram for LLGL::RenderTarget:

Public Member Functions

- virtual Extent2D GetResolution () const =0

  Returns the render target resolution.
- virtual std::uint32_t GetSamples () const =0

  Returns the number of samples this render target was created with.
- 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.

10.90.1 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 SwapChain, a special case of render targets used to present the result on the screen.

See also

RenderSystem::CreateRenderTarget
CommandBuffer::BeginRenderPass
SwapChain

10.90.2 Member Function Documentation

10.90.2.1 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 swap-chain, this will always be 1.

See also

SwapChain::GetColorFormat

Implemented in LLGL::SwapChain.

10.90.2.2 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

10.90.2.3 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 swap-chain descriptor.

See also

SwapChain::ResizeBuffers
RenderTargetDescriptor::resolution
SwapChainDescriptor::resolution

Implemented in LLGL::SwapChain.

10.90.2.4 virtual std::uint32_t LLGL::RenderTarget::GetSamples ( ) const [pure virtual]

Returns the number of samples this render target was created with.

Remarks

If a certain number of samples is not supported by the rendering API, LLGL will reduce the amount of samples. This function returns the actual number of samples the hardware object was created with.

See also

RenderTargetDescriptor::samples
SwapChainDescriptor::samples

10.90.2.5 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

SwapChain::GetDepthStencilFormat

Implemented in LLGL::SwapChain.

10.90.2.6 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

SwapChain::GetDepthStencilFormat

Implemented in LLGL::SwapChain.

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

Applies the resolution of the texture MIP level.

See also

Texture::GetMipExtent
ValidateResolution

10.90.2.8 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

10.91 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.
- std::uint32_t samples = 1

  Number of samples for the render targets. By default 1.
- 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.

10.91.1 Detailed Description

Render target descriptor structure.

Remarks

Here is a small example of a render target descriptor with a color attachmnet and an anonymous depth attachment (i.e. without a texture reference, which is only allowed for depth/stencil attachments):

```
LLGL::RenderTargetDescriptor myRenderTargetDesc;

auto myRenderTargetSize = myColorTexture->GetMipExtent(0);
myRenderTargetDesc.resolution = { myRenderTargetSize.width, myRenderTargetSize.height };

myRenderTargetDesc.attachments = {
    LLGL::AttachmentDescriptor{
      LLGL::AttachmentType::Color, myColorTexture },
    LLGL::AttachmentDescriptor{
      LLGL::AttachmentType::Depth },
};

auto myRenderTarget = myRenderer->CreateRenderTarget(myRenderTargetDesc);
```

See also

RenderSystem::CreateRenderTarget

10.91.2 Member Data Documentation

10.91.2.1 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.

10.91.2.2 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.

Todo Remove this attribute and support custom resolve attachments instead.

10.91.2.3 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

10.91.2.4 Extent2D LLGL::RenderTargetDescriptor::resolution

Specifies the resolution of the render targets.

Remarks

All attachments with a reference to a texture must have the same resolution, i.e. the specified array layer and MIP-map level must have the same extent.

See also

Texture::GetMipExtent

10.91.2.5 std::uint32_t LLGL::RenderTargetDescriptor::samples = 1

Number of samples for the render targets. By default 1.

Remarks

If the specified number of samples is not supported, LLGL will silently reduce it. The actual number of samples can be queried by the `GetSamples` function of the RenderTarget interface.

See also

RenderTarget::GetSamples

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

- RenderTargetFlags.h

10.92 LLGL::Report Class Reference

Error and warning report interface.

```
#include <Report.h>
```

Inheritance diagram for LLGL::Report:

Public Member Functions

- virtual const char ∗ GetText () const =0

  Returns a NUL-terminated string of the report text. This must never be null.
- virtual bool HasErrors () const =0

  Returns true if this report contains error messages.

Additional Inherited Members

10.92.1 Detailed Description

Error and warning report interface.

See also

PipelineState::GetReport

10.92.2 Member Function Documentation

10.92.2.1 virtual const char* LLGL::Report::GetText ( ) const [pure virtual]

Returns a NUL-terminated string of the report text. This must never be null.

10.92.2.2 virtual bool LLGL::Report::HasErrors ( ) const [pure virtual]

Returns true if this report contains error messages.

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

- Report.h

10.93 LLGL::ResizeBuffersFlags Struct Reference

Swap-chain resize buffers flags.

```
#include <SwapChainFlags.h>
```

Public Types

- enum { AdaptSurface = (1 << 0), FullscreenMode = (1 << 1), WindowedMode = (1 << 2) }

10.93.1 Detailed Description

Swap-chain resize buffers flags.

See also

SwapChain::ResizeBuffers

10.93.2 Member Enumeration Documentation

10.93.2.1 anonymous enum

Enumerator

AdaptSurface Adapts the swap-chain's surface for the new resolution.

See also

Surface::AdaptForVideoMode

FullscreenMode Puts the swap-chain into fullscreen mode.

Remarks

This implies AdaptSurface but cannot be used in combination with the WindowedMode flag.

See also

WindowedMode
SwapChain::SwitchFullscreen

WindowedMode Puts the swap-chain into windowed mode.

Remarks

This implies AdaptSurface but cannot be used in combination with the FullscreenMode flag.

See also

FullscreenMode
SwapChain::SwitchFullscreen

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

- SwapChainFlags.h

10.94 LLGL::Resource Class Reference

Base class for all hardware resource interfaces.

```
#include <Resource.h>
```

Inheritance diagram for LLGL::Resource:

Public Member Functions

- virtual ResourceType GetResourceType () const =0

  Returns the type of this resource object.

Additional Inherited Members

10.94.1 Detailed Description

Base class for all hardware resource interfaces.

See also

Buffer
Texture
Sampler

10.94.2 Member Function Documentation

10.94.2.1 virtual ResourceType LLGL::Resource::GetResourceType ( ) const [pure virtual]

Returns the type of this resource object.

Remarks

This is queried by a virtual function call, so the resource type does not need to be stored per instance.

See also

ResourceType

Todo (Maybe) replace by IsInstanceOf

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

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

- Resource.h

10.95 LLGL::ResourceHeap Class Reference

Resource heap interface.

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

Inheritance diagram for LLGL::ResourceHeap:

Public Member Functions

- virtual std::uint32_t GetNumDescriptorSets () const =0

  Returns the number of descriptor sets in this heap.

Additional Inherited Members

10.95.1 Detailed Description

Resource heap 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::CreateResourceHeap
CommandBuffer::SetResourceHeap

10.95.2 Member Function Documentation

10.95.2.1 virtual std::uint32_t LLGL::ResourceHeap::GetNumDescriptorSets ( ) const [pure virtual]

Returns the number of descriptor sets in this heap.

Remarks

This is determined by the number of resources in the heap divided by the number of bindings in the pipeline layout. The total number of resources in the heap can therefore be determined by the following code:

```
myResourceHeap->GetNumDescriptorSets() * myPipelineLayout->GetNumBindings();
```

See also

PipelineLayout::GetNumBindings
CommandBuffer::SetResourceHeap

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

- ResourceHeap.h

10.96 LLGL::ResourceHeapDescriptor Struct Reference

Resource heap descriptor structure.

```
#include <ResourceHeapFlags.h>
```

Public Member Functions

- ResourceHeapDescriptor ()=default
- ResourceHeapDescriptor (PipelineLayout *pipelineLayout, std::uint32_t numResourceViews=0, long barrierFlags=0)

  Initializes the resource heap descriptor with the specified pipeline layout and optional secondary parameters.

Public Attributes

- PipelineLayout * pipelineLayout = nullptr

  Reference to the pipeline layout. This must not be null, when a resource heap is created.
- std::uint32_t numResourceViews = 0

  Specifies the number of resource views.
- long barrierFlags = 0

  Specifies optional resource barrier flags. By default 0.

10.96.1 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

10.96.2 Constructor & Destructor Documentation

10.96.2.1 LLGL::ResourceHeapDescriptor::ResourceHeapDescriptor ( ) [default]

10.96.2.2 LLGL::ResourceHeapDescriptor::ResourceHeapDescriptor (PipelineLayout * *pipelineLayout*, std::uint32_t *numResourceViews* = 0, long *barrierFlags* = 0) [inline]

Initializes the resource heap descriptor with the specified pipeline layout and optional secondary parameters.

10.96.3 Member Data Documentation

10.96.3.1 long LLGL::ResourceHeapDescriptor::barrierFlags = 0

Specifies optional resource barrier flags. By default 0.

Remarks

If the barrier flags are non-zero, they will be applied before any resource are bound to the graphics/compute pipeline. This should be used when a resource is bound to the pipeline that was previously written to.

See also

BarrierFlags

10.96.3.2 std::uint32_t LLGL::ResourceHeapDescriptor::numResourceViews = 0

Specifies the number of resource views.

Remarks

If the number of resource views is non-zero, it **must** a multiple of the bindings in the pipeline layout.
If the number of resource views is zero, the number will be determined by the initial resource views and they must *not* be empty and they **must** be a multiple of the bindings in the pipeline layout.

See also

PipelineLayoutDescriptor::bindings
RenderSystem::CreateResourceHeap

10.96.3.3 PipelineLayout* LLGL::ResourceHeapDescriptor::pipelineLayout = nullptr

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

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

- ResourceHeapFlags.h

10.97 LLGL::ResourceViewDescriptor Struct Reference

Resource view descriptor structure.

```
#include <ResourceHeapFlags.h>
```

Public Member Functions

- ResourceViewDescriptor ()=default

  Default constructor to initialize the resource view with a null pointer.
- ResourceViewDescriptor (Resource *resource)

  Constructor to initialize the descriptor with a resource. The resource view will access the entire resource.
- ResourceViewDescriptor (Texture *texture, const TextureViewDescriptor &subresourceDesc)

  Constructor to initialize a descriptor with a texture subresource view.
- ResourceViewDescriptor (Buffer *buffer, const BufferViewDescriptor &subresourceDesc)

  Constructor to initialize a descriptor with a buffer subresource view.

Public Attributes

- Resource * resource = nullptr

  Pointer to the hardware resource.
- TextureViewDescriptor textureView

  Optional texture view descriptor.
- BufferViewDescriptor bufferView

  Optional buffer view descriptor.
- std::uint32_t initialCount = 0

  Initial counter value for an `AppendStructuredBuffer` and `ConsumeStructuredBuffer` in HLSL.

10.97.1 Detailed Description

Resource view descriptor structure.

See also

ResourceHeapDescriptor::resourceViews
RenderSystem::WriteResourceHeap

10.97.2 Constructor & Destructor Documentation

10.97.2.1 LLGL::ResourceViewDescriptor::ResourceViewDescriptor ( ) [default]

Default constructor to initialize the resource view with a null pointer.

10.97.2.2 LLGL::ResourceViewDescriptor::ResourceViewDescriptor (Resource * *resource*) [inline]

Constructor to initialize the descriptor with a resource. The resource view will access the entire resource.

10.97.2.3 LLGL::ResourceViewDescriptor::ResourceViewDescriptor (Texture * *texture*, const TextureViewDescriptor & *subresourceDesc*) [inline]

Constructor to initialize a descriptor with a texture subresource view.

10.97.2.4 LLGL::ResourceViewDescriptor::ResourceViewDescriptor (Buffer * *buffer*, const BufferViewDescriptor & *subresourceDesc*) [inline]

Constructor to initialize a descriptor with a buffer subresource view.

10.97.3 Member Data Documentation

10.97.3.1 BufferViewDescriptor LLGL::ResourceViewDescriptor::bufferView

Optional buffer view descriptor.

Remarks

Can be used to declare a subresource view of a buffer resource.
This attribute is ignored if *all* of the following sub members have the respective value listed below:

- `bufferView.format` is Format::Undefined;
- `bufferView.offset` is 0.
- `bufferView.size` is `Constants::wholeSize`.

10.97.3.2 std::uint32_t LLGL::ResourceViewDescriptor::initialCount = 0

Initial counter value for an `AppendStructuredBuffer` and `ConsumeStructuredBuffer` in HLSL.

Remarks

This is only used for HLSL (D3D) to initialize the hidden counter of appendable and consumable unordered access views (UAV). This will be used in the D3D backends for buffer resources that have been created with either the MiscFlags::Append or MiscFlags::Counter flags.

Note

Only supported with: Direct3D 11, Direct3D 12.

See also

MiscFlags::Append
MiscFlags::Counter

10.97.3.3 Resource* LLGL::ResourceViewDescriptor::resource = nullptr

Pointer to the hardware resource.

See also

This *can* be null when passed to a ResourceHeap to skip over resources that are intended to be unchanged. This way a single `WriteResourceHeap` invocation can be used to write a partial range of resource views.

10.97.3.4 TextureViewDescriptor LLGL::ResourceViewDescriptor::textureView

Optional texture view descriptor.

Remarks

Can be used to declare a subresource view of a texture resource.
This attribute is ignored if *one* of the following sub members has the respective value listed below:

- `textureView.format` is Format::Undefined
- `textureView.subresource.numMipLevels` is 0
- `textureView.subresource.numArrayLayers` is 0

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

- ResourceHeapFlags.h

10.98 LLGL::Sampler Class Reference

Sampler interface.

`#include <Sampler.h>`

Inheritance diagram for LLGL::Sampler:

Public Member Functions

- ResourceType GetResourceType () const override final

  Returns ResourceType::Sampler.

Protected Member Functions

- Sampler ()=default

10.98.1 Detailed Description

Sampler interface.

See also

RenderSystem::CreateSampler

10.98.2 Constructor & Destructor Documentation

10.98.2.1 LLGL::Sampler::Sampler ( ) [protected], [default]

10.98.3 Member Function Documentation

10.98.3.1 ResourceType LLGL::Sampler::GetResourceType ( ) const [final], [override], [virtual]

Returns ResourceType::Sampler.

Implements LLGL::Resource.

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

- Sampler.h

10.99 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).

10.99.1 Detailed Description

Texture sampler descriptor structure.

See also

RenderSystem::CreateSampler

10.99.2 Member Data Documentation

10.99.2.1 SamplerAddressMode LLGL::SamplerDescriptor::addressModeU = SamplerAddressMode::Repeat

Sampler address mode in U direction (also X axis). By default SamplerAddressMode::Repeat.

10.99.2.2 SamplerAddressMode LLGL::SamplerDescriptor::addressModeV = SamplerAddressMode::Repeat

Sampler address mode in V direction (also Y axis). By default SamplerAddressMode::Repeat.

10.99.2.3 SamplerAddressMode LLGL::SamplerDescriptor::addressModeW = SamplerAddressMode::Repeat

Sampler address mode in W direction (also Z axis). By default SamplerAddressMode::Repeat.

10.99.2.4 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}`

10.99.2.5 bool LLGL::SamplerDescriptor::compareEnabled = false

Specifies whether the compare operation for depth textures is to be used or not. By default false.

10.99.2.6 CompareOp LLGL::SamplerDescriptor::compareOp = CompareOp::Less

Compare operation for depth textures. By default CompareOp::Less.

10.99.2.7 SamplerFilter LLGL::SamplerDescriptor::magFilter = SamplerFilter::Linear

Magnification filter. By default SamplerFilter::Linear.

10.99.2.8 std::uint32_t LLGL::SamplerDescriptor::maxAnisotropy = 1

Maximal anisotropy in the range [1, 16].

Note

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

10.99.2.9 float LLGL::SamplerDescriptor::maxLOD = 1000.0f

Upper end of the MIP-map range. Must be greater than or equal to `minLOD`. By default 1000.

10.99.2.10 SamplerFilter LLGL::SamplerDescriptor::minFilter = SamplerFilter::Linear

Minification filter. By default SamplerFilter::Linear.

10.99.2.11 float LLGL::SamplerDescriptor::minLOD = 0.0f

Lower end of the MIP-map range. By default 0.

10.99.2.12 SamplerFilter LLGL::SamplerDescriptor::mipMapFilter = SamplerFilter::Linear

MIP-mapping filter. By default SamplerFilter::Linear.

10.99.2.13 float LLGL::SamplerDescriptor::mipMapLODBias = 0.0f

MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.

Remarks

For Metal and OpenGLES, the LOD bias can only be specified within the shader code.

Note

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

10.99.2.14 bool LLGL::SamplerDescriptor::mipMapping = true

Specifies whether MIP-maps are used or not. By default true.

Remarks

The number of MIP-maps a texture has is specified by the TextureDescriptor::mipLevels attribute.

See also

TextureDescriptor::mipLevels

Todo Rename to `mipMapEnabled`.

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

- SamplerFlags.h

10.100 LLGL::Scissor Struct Reference

Scissor dimensions.

```
#include <PipelineStateFlags.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.

10.100.1 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

10.100.2 Constructor & Destructor Documentation

10.100.2.1 LLGL::Scissor::Scissor ( ) [default]

10.100.2.2 LLGL::Scissor::Scissor (const Scissor &) [default]

10.100.2.3 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.

10.100.2.4 LLGL::Scissor::Scissor (const Offset2D & *offset*, const Extent2D & *extent*) [inline]

Scissor constructor with offset and extent parameters.

10.100.3 Member Data Documentation

10.100.3.1 std::int32_t LLGL::Scissor::height = 0

Right-bottom height.

10.100.3.2 std::int32_t LLGL::Scissor::width = 0

Right-bottom width.

10.100.3.3 std::int32_t LLGL::Scissor::x = 0

Left-top X coordinate.

10.100.3.4 std::int32_t LLGL::Scissor::y = 0

Left-top Y coordinate.

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

- PipelineStateFlags.h

10.101 LLGL::Shader Class Reference

Shader interface.

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

Inheritance diagram for LLGL::Shader:

Public Member Functions

- virtual const Report * GetReport () const =0

  Returns a pointer to the report or null if there is none.
- virtual bool Reflect (ShaderReflection &reflection) const =0

  Returns a reflection of the shader pipeline layout with all required resources for this shader.
- virtual UniformLocation FindUniformLocation (const char *name) const

  Returns the location of a single shader uniform by its name.
- virtual bool IsPostTessellationVertex () const

  Returns true if this is a post-tessellation vertex shader.
- 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)

10.101.1 Detailed Description

Shader interface.

See also

RenderSystem::CreateShader

10.101.2 Constructor & Destructor Documentation

10.101.2.1 LLGL::Shader::Shader (const ShaderType *type*) [protected]

10.101.3 Member Function Documentation

10.101.3.1 virtual UniformLocation LLGL::Shader::FindUniformLocation (const char * *name*) const [virtual]

Returns the location of a single shader uniform by its name.

Returns

Uniform location of the specified uniform, or -1 if there is no such uniform in the shader program.

Remarks

This is a helper function when only one or a few number of uniform locations are meant to be determined. If more uniforms are involved, use the Reflect function.
Default implementation always returns Constants::invalidLocation.

See also

Reflect

Note

Only supported with: OpenGL.

10.101.3.2 virtual const Report* LLGL::Shader::GetReport ( ) const [pure virtual]

Returns a pointer to the report or null if there is none.

Remarks

If there is a report, it might contain warnings and/or errors from shader compilation process.

See also

Report

10.101.3.3 long LLGL::Shader::GetStageFlags ( ) const

Returns the shader stage bitmask for this shader object.

See also

StageFlags

10.101.3.4 ShaderType LLGL::Shader::GetType ( ) const [inline]

Returns the type of this shader.

10.101.3.5 virtual bool LLGL::Shader::IsPostTessellationVertex ( ) const [virtual]

Returns true if this is a post-tessellation vertex shader.

Remarks

This is only used for the Metal backend to determine whether a post-tessellation vertex shader is used in conjunction with a compute kernel.
Default implementation always returns false.

Note

Only supported with: Metal.

10.101.3.6 virtual bool LLGL::Shader::Reflect (ShaderReflection & *reflection*) const [pure virtual]

Returns a reflection of the shader pipeline layout with all required resources for this shader.

Parameters

out	*reflection*	Specifies the output shader reflection. If the function returns `false`, the content of this parameter is **undefined**!

Remarks

The list of resources in the reflection output is always sorted by the following attributes (lower number means higher priority for sorting):

1. Resource type in ascending order (see BindingDescriptor::type).
2. Binding flags in ascending order (see BindingDescriptor::bindFlags).
3. Binding slot in ascending order (see BindingDescriptor::slot).

Here is an example of such a sorted list (pseudocode):

```
resources[0] = { type: ResourceType::Buffer,  bindFlags:
      BindFlags::ConstantBuffer, slot: 0 }
resources[1] = { type: ResourceType::Buffer,  bindFlags:
      BindFlags::ConstantBuffer, slot: 2 }
resources[2] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 0 }
resources[3] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 1 }
resources[4] = { type: ResourceType::Texture, bindFlags:
      BindFlags::Sampled,        slot: 2 }
resources[5] = { type: ResourceType::Sampler, bindFlags: 0,
      slot: 2 }
```

The `instanceDivisor` and `offset` members of the vertex attributes are ignored by this function.

Returns

True, if the reflection was successful. Otherwise, the shader reflection failed and the content of the output parameter `reflection` is undefined.

Note

Since Metal needs a complete pipeline state for shader reflection, this function is only supported for compute shaders in the Metal backend.

See also

ShaderReflection::resources
ShaderReflection::vertexAttributes
VertexAttribute::instanceDivisor
VertexAttribute::offset

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

- Shader.h

10.102 LLGL::ShaderCompileFlags Struct Reference

Shader compilation flags enumeration.

```
#include <ShaderFlags.h>
```

Public Types

- enum {
  Debug = (1 << 0), NoOptimization = (1 << 1), OptimizationLevel1 = (1 << 2), OptimizationLevel2 = (1 << 3),
  OptimizationLevel3 = (1 << 4), WarningsAreErrors = (1 << 5), PatchClippingOrigin = (1 << 6), Separate↩Shader = (1 << 7) }

10.102.1 Detailed Description

Shader compilation flags enumeration.

See also

ShaderDescriptor::flags

10.102.2 Member Enumeration Documentation

10.102.2.1 anonymous enum

Enumerator

Debug Generate debug information.

Remarks

This compile option is equivalent to the command line arguments `fxc /Od`, `dxc -Od`, `metal -O0`.

Note

Only supported with: HLSL, Metal.

NoOptimization Disable optimizations.

Remarks

This compile option is equivalent to command line arguments `fxc /Od`, `dxc -Od`, `metal -↩O0`.

Note

Only supported with: HLSL, Metal, GLSL (adds `#pragma optimize(off)` after the `#version`-directive).

OptimizationLevel1 Optimization level 1.

Remarks

This compile option is equivalent to command line arguments `fxc /O1`, `dxc -O1`, `metal -↩O1`.

Note

Only supported with: HLSL, Metal.

OptimizationLevel2 Optimization level 2.

Remarks

This compile option is equivalent to command line arguments `fxc /O2`, `dxc -O2`, `metal -↩O2`.

Note

Only supported with: HLSL, Metal.

OptimizationLevel3 Optimization level 3.

Remarks

This compile option is equivalent to command line arguments `fxc /O3`, `dxc -O3`, `metal -O3`.

Note

Only supported with: HLSL, Metal.

WarningsAreErrors Warnings are treated as errors.

Remarks

This compile option is equivalent to command line arguments `fxc /WX`, `dxc -WX`, `metal -Werror`.

Note

Only supported with: HLSL, Metal.

PatchClippingOrigin Patches the GLSL shader source to accommodate a flipped coordinate system from lower-left to upper-left and vice-versa, effectively injecting `gl_Position.y = -gl_Position.y;` statements into a vertex shader.

Remarks

This can be used to maintain the same vertex shader logic between GLSL and other shading languages when the screen origin is lower-left (see ScreenOrigin::LowerLeft). This flag should also only be used for shaders that render into an OpenGL texture as their coordinate system is reversed compared to the other rendering APIs. What shader stage should this flag be used with depends on what shader stage is the last to modify vertex positions before they are passed to the clipping stage, i.e. either vertex, tessellation-evaluation, or geometry shaders.

Note

Since there is no preprocessing performed prior to scanning the shader source, control-flow modifying macros are not recognized. For example, using macros that change the control flow in the main entry point or even obfuscate the declaration of the entry point will not be scanned correctly by this feature. If in doubt, write your own adjustment in the shader source like this:

```
void main() {
  // Vertex shader body ...
  #if FLIP_POSITION_Y
  gl_Position.y = -gl_Position.y;
  #endif
}
```

Then define the macro `FLIP_POSITION_Y` on the API side like this and pass it to all vertex (or tessellation-evaluation or geometry) shaders that will be used for render targets, i.e. those shaders that render into a teture instead of a Window or Canvas:

```
const LLGL::ShaderMacro myDefinesForRenderTargetShaders[] = {
  { "FLIP_POSITION_Y", myRenderer->GetRenderingCaps().screenOrigin ==
      LLGL::ScreenOrigin::LowerLeft ? "1" : "0" },
  { nullptr, nullptr } // Null terminating entry
};
```

Only supported with: GLSL.

See also

RenderingCapabilities::screenOrigin

SeparateShader Specifies whether to create separable or legacy shaders.

Remarks

This is only used for the OpenGL backend. Separate and non-separate shaders (i.e. legacy shaders) must not be mixed and matched when a graphics PSO is created! If specified, the GLSL vertex shader must contain a `gl_PerVertex` block and the GLSL fragment shader *may* contain a `gl_PerFragment` block.

See also

https://registry.khronos.org/OpenGL/extensions/ARB/ARB_separate↩_shader_objects.txt

Note

Only supported with: GLSL.

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

- ShaderFlags.h

10.103 LLGL::ShaderDescriptor Struct Reference

Shader source and binary code descriptor structure.

```
#include <ShaderFlags.h>
```

Public Member Functions

- ShaderDescriptor ()=default
- ShaderDescriptor (const ShaderDescriptor &)=default
- ShaderDescriptor & operator= (const ShaderDescriptor &)=default
- ShaderDescriptor (const ShaderType type, const char *source)

  Constructor to initialize the shader descriptor with a source filename.
- ShaderDescriptor (const ShaderType type, const char *source, const char *entryPoint, const char *profile, long flags=0)

  Constructor to initialize the shader descriptor with a source filename, entry point, profile, and optional flags.

Public Attributes

- ShaderType type = ShaderType::Undefined

  Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like. By default ShaderType::↩Undefined.
- const char * source = nullptr

  Pointer to the shader source. This is either a null terminated string or a raw byte buffer (depending on the `sourceType` member).
- std::size_t sourceSize = 0

  Specifies the size of the shader source (excluding the null terminator).
- ShaderSourceType sourceType = ShaderSourceType::CodeFile

  Specifies the type of the shader source. By default ShaderSourceType::CodeFile.
- const char * entryPoint = nullptr

  Shader entry point (shader main function). If this is null, the empty string is used. By default null.
- const char * profile = nullptr

  Shader target profile. If this is null, the empty string is used. By default null.
- const ShaderMacro * defines = nullptr

  Optional array of macro definitions. By default null.
- long flags = 0

  Optional compilation flags. By default 0.
- VertexShaderAttributes vertex

  Vertex (or geometry) shader specific attributes.
- FragmentShaderAttributes fragment

  Fragment shader specific attributes.
- ComputeShaderAttributes compute

  Compute shader specific attributes.

10.103.1 Detailed Description

Shader source and binary code descriptor structure.

See also

RenderSystem::CreateShader

10.103.2 Constructor & Destructor Documentation

10.103.2.1 LLGL::ShaderDescriptor::ShaderDescriptor ( ) [default]

10.103.2.2 LLGL::ShaderDescriptor::ShaderDescriptor (const ShaderDescriptor &) [default]

10.103.2.3 LLGL::ShaderDescriptor::ShaderDescriptor (const ShaderType *type*, const char * *source*) [inline]

Constructor to initialize the shader descriptor with a source filename.

10.103.2.4 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.

10.103.3 Member Function Documentation

10.103.3.1 ShaderDescriptor& LLGL::ShaderDescriptor::operator= (const ShaderDescriptor &) [default]

10.103.4 Member Data Documentation

10.103.4.1 ComputeShaderAttributes LLGL::ShaderDescriptor::compute

Compute shader specific attributes.

Remarks

This member is only used to specify the number of threads per threadgroup for the Metal backend.

Note

Only supported with: Metal.

10.103.4.2 const ShaderMacro* LLGL::ShaderDescriptor::defines = nullptr

Optional array of macro definitions. By default null.

Remarks

Shader macros can only be defined if `sourceType` refers to source code, i.e. ShaderSourceType::CodeString or ShaderSourceType::CodeFile. Otherwise, this field is ignored.
This must either be null or a null-terminated array of ShaderMacro entries as shown here:

```
const LLGL::ShaderMacro myDefines[] = {
    { "ENABLE_SHADER_PASS_FOO", "1" }, // first macro
    { "ENABLE_SHADER_PASS_BAR", "0" }, // second macro
    { nullptr, nullptr },              // terminate array
};
LLGL::ShaderDescriptor myShaderDesc;
myShaderDesc.defines = myDefines;
```

10.103.4.3 const char* LLGL::ShaderDescriptor::entryPoint = nullptr

Shader entry point (shader main function). If this is null, the empty string is used. By default null.

Note

Only supported with: HLSL, SPIR-V, Metal.

10.103.4.4 long LLGL::ShaderDescriptor::flags = 0

Optional compilation flags. By default 0.

Remarks

This can be a bitwise OR combination of the ShaderCompileFlags enumeration entries.

See also

ShaderCompileFlags

10.103.4.5 FragmentShaderAttributes LLGL::ShaderDescriptor::fragment

Fragment shader specific attributes.

10.103.4.6 const char* LLGL::ShaderDescriptor::profile = nullptr

Shader target profile. If this is null, the empty string is used. By default null.

Remarks

This is renderer API dependent and is forwarded to the respective shader compiler.
Here are a few examples:

- For HLSL: `"vs_5_0"` specifies vertex shader model 5.0.
- For Metal: `"2.1"` specifies shader version 2.1.

Note

Only supported with: HLSL, Metal.

See also

https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.↩85).aspx

10.103.4.7 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

10.103.4.8 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 binary buffer source type (i.e. ShaderSourceType::BinaryBuffer), this must not be zero!

See also

source

10.103.4.9 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

10.103.4.10 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.

10.103.4.11 VertexShaderAttributes LLGL::ShaderDescriptor::vertex

Vertex (or geometry) shader specific attributes.

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

- ShaderFlags.h

10.104 LLGL::ShaderMacro Struct Reference

Shader macro structure with name and optional body definition.

```
#include <ShaderFlags.h>
```

Public Member Functions

- ShaderMacro ()=default
- ShaderMacro (const ShaderMacro &)=default
- ShaderMacro & operator= (const ShaderMacro &)=default
- ShaderMacro (const char *name, const char *definition=nullptr)

  Constructor to initialize the shader macro with a name and an optional body definition.

Public Attributes

- const char * name = nullptr

  Specifies the name of the macro. This must not be null.
- const char * definition = nullptr

  Specifies the macro definition. If this is null, the macro has no body definition.

10.104.1 Detailed Description

Shader macro structure with name and optional body definition.

See also

ShaderDescriptor::defines

10.104.2 Constructor & Destructor Documentation

10.104.2.1 LLGL::ShaderMacro::ShaderMacro ( ) [default]

10.104.2.2 LLGL::ShaderMacro::ShaderMacro (const ShaderMacro &) [default]

10.104.2.3 LLGL::ShaderMacro::ShaderMacro (const char * *name*, const char * *definition* = nullptr) [inline]

Constructor to initialize the shader macro with a name and an optional body definition.

10.104.3 Member Function Documentation

10.104.3.1 ShaderMacro& LLGL::ShaderMacro::operator= (const ShaderMacro &) [default]

10.104.4 Member Data Documentation

10.104.4.1 const char* LLGL::ShaderMacro::definition = nullptr

Specifies the macro definition. If this is null, the macro has no body definition.

10.104.4.2 const char* LLGL::ShaderMacro::name = nullptr

Specifies the name of the macro. This must not be null.

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

- ShaderFlags.h

10.105 LLGL::ShaderReflection Struct Reference

Shader reflection structure.

```
#include <ShaderReflection.h>
```

Public Attributes

- std::vector< ShaderResource > resources

  List of all shader reflection resource views.
- std::vector< ShaderUniform > uniforms

  List of all uniforms (a.k.a. shader constants).
- VertexShaderAttributes vertex

  Reflection data that is specificly for the vertex shader.
- FragmentShaderAttributes fragment

  Reflection data that is specificly for the fragment shader.
- ComputeShaderAttributes compute

  Reflection data that is specificly for the compute shader.

10.105.1 Detailed Description

Shader reflection structure.

Remarks

Contains all information of resources and attributes that can be queried from a shader program. This is not a "descriptor", because it is only used as output from an interface rather than a description to create something.

See also

Shader::Reflect

10.105.2 Member Data Documentation

10.105.2.1 ComputeShaderAttributes LLGL::ShaderReflection::compute

Reflection data that is specificly for the compute shader.

10.105.2.2 FragmentShaderAttributes LLGL::ShaderReflection::fragment

Reflection data that is specificly for the fragment shader.

10.105.2.3 std::vector<ShaderResource> LLGL::ShaderReflection::resources

List of all shader reflection resource views.

10.105.2.4 std::vector<ShaderUniform> LLGL::ShaderReflection::uniforms

List of all uniforms (a.k.a. shader constants).

Note

Only supported with: OpenGL, Vulkan.

10.105.2.5 VertexShaderAttributes LLGL::ShaderReflection::vertex

Reflection data that is specificly for the vertex shader.

Remarks

The shader reflection only considers the following members of the VertexAttribute structure, for both `input↩Attribs` and `outputAttribs` in VertexShaderAttributes:

- `name`
- `format`
- `location`
- `semanticIndex`
- `systemValue`

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

- ShaderReflection.h

10.106 LLGL::ShaderResource Struct Reference

Shader reflection resource structure.

```
#include <ShaderReflection.h>
```

Public Attributes

- BindingDescriptor binding

  Binding descriptor with resource name, binding slot, flags, and array size.
- std::uint32_t constantBufferSize = 0

  Specifies the size (in bytes) for a constant buffer resource.
- StorageBufferType storageBufferType = StorageBufferType::Undefined

  Specifies the sub-type of a storage buffer resource.

10.106.1 Detailed Description

Shader reflection resource structure.

See also

ShaderReflection::resources
BindingDescriptor

10.106.2 Member Data Documentation

10.106.2.1 BindingDescriptor LLGL::ShaderResource::binding

Binding descriptor with resource name, binding slot, flags, and array size.

Remarks

Although the `name` attribute in the BindingDescriptor structure is optional for pipeline layouts, the shader reflection always queries this attribute as well.

10.106.2.2 std::uint32_t LLGL::ShaderResource::constantBufferSize = 0

Specifies the size (in bytes) for a constant buffer resource.

Remarks

Additional attribute exclusively used for constant buffer resources. For all other resources, i.e. when `type` is not equal to ShaderResourceType::ConstantBuffer, this attribute is zero.

See also

ResourceType::ConstantBuffer

10.106.2.3 StorageBufferType LLGL::ShaderResource::storageBufferType = StorageBufferType::Undefined

Specifies the sub-type of a storage buffer resource.

Remarks

Additional attribute exclusively used for storage buffer resources.

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

- ShaderReflection.h

10.107 LLGL::ShaderUniform Struct Reference

Shader reflection uniform structure.

```
#include <ShaderReflection.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.

10.107.1 Detailed Description

Shader reflection uniform structure.

See also

ShaderReflection::uniforms

10.107.2 Member Data Documentation

10.107.2.1 UniformLocation LLGL::ShaderUniform::location = 0

Internal location of the uniform within a shader program.

10.107.2.2 std::string LLGL::ShaderUniform::name

Name of the uniform inside the shader.

10.107.2.3 std::uint32_t LLGL::ShaderUniform::size = 0

Array size of the uniform.

10.107.2.4 UniformType LLGL::ShaderUniform::type = UniformType::Undefined

Data type of the uniform. By default UniformType::Undefined.

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

- ShaderReflection.h

10.108 LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy > Class Template Reference

Generic container class for consecutive arrays optimized for small sizes.

```
#include <SmallVector.h>
```

Public Types

- using value_type = T
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using reference = value_type &
- using const_reference = const value_type &
- using pointer = value_type *
- using const_pointer = const value_type *
- using iterator = value_type *
- using const_iterator = const value_type *
- using reverse_iterator = std::reverse_iterator< iterator >
- using const_reverse_iterator = std::reverse_iterator< const_iterator >

Public Member Functions

- SmallVector ()
- SmallVector (const SmallVector &other)
- SmallVector (SmallVector &&other)
- template<typename InputIter >
  SmallVector (InputIter from, InputIter to)
- SmallVector (const std::initializer_list< T > &list)
- template<template< typename, typename > class OtherContainer, typename OtherAllocator >
  SmallVector (const OtherContainer< T, OtherAllocator > &other)
- ~SmallVector ()
- bool empty () const noexcept
- size_type size () const noexcept
- size_type capacity () const noexcept
- pointer data () noexcept
- const_pointer data () const noexcept
- reference at (size_type pos) noexcept
- const_reference at (size_type pos) const noexcept
- reference front () noexcept
- const_reference front () const noexcept
- reference back () noexcept
- const_reference back () const noexcept
- void clear ()
- void resize (size_type size)
- void resize (size_type size, const value_type &value)
- void reserve (size_type size)
- void shrink_to_fit ()
- void push_back (const value_type &value)
- void push_back (value_type &&value)
- void pop_back ()
- iterator insert (const_iterator pos, const value_type &value)
- template<typename InputIter >
  iterator insert (const_iterator pos, InputIter from, InputIter to)
- iterator insert (const_iterator pos, const std::initializer_list< T > &list)
- iterator erase (const_iterator pos)
- iterator erase (const_iterator from, const_iterator to)
- void swap (SmallVector &other)
- iterator begin ()
- const_iterator begin () const
- const_iterator cbegin () const

- reverse_iterator rbegin ()
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const
- iterator end ()
- const_iterator end () const
- const_iterator cend () const
- reverse_iterator rend ()
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const
- SmallVector & operator= (const SmallVector &rhs)
- SmallVector & operator= (SmallVector &&rhs)
- const_reference operator[ ] (size_type pos) const
- reference operator[ ] (size_type pos)
- operator ArrayView< T > () const

10.108.1 Detailed Description

template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf>
class LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >

Generic container class for consecutive arrays optimized for small sizes.

Template Parameters

T	Specifies the array element type.
LocalCapacity	Specifies the capacity of the local buffer. Up to this number of elemnts no dynamic memory allocation is necessary. By default 16.
Allocator	Specifies the memory allocator. This has to be compatible with std::allocator. By default std::allocator<T>.
GrowStrategy	Specifies the strategy to grow the internal storage when the container switched to heap allocations. By default GrowStrategyAddHalf.

10.108.2 Member Typedef Documentation

10.108.2.1 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::const_iterator = const value_type*

10.108.2.2 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::const_pointer = const value_type*

10.108.2.3 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::const_reference = const value_type&

10.108.2.4 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::const_reverse_iterator = std::reverse_iterator<const_iterator>

10.108.2.5 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**difference_type** = std::ptrdiff_t

10.108.2.6 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**iterator** = **value_type***

10.108.2.7 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**pointer** = **value_type***

10.108.2.8 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**reference** = **value_type&**

10.108.2.9 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**reverse_iterator** = std::reverse_iterator<**iterator**>

10.108.2.10 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**size_type** = std::size_t

10.108.2.11 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> using **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**value_type** = T

10.108.3 Constructor & Destructor Documentation

10.108.3.1 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** ( ) [inline]

10.108.3.2 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** (const **SmallVector**< T, LocalCapacity, Allocator, GrowStrategy > & *other*) [inline]

10.108.3.3 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** (**SmallVector**< T, LocalCapacity, Allocator, GrowStrategy > && *other*) [inline]

10.108.3.4 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> template<typename InputIter > **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** (InputIter *from,* InputIter *to*) [inline]

10.108.3.5 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** (const std::initializer_list< T > & *list*) [inline]

10.108.3.6 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> template<template< typename, typename > class OtherContainer, typename OtherAllocator > **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::**SmallVector** (const OtherContainer< T, OtherAllocator > & *other*) [inline]

10.108.3.7 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::~**SmallVector** ( ) [inline]

10.108.4 Member Function Documentation

10.108.4.1 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::at (**size_type** *pos*) [inline], [noexcept]

10.108.4.2 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::at (**size_type** *pos*) const [inline], [noexcept]

10.108.4.3 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::back ( ) [inline], [noexcept]

10.108.4.4 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::back ( ) const [inline], [noexcept]

10.108.4.5 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::begin ( ) [inline]

10.108.4.6 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::begin ( ) const [inline]

10.108.4.7 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **size_type LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::capacity ( ) const [inline], [noexcept]

10.108.4.8 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::cbegin ( ) const [inline]

10.108.4.9 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::cend ( ) const [inline]

10.108.4.10 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::clear ( ) [inline]

10.108.4.11 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reverse_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::crbegin ( ) const [inline]

10.108.4.12 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reverse_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::crend ( ) const [inline]

10.108.4.13 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **pointer LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::data ( ) [inline], [noexcept]

10.108.4.14 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_pointer LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::data ( ) const [inline], [noexcept]

10.108.4.15 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> bool **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::empty ( ) const [inline], [noexcept]

10.108.4.16 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::end ( ) [inline]

10.108.4.17 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::end ( ) const [inline]

10.108.4.18 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::erase (**const_iterator** *pos*) [inline]

10.108.4.19 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::erase (**const_iterator** *from,* **const_iterator** *to*) [inline]

10.108.4.20 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::front ( ) [inline], [noexcept]

10.108.4.21 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reference LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::front ( ) const [inline], [noexcept]

10.108.4.22 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::insert (**const_iterator** *pos,* const **value_type** & *value*) [inline]

10.108.4.23 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> template<typename InputIter > **iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::insert (**const_iterator** *pos,* InputIter *from,* InputIter *to*) [inline]

10.108.4.24 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> iterator LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::insert (const_iterator *pos,* const std::initializer_list< T > & *list*) [inline]

10.108.4.25 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::operator ArrayView< T > ( ) const [inline]

10.108.4.26 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> SmallVector& LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::operator= (const SmallVector< T, LocalCapacity, Allocator, GrowStrategy > & *rhs*) [inline]

10.108.4.27 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> SmallVector& LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::operator= (SmallVector< T, LocalCapacity, Allocator, GrowStrategy > && *rhs*) [inline]

10.108.4.28 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> const_reference LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::operator[] (size_type *pos*) const [inline]

10.108.4.29 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> reference LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::operator[] (size_type *pos*) [inline]

10.108.4.30 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::pop_back ( ) [inline]

10.108.4.31 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::push_back (const value_type & *value*) [inline]

10.108.4.32 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::push_back (value_type && *value*) [inline]

10.108.4.33 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> reverse_iterator LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::rbegin ( ) [inline]

10.108.4.34 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> const_reverse_iterator LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::rbegin ( ) const [inline]

10.108.4.35 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> reverse_iterator LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >::rend ( ) [inline]

10.108.4.36 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **const_reverse_iterator LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::rend ( ) const [inline]

10.108.4.37 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::reserve (**size_type** *size*) [inline]

10.108.4.38 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::resize (**size_type** *size*) [inline]

10.108.4.39 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::resize (**size_type** *size,* const **value_type** & *value*) [inline]

10.108.4.40 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::shrink_to_fit ( ) [inline]

10.108.4.41 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> **size_type LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::size ( ) const [inline], [noexcept]

10.108.4.42 template<typename T, std::size_t LocalCapacity = 16, typename Allocator = std::allocator<T>, typename GrowStrategy = GrowStrategyAddHalf> void **LLGL::SmallVector**< T, LocalCapacity, Allocator, GrowStrategy >::swap (**SmallVector**< T, LocalCapacity, Allocator, GrowStrategy > & *other*) [inline]

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

- SmallVector.h

10.109 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.

10.109.1 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

10.109.2 Constructor & Destructor Documentation

10.109.2.1 LLGL::SrcImageDescriptor::SrcImageDescriptor ( ) [default]

10.109.2.2 LLGL::SrcImageDescriptor::SrcImageDescriptor (const SrcImageDescriptor &) [default]

10.109.2.3 LLGL::SrcImageDescriptor::SrcImageDescriptor (ImageFormat *format*, DataType *dataType*, const void * *data*, std::size_t *dataSize*) [inline]

Constructor to initialize all attributes.

10.109.3 Member Data Documentation

10.109.3.1 const void* LLGL::SrcImageDescriptor::data = nullptr

Pointer to the read-only image data.

10.109.3.2 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.

10.109.3.3 DataType LLGL::SrcImageDescriptor::dataType = DataType::UInt8

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

10.109.3.4 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

10.110 LLGL::StageFlags Struct Reference

Shader stage flags enumeration.

```
#include <ShaderFlags.h>
```

Public Types

- enum {
  VertexStage = (1 << 0), TessControlStage = (1 << 1), TessEvaluationStage = (1 << 2), GeometryStage = (1 << 3),
  FragmentStage = (1 << 4), ComputeStage = (1 << 5), AllTessStages = (TessControlStage | TessEvaluationStage), AllGraphicsStages = (VertexStage | AllTessStages | GeometryStage | FragmentStage),
  AllStages = (AllGraphicsStages | ComputeStage) }

10.110.1 Detailed Description

Shader stage flags enumeration.

Remarks

Specifies which shader stages are affected by a state change, e.g. to which shader stages a constant buffer is bound.

See also

BindingDescriptor::stageFlags

10.110.2 Member Enumeration Documentation

10.110.2.1 anonymous enum

Enumerator

VertexStage Specifies the vertex shader stage.

TessControlStage Specifies the tessellation-control shader stage (also referred to as "Hull Shader").

TessEvaluationStage Specifies the tessellation-evaluation shader stage (also referred to as "Domain Shader").

GeometryStage Specifies the geometry shader stage.

FragmentStage Specifies the fragment shader stage (also referred to as "Pixel Shader").

ComputeStage Specifies the compute shader stage.

AllTessStages Specifies all tessellation stages, i.e. tessellation-control-, tessellation-evaluation shader stages.

AllGraphicsStages Specifies all graphics pipeline shader stages, i.e. vertex-, tessellation-, geometry-, and fragment shader stages.

AllStages Specifies all shader stages.

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

- ShaderFlags.h

10.111 LLGL::StencilDescriptor Struct Reference

Stencil state descriptor structure.

```
#include <PipelineStateFlags.h>
```

Public Attributes

- bool testEnabled = false

  Specifies whether the stencil test is enabled or disabled. By default disabled.
- bool referenceDynamic = false

  Specifies whether the stencil reference values will be set dynamically with the command buffer. By default false.
- StencilFaceDescriptor front

  Specifies the front face settings for the stencil test.
- StencilFaceDescriptor back

  Specifies the back face settings for the stencil test.

10.111.1 Detailed Description

Stencil state descriptor structure.

See also

GraphicsPipelineDescriptor::stencil

10.111.2 Member Data Documentation

10.111.2.1 StencilFaceDescriptor LLGL::StencilDescriptor::back

Specifies the back face settings for the stencil test.

10.111.2.2 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

10.111.2.3 bool LLGL::StencilDescriptor::referenceDynamic = false

Specifies whether the stencil reference values will be set dynamically with the command buffer. By default false.

Remarks

If this is true, StencilFaceDescriptor::reference in `front` and `back` is ignored and the stencil reference values must be set with the `SetStencilReference` function everytime the graphics pipeline is set.

See also

CommandBuffer::SetStencilReference

10.111.2.4 bool LLGL::StencilDescriptor::testEnabled = false

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

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

- PipelineStateFlags.h

10.112 LLGL::StencilFaceDescriptor Struct Reference

Stencil face descriptor structure.

```
#include <PipelineStateFlags.h>
```

Public Attributes

- StencilOp stencilFailOp = StencilOp::Keep

  Specifies the operation to take when the stencil test fails. By default StencilOp::Keep.
- StencilOp depthFailOp = StencilOp::Keep

  Specifies the operation to take when the stencil test passes but the depth test fails. By default StencilOp::Keep.
- StencilOp depthPassOp = StencilOp::Keep

  Specifies the operation to take when both the stencil test and the depth test pass. By default StencilOp::Keep.
- CompareOp compareOp = CompareOp::Less

  Specifies the stencil compare operation. By default CompareOp::Less.
- std::uint32_t readMask = ~0u

  Specifies the portion of the depth-stencil buffer for reading stencil data. By default `0xFFFFFFFF`.
- std::uint32_t writeMask = ~0u

  Specifies the portion of the depth-stencil buffer for writing stencil data. By default `0xFFFFFFFF`.
- std::uint32_t reference = 0u

  Specifies the stencil reference value. By default 0.

10.112.1 Detailed Description

Stencil face descriptor structure.

See also

StencilDescriptor::front
StencilDescriptor::back

10.112.2 Member Data Documentation

10.112.2.1 CompareOp LLGL::StencilFaceDescriptor::compareOp = CompareOp::Less

Specifies the stencil compare operation. By default CompareOp::Less.

10.112.2.2 StencilOp LLGL::StencilFaceDescriptor::depthFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test passes but the depth test fails. By default StencilOp::Keep.

10.112.2.3 StencilOp LLGL::StencilFaceDescriptor::depthPassOp = StencilOp::Keep

Specifies the operation to take when both the stencil test and the depth test pass. By default StencilOp::Keep.

10.112.2.4 std::uint32_t LLGL::StencilFaceDescriptor::readMask = ~0u

Specifies the portion of the depth-stencil buffer for reading stencil data. By default `0xFFFFFFFF`.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (i.e. `readMask & 0xFF`) of the read mask value of the front face will be used.

See also

StencilDescriptor::front

10.112.2.5 std::uint32_t LLGL::StencilFaceDescriptor::reference = 0u

Specifies the stencil reference value. By default 0.

Remarks

This value will be used when the stencil operation is StencilOp::Replace.
If `referenceDynamic` is set to true, this member is ignored.

Note

For Direct3D 11 and Direct3D 12, only the stencil reference value of the front face will be used.

See also

StencilDescriptor::front
StencilDescriptor::referenceDynamic
CommandBuffer::SetStencilReference

10.112.2.6 StencilOp LLGL::StencilFaceDescriptor::stencilFailOp = StencilOp::Keep

Specifies the operation to take when the stencil test fails. By default StencilOp::Keep.

10.112.2.7 std::uint32_t LLGL::StencilFaceDescriptor::writeMask = ~0u

Specifies the portion of the depth-stencil buffer for writing stencil data. By default `0xFFFFFFFF`.

Note

For Direct3D 11 and Direct3D 12, only the first 8 least significant bits (i.e. `writeMask & 0xFF`) of the write mask value of the front face will be used.

See also

StencilDescriptor::front

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

- PipelineStateFlags.h

10.113 LLGL::Surface Class Reference

The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on mobile platforms).

```
#include <Surface.h>
```

Inheritance diagram for LLGL::Surface:

Public Member Functions

- virtual bool GetNativeHandle (void *nativeHandle, std::size_t nativeHandleSize) 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 (Extent2D *resolution, bool *fullscreen)=0

  Adapts the surface to fits the needs for the specified video mode.
- virtual void ResetPixelFormat ()=0

  Resets the internal pixel format of the surface.
- virtual bool ProcessEvents ()=0

  Processes all events for this surface, i.e. input-, movement-, resize-, and other events.
- virtual Display * FindResidentDisplay () const =0

  Returns the Display interface where this surface is resident in.

Additional Inherited Members

10.113.1 Detailed Description

The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on mobile 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

10.113.2 Member Function Documentation

10.113.2.1 virtual bool LLGL::Surface::AdaptForVideoMode (Extent2D * *resolution*, bool * *fullscreen*) [pure virtual]

Adapts the surface to fits the needs for the specified video mode.

Parameters

in,out	*resolution*	Optional pointer to the input and output resolution.

Returns

If the video mode has been accepted with no modifications and this surface has been updated then the return value is true. Otherwise, `resolution` and/or `fullscreen` arguments have been modified to the value this surface supports and the return value is false.

Implemented in LLGL::Window, and LLGL::Canvas.

10.113.2.2 virtual Display* LLGL::Surface::FindResidentDisplay ( ) const [pure virtual]

Returns the Display interface where this surface is resident in.

Remarks

A surface is considered resident in a display if more than the half of its client area is visible in that display.

Returns

New instance of a Display where this surface is resident or null if there no display has been found.

Implemented in LLGL::Window, and LLGL::Canvas.

10.113.2.3 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

10.113.2.4 virtual bool LLGL::Surface::GetNativeHandle (void * *nativeHandle*, std::size_t *nativeHandleSize*) const [pure virtual]

Returns the native surface handle.

Parameters

out	*nativeHandle*	Raw pointer to the platform specific structure to store the native handle. This must be of type LLGL::NativeHandle.
in	*nativeHandleSize*	Specifies the size (in bytes) of the native handle structure for robustness. This must be `sizeof(LLGL::NativeHandle)`.

Returns

True if the native handle was successfully retrieved. Otherwise, `nativeHandleSize` specifies an incompatible structure size.

Remarks

This must be casted to a platform specific structure:

```
// Example for a custom Win32 window class
#include <LLGL/Platform/NativeHandle.h>
//...
bool MyWindowClass::GetNativeHandle(void* nativeHandle, std::size_t nativeHandleSize) {
    if (nativeHandleSize == sizeof(LLGL::NativeHandle)) {
        auto handle = reinterpret_cast<LLGL::NativeHandle*>(nativeHandle);
        //handle->window = 'some HWND window handle';
        return true;
    }
    return false;
}
```

10.113.2.5 virtual bool LLGL::Surface::ProcessEvents ( ) [pure virtual]

Processes all events for this surface, i.e. input-, movement-, resize-, and other events.

Remarks

This function is only implemented by the Window and Canvas interfaces.

See also

Window::ProcessEvents
Canvas::ProcessEvents

Implemented in LLGL::Window, and LLGL::Canvas.

10.113.2.6 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

10.114 LLGL::SwapChain Class Reference

Swap-chain interface.

```
#include <SwapChain.h>
```

Inheritance diagram for LLGL::SwapChain:

Public Member Functions

- ~SwapChain ()

  Release the internal data.
- Extent2D GetResolution () const final

  Returns the resolution of the current video mode.
- std::uint32_t GetNumColorAttachments () const final

  Returns 1, since each swap-chain has always a single color attachment.
- bool HasDepthAttachment () const final

  Returns true if this swap-chain has a depth format.
- bool HasStencilAttachment () const final

  Returns true if this swap-chain 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 swap-chain).
- virtual Format GetColorFormat () const =0

  Returns the color format of this swap-chain.
- virtual Format GetDepthStencilFormat () const =0

  Returns the depth-stencil format of this swap-chain.
- bool ResizeBuffers (const Extent2D &resolution, long flags=0)

  Resizes all swap buffers of this swap-chain.
- virtual bool SetVsyncInterval (std::uint32_t vsyncInterval)=0

  Sets the new vertical sychronization (V-sync) interval for this swap chain.
- bool SwitchFullscreen (bool enable)

  Puts the display, the swap-chain's surface is resident in, into fullscreen mode or puts it back into normal mode.
- Surface & GetSurface () const

  Returns the surface which is used to present the content on the screen.

Protected Member Functions

- SwapChain ()

  Allocates the internal data.
- SwapChain (const SwapChainDescriptor &desc)

  Constructor to initialize the swap-chain with the specified video mode and V-sync.
- void SetOrCreateSurface (const std::shared_ptr< Surface > &surface, const Extent2D &size, bool fullscreen, const void *windowContext)

  Sets the swap-chain surface or creates one if 'surface' is null, and switches to fullscreen mode if enabled.
- void ShareSurfaceAndConfig (SwapChain &other)

  Shares the surface and resolution with another swap-chain.
- bool SetDisplayFullscreenMode (const Extent2D &resolution)

  Puts the display the swap-chain's surface is resident in into fullscreen mode.
- bool ResetDisplayFullscreenMode ()

  Puts the display the swap-chain's surface is resident in back into normal mode.
- virtual bool ResizeBuffersPrimary (const Extent2D &resolution)=0

10.114.1 Detailed Description

Swap-chain interface.

Remarks

Each swap-chain has its own surface and swap buffers to draw into.

See also

RenderSystem::CreateSwapChain
CommandBuffer::BeginRenderPass

10.114.2 Constructor & Destructor Documentation

10.114.2.1 LLGL::SwapChain::~SwapChain ( )

Release the internal data.

10.114.2.2 LLGL::SwapChain::SwapChain ( ) [protected]

Allocates the internal data.

10.114.2.3 LLGL::SwapChain::SwapChain (const SwapChainDescriptor & *desc*) [protected]

Constructor to initialize the swap-chain with the specified video mode and V-sync.

10.114.3 Member Function Documentation

10.114.3.1 virtual Format LLGL::SwapChain::GetColorFormat ( ) const [pure virtual]

Returns the color format of this swap-chain.

Remarks

This may depend on the settings specified for the video mode. A common value for a swap-chain color format is Format::BGRA8UNorm.

See also

AttachmentFormatDescriptor::format
Format

10.114.3.2 virtual Format LLGL::SwapChain::GetDepthStencilFormat ( ) const [pure virtual]

Returns the depth-stencil format of this swap-chain.

Remarks

This may depend on the settings specified for the video mode.

See also

AttachmentFormatDescriptor::format
Format

10.114.3.3 std::uint32_t LLGL::SwapChain::GetNumColorAttachments ( ) const [final], [virtual]

Returns 1, since each swap-chain has always a single color attachment.

Implements LLGL::RenderTarget.

10.114.3.4 Extent2D LLGL::SwapChain::GetResolution ( ) const [final], [virtual]

Returns the resolution of the current video mode.

See also

ResizeBuffers

Implements LLGL::RenderTarget.

10.114.3.5 Surface& LLGL::SwapChain::GetSurface ( ) const

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&>(mySwapChain->GetSurface());
```

10.114.3.6 bool LLGL::SwapChain::HasDepthAttachment ( ) const [final], [virtual]

Returns true if this swap-chain has a depth format.

See also

GetDepthStencilFormat
IsDepthFormat

Implements LLGL::RenderTarget.

10.114.3.7 bool LLGL::SwapChain::HasStencilAttachment ( ) const [final], [virtual]

Returns true if this swap-chain has a stencil format.

See also

GetDepthStencilFormat
IsStencilFormat

Implements LLGL::RenderTarget.

10.114.3.8 virtual void LLGL::SwapChain::Present ( ) [pure virtual]

Swaps the back buffer with the front buffer to present it on the screen (or rather on this swap-chain).

10.114.3.9 bool LLGL::SwapChain::ResetDisplayFullscreenMode ( ) [protected]

Puts the display the swap-chain's surface is resident in back into normal mode.

See also

SetDisplayFullscreenMode

10.114.3.10 bool LLGL::SwapChain::ResizeBuffers (const Extent2D & *resolution*, long *flags* = 0)

Resizes all swap buffers of this swap-chain.

Parameters

in	*resolution*	Specifies the new resolution.
in	*flags*	Optional flags to specify whether the swap-chain's surface is to be adjusted as well and to toggle fullscreen mode.

See also

GetResolution
ResizeBuffersFlags

10.114.3.11 virtual bool LLGL::SwapChain::ResizeBuffersPrimary (const Extent2D & *resolution*) [protected], [pure virtual]

Primary function to resize all swap buffers.

See also

ResizeBuffers

10.114.3.12 bool LLGL::SwapChain::SetDisplayFullscreenMode (const Extent2D & *resolution*) [protected]

Puts the display the swap-chain's surface is resident in into fullscreen mode.

See also

ResetDisplayFullscreenMode

10.114.3.13 void LLGL::SwapChain::SetOrCreateSurface (const std::shared_ptr< Surface > & *surface*, const Extent2D & *size*, bool *fullscreen*, const void * *windowContext*) [protected]

Sets the swap-chain 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 swap-chain.
in	*size*	Specifies the surface content size. This is only used if `surface` is null. Otherwise, the size is determined by the content size of the specified surface (i.e. with the Surface::GetContentSize function).
in	*fullscreen*	Specifies whether to put the surface into fullscreen mode.
in	*windowContext*	Optional pointer to a NativeContextHandle structure. This is only used for desktop platforms.

See also

WindowDescriptor::windowContext
Surface::GetContentSize
SwitchFullscreenMode

10.114.3.14 virtual bool LLGL::SwapChain::SetVsyncInterval (std::uint32_t *vsyncInterval*) [pure virtual]

Sets the new vertical sychronization (V-sync) interval for this swap chain.

Parameters

in	*vsyncInterval*	Specifies the new V-sync interface.

Returns

True on success, otherwise the V-sync value is invalid for this swap chain.

Remarks

This is typically 0 to disable V-sync or 1 to enable V-sync, but higher values are possible, too. A value of 2 for instance effectively halves the frame refresh rate that the active display is capable of, e.g. a display with a refresh rate of 60 Hz and a V-sync value of 2 limits the frame rate to 30 Hz.

10.114.3.15 void LLGL::SwapChain::ShareSurfaceAndConfig (**SwapChain** & *other*) [protected]

Shares the surface and resolution with another swap-chain.

Note

This is only used by the renderer debug layer.

10.114.3.16 bool LLGL::SwapChain::SwitchFullscreen (bool *enable*)

Puts the display, the swap-chain's surface is resident in, into fullscreen mode or puts it back into normal mode.

Parameters

in	*enable*	If true, puts the display into fullscreen mode (Display::SetDisplayMode). Otherwise, puts the display back into normal mode (Display::ResetDisplayMode).

Returns

True on success, otherwise the display does not support the resolution of this swap-chain.

Remarks

When switchting back from fullscreen into normal mode, this function restores the previous position of the swap-chain's surface.

See also

Display::SetDisplayMode
Display::ResetDisplayMode

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

- SwapChain.h

10.115 LLGL::SwapChainDescriptor Struct Reference

Swap chain descriptor structure.

```
#include <SwapChainFlags.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.
- std::uint32_t samples = 1

  Number of samples for the swap-chain buffers. By default 1.
- std::uint32_t swapBuffers = 2

  Number of swap buffers. By default 2 (for double-buffering).
- bool fullscreen = false

  Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

10.115.1 Detailed Description

Swap chain descriptor structure.

See also

RenderSystem::CreateSwapChain

10.115.2 Member Data Documentation

10.115.2.1 int LLGL::SwapChainDescriptor::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 swap-chain, use the SwapChain::GetColorFormat function.

See also

SwapChain::GetColorFormat

10.115.2.2 int LLGL::SwapChainDescriptor::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 swap-chain, use the SwapChain::GetDepthStencil↩Format function.

See also

SwapChain::GetDepthStencilFormat

10.115.2.3 bool LLGL::SwapChainDescriptor::fullscreen = false

Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.

10.115.2.4 Extent2D LLGL::SwapChainDescriptor::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

10.115.2.5 std::uint32_t LLGL::SwapChainDescriptor::samples = 1

Number of samples for the swap-chain buffers. By default 1.

Remarks

If the specified number of samples is not supported, LLGL will silently reduce it. The actual number of samples can be queried by the `GetSamples` function of the RenderTarget interface.

See also

RenderTarget::GetSamples

10.115.2.6 int LLGL::SwapChainDescriptor::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 swap-chain, use the SwapChain::GetDepth↩StencilFormat function.

See also

SwapChain::GetDepthStencilFormat

10.115.2.7 std::uint32_t LLGL::SwapChainDescriptor::swapBuffers = 2

Number of swap 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.

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

- SwapChainFlags.h

10.116 LLGL::TessellationDescriptor Struct Reference

Tessellation descriptor structure for the graphics pipeline.

```
#include <PipelineStateFlags.h>
```

Public Attributes

- TessellationPartition partition = TessellationPartition::Undefined

  Specifies the partition mode of the tessellator stage. By default TessellationPartition::Undefined.
- Format indexFormat = Format::Undefined

  Specifies the index buffer format. By default.
- std::uint32_t maxTessFactor = 64

  Specifies the maximum tessellation factor. By default 64.
- bool outputWindingCCW = false

  If enabled, the output topology is in counter-clockwise winding order. By default disabled.

10.116.1 Detailed Description

Tessellation descriptor structure for the graphics pipeline.

Remarks

This is only used for the Metal backend or shader reflection.

See also

GraphicsPipelineDescriptor::tessellation
RenderingFeatures::hasTessellatorStage

10.116.2 Member Data Documentation

10.116.2.1 Format LLGL::TessellationDescriptor::indexFormat = Format::Undefined

Specifies the index buffer format. By default.

Remarks

If patches are rendered with an index buffer (i.e. `DrawIndexed` or `DrawIndexedInstanced`) this must be either Format::R16UInt or Format::R32UInt.

See also

CommandBuffer::DrawIndexed
CommandBuffer::DrawIndexedInstanced

10.116.2.2 std::uint32_t LLGL::TessellationDescriptor::maxTessFactor = 64

Specifies the maximum tessellation factor. By default 64.

Remarks

Depending on the partition mode, this value must be:

- TessellationPartition::Integer: An odd or even number.
- TessellationPartition::Pow2: A power of two.
- TessellationPartition::FractionalOdd: An even number (same as `FractionalEven`).
- TessellationPartition::FractionalEven: An even number (same as `FractionalOdd`).

This value is automatically clamped to the maximum value allowed by the rendering API, e.g. 64 for macOS and 16 for iOS.
Equivalent of `[maxtessfactor(64.0)]` in HLSL.

See also

RenderingLimits::maxTessFactor

10.116.2.3 bool LLGL::TessellationDescriptor::outputWindingCCW = false

If enabled, the output topology is in counter-clockwise winding order. By default disabled.

Remarks

Equivalent of `[outputtopology("triangle_ccw")]` in HLSL and `layout(ccw)` in GLSL.

See also

RasterizerDescriptor::frontCCW

10.116.2.4 TessellationPartition LLGL::TessellationDescriptor::partition = TessellationPartition::Undefined

Specifies the partition mode of the tessellator stage. By default TessellationPartition::Undefined.

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

- PipelineStateFlags.h

10.117 LLGL::Texture Class Reference

Texture interface.

```
#include <Texture.h>
```

Inheritance diagram for LLGL::Texture:

Public Member Functions

- ResourceType GetResourceType () const override final

  Returns ResourceType::Texture.
- std::uint32_t GetMemoryFootprint () const

  Returns the memory footprint (in bytes) of the entire texture.
- std::uint32_t GetMemoryFootprint (const Extent3D &extent, const TextureSubresource &subresource) const

  Returns the memory footprint (in bytes) of the specified subresource and extent for this texture.
- TextureType GetType () const

  Returns the type of this texture.
- long GetBindFlags () const
- virtual TextureDescriptor GetDesc () const =0

  Queries a descriptor of this texture.
- virtual Extent3D GetMipExtent (std::uint32_t mipLevel) const =0

  Returns the texture extent for the specified MIP-level. This also includes the number of array layers.
- virtual Format GetFormat () const =0

Protected Member Functions

- Texture (const TextureType type, long bindFlags)

10.117.1 Detailed Description

Texture interface.

See also

RenderSystem::CreateTexture
TextureDescriptor
TextureLocation
TextureRegion

10.117.2 Constructor & Destructor Documentation

10.117.2.1 LLGL::Texture::Texture (const TextureType *type*, long *bindFlags*) [protected]

10.117.3 Member Function Documentation

10.117.3.1 long LLGL::Texture::GetBindFlags ( ) const [inline]

10.117.3.2 virtual TextureDescriptor LLGL::Texture::GetDesc ( ) const [pure virtual]

Queries a descriptor of this texture.

Remarks

This function only queries the following attributes:

- `type`
- `format`
- `extent`
- `arrayLayers`
- `mipLevels`
- `samples`

All other attributes (such as `bindFlags`, `miscFlags` etc.) cannot be queried by this function. Those attributes are either set to zero (for flags) or the default value specified in TextureDescriptor is used. If only the texture format is required, use `GetFormat` instead.

See also

TextureDescriptor
GetFormat
Buffer::GetDesc

10.117.3.3 virtual Format LLGL::Texture::GetFormat ( ) const [pure virtual]

10.117.3.4 std::uint32_t LLGL::Texture::GetMemoryFootprint ( ) const

Returns the memory footprint (in bytes) of the entire texture.

Remarks

This function uses the member `GetDesc` and the global `GetMemoryFootprint` functions.

See also

LLGL::GetMemoryFootprint(const Format, std::uint32_t)
LLGL::GetMemoryFootprint(const TextureType, const Format, const Extent3D&, const TextureSubresource&)
GetDesc

Todo Maybe make this virtual to allow memory alignment (especially for D3D12).

10.117.3.5 std::uint32_t LLGL::Texture::GetMemoryFootprint (const Extent3D & *extent*, const TextureSubresource & *subresource*) const

Returns the memory footprint (in bytes) of the specified subresource and extent for this texture.

Remarks

This function uses the member `GetFormat` and the global `GetMemoryFootprint` functions.

See also

LLGL::GetMemoryFootprint(const Format, std::uint32_t)
LLGL::GetMemoryFootprint(const TextureType, const Format, const Extent3D&, const TextureSubresource&)
GetFormat

Todo Maybe make this virtual to allow memory alignment (especially for D3D12).

10.117.3.6 virtual Extent3D LLGL::Texture::GetMipExtent (std::uint32_t *mipLevel*) const [pure virtual]

Returns the texture extent for the specified MIP-level. This also includes the number of array layers.

Parameters

in	*mipLevel*	Specifies the MIP-map level to query from. The first and largest MIP-map is level zero. If this level is greater than or equal to the maxmimum number of MIP-maps for this texture, the return value is undefined (i.e. depends on the render system).

Remarks

For 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.
For cube textures and cube array textures (i.e. texture type TextureType::TextureCube and TextureType::↩ TextureCubeArray), the depth extent will be a multiple of 6. This is in contrast to the other handling of cube array layers, because this function determines the actual buffer extent of the hardware texture.

See also

TextureDescriptor::mipLevels
CommandBuffer::GenerateMips
NumMipDimensions
LLGL::GetMipExtent

10.117.3.7 ResourceType LLGL::Texture::GetResourceType ( ) const [final], [override], [virtual]

Returns ResourceType::Texture.

Implements LLGL::Resource.

10.117.3.8 TextureType LLGL::Texture::GetType ( ) const [inline]

Returns the type of this texture.

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

- Texture.h

10.118 LLGL::TextureDescriptor Struct Reference

Texture descriptor structure.

```
#include <TextureFlags.h>
```

Public Attributes

- TextureType type = TextureType::Texture2D

  Hardware texture type. By default TextureType::Texture2D.
- long bindFlags = (BindFlags::Sampled | BindFlags::ColorAttachment)

  These flags describe to which resource slots and render target attachments the texture can be bound. By default BindFlags::Sampled and BindFlags::ColorAttachment.
- long miscFlags = (MiscFlags::FixedSamples | MiscFlags::GenerateMips)

  Miscellaneous texture flags. By default MiscFlags::FixedSamples and MiscFlags::GenerateMips.
- Format format = Format::RGBA8UNorm

  Hardware texture format. By default Format::RGBA8UNorm.
- Extent3D extent = { 1, 1, 1 }

  Size of the texture (excluding the number of array layers). By default (1, 1, 1).
- std::uint32_t arrayLayers = 1

  Number of array layers. By default 1.
- std::uint32_t mipLevels = 0

  Number of MIP-map levels. By default 0.
- std::uint32_t samples = 1

  Number of samples per texel. By default 1.
- ClearValue clearValue

  Specifies a clear value to initialize the texture with, if no initial image data is provided.

10.118.1 Detailed Description

Texture descriptor structure.

Remarks

Contains all information about type, format, and dimension to create a texture resource.

See also

RenderSystem::CreateTexture

10.118.2 Member Data Documentation

10.118.2.1 std::uint32_t LLGL::TextureDescriptor::arrayLayers = 1

Number of array layers. By default 1.

Remarks

For array textures and cube textures (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCube, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray), this **must** be greater than or equal to 1.
For cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray), this **must** be a multiple of 6 and greater than zero (one array layer for each cube face).
For all other texture types, this **must** be 1.
The index offsets for each cube face are as follows:

- `X+` direction has index offset 0.
- `X-` direction has index offset 1.
- `Y+` direction has index offset 2.
- `Y-` direction has index offset 3.
- `Z+` direction has index offset 4.
- `Z-` direction has index offset 5.

See also

IsArrayTexture
IsCubeTexture
RenderingLimits::maxTextureArrayLayers

10.118.2.2 long LLGL::TextureDescriptor::bindFlags = (BindFlags::Sampled | BindFlags::ColorAttachment)

These flags describe to which resource slots and render target attachments the texture can be bound. By default BindFlags::Sampled and BindFlags::ColorAttachment.

Remarks

When the texture will be bound as a color attachment to a render target for instance, the BindFlags::ColorAttachment flag is required.

See also

BindFlags

10.118.2.3 ClearValue LLGL::TextureDescriptor::clearValue

Specifies a clear value to initialize the texture with, if no initial image data is provided.

Remarks

The initial texture data is only determined by this attribute if the `imageDesc` parameter of RenderSystem::CreateTexture is null and the MiscFlags::NoInitialData bit is **not** set in the `miscFlags` attribute. In either case, this value may be used by the renderer API as a hint which clear value the resource is optimized for (especially for Direct3D 12).

See also

RenderSystem::CreateTexture
TextureDescriptor::miscFlags

10.118.2.4 Extent3D LLGL::TextureDescriptor::extent = { 1, 1, 1 }

Size of the texture (excluding the number of array layers). By default (1, 1, 1).

Remarks

The `height` component is only used for 2D, 3D, and Cube textures (i.e. TextureType::Texture2D, TextureType::Texture2DArray, TextureType::Texture3D, TextureType::TextureCube, TextureType::TextureCubeArray, TextureType::Texture2DMS, and TextureType::Texture2DMSArray).
The `depth` component is only used for 3D textures (i.e. TextureType::Texture3D).
The `width` and `height` components must be equal for cube textures (i.e. TextureType::TextureCube and TextureType::TextureCubeArray).

See also

IsCubeTexture

10.118.2.5 Format LLGL::TextureDescriptor::format = Format::RGBA8UNorm

Hardware texture format. By default Format::RGBA8UNorm.

See also

RenderingCapabilities::textureFormats

10.118.2.6 std::uint32_t LLGL::TextureDescriptor::mipLevels = 0

Number of MIP-map levels. By default 0.

Remarks

If this is 0, the number of MIP-map levels will be determined automatically by the texture type and extent for a full MIP-chain.
If this is 1, no MIP-mapping is used for this texture and it has only a single MIP-map level.
For multi-sampled textures (i.e. TextureType::Texture2DMS, TextureType::Texture2DMSArray), this value must be either 0 or 1.

See also

NumMipLevels
CommandBuffer::GenerateMips

10.118.2.7 long LLGL::TextureDescriptor::miscFlags = (MiscFlags::FixedSamples | MiscFlags::GenerateMips)

Miscellaneous texture flags. By default MiscFlags::FixedSamples and MiscFlags::GenerateMips.

Remarks

This can be used as a hint for the renderer how frequently the texture will be updated, or whether a multi-sampled texture has fixed sample locations.

See also

MiscFlags

10.118.2.8 std::uint32_t LLGL::TextureDescriptor::samples = 1

Number of samples per texel. By default 1.

Remarks

This is only used for multi-sampled textures (i.e. TextureType::Texture2DMS and TextureType::Texture2DMSArray).

See also

IsMultiSampleTexture

10.118.2.9 TextureType LLGL::TextureDescriptor::type = TextureType::Texture2D

Hardware texture type. By default TextureType::Texture2D.

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

- TextureFlags.h

10.119 LLGL::TextureLocation Struct Reference

Texture location structure: MIP-map level and offset.

```
#include <TextureFlags.h>
```

Public Member Functions

- TextureLocation ()=default
- TextureLocation (const TextureLocation &)=default
- TextureLocation (const Offset3D &offset, std::uint32_t arrayLayer=0, std::uint32_t mipLevel=0)

  Constructor to initialize all attributes.

Public Attributes

- Offset3D offset

  Zero-based offset within the texture data.
- std::uint32_t arrayLayer = 0

  Zero-based array layer index.
- std::uint32_t mipLevel = 0

  MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.

10.119.1 Detailed Description

Texture location structure: MIP-map level and offset.

Remarks

This is used to specifiy the source and destination location of a texture copy operation.

See also

CommandBuffer::CopyTexture
TextureRegion

10.119.2 Constructor & Destructor Documentation

10.119.2.1 LLGL::TextureLocation::TextureLocation ( ) [default]

10.119.2.2 LLGL::TextureLocation::TextureLocation (const TextureLocation &) [default]

10.119.2.3 LLGL::TextureLocation::TextureLocation (const Offset3D & *offset*, std::uint32_t *arrayLayer* = 0, std::uint32_t *mipLevel* = 0) [inline]

Constructor to initialize all attributes.

10.119.3 Member Data Documentation

10.119.3.1 std::uint32_t LLGL::TextureLocation::arrayLayer = 0

Zero-based array layer index.

Remarks

Only used by array texture types (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, Texture↩Type::TextureCubeArray, and TextureType::Texture2DMSArray).
This field is ignored by all other texture types.

See also

TextureDescriptor::arrayLayers

10.119.3.2 std::uint32_t LLGL::TextureLocation::mipLevel = 0

MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.

See also

TextureDescriptor::mipLevels

10.119.3.3 Offset3D LLGL::TextureLocation::offset

Zero-based offset within the texture data.

Remarks

Any component of this field that is not meant for the respective texture type is ignored. All other components must be greater than or equal to zero.

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

- TextureFlags.h

10.120 LLGL::TextureRegion Struct Reference

Texture region structure: Subresource (MIP-map level and array layer range), offset, and extent.

```
#include <TextureFlags.h>
```

Public Member Functions

- TextureRegion ()=default
- TextureRegion (const TextureRegion &)=default
- TextureRegion (const Offset3D &offset, const Extent3D &extent)

  Constructor to initialize offset and extent only.
- TextureRegion (const TextureSubresource &subresource, const Offset3D &offset, const Extent3D &extent)

  Constructor to initialize all members.

Public Attributes

- TextureSubresource subresource

  Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.
- Offset3D offset

  Zero-based offset within the texture data.
- Extent3D extent

  Extent of the sub texture region.

10.120.1 Detailed Description

Texture region structure: Subresource (MIP-map level and array layer range), offset, and extent.

Remarks

This is used to write (or partially write) and read (or partially read) the image data of a **single** texture MIP-map level.

See also

RenderSystem::WriteTexture
RenderSystem::ReadTexture
TextureLocation

10.120.2 Constructor & Destructor Documentation

10.120.2.1 LLGL::TextureRegion::TextureRegion ( ) [default]

10.120.2.2 LLGL::TextureRegion::TextureRegion (const TextureRegion &) [default]

10.120.2.3 LLGL::TextureRegion::TextureRegion (const Offset3D & *offset,* const Extent3D & *extent*) [inline]

Constructor to initialize offset and extent only.

10.120.2.4 LLGL::TextureRegion::TextureRegion (const TextureSubresource & *subresource,* const Offset3D & *offset,* const Extent3D & *extent*) [inline]

Constructor to initialize all members.

10.120.3 Member Data Documentation

10.120.3.1 Extent3D LLGL::TextureRegion::extent

Extent of the sub texture region.

See also

TextureDescriptor::extent

10.120.3.2 Offset3D LLGL::TextureRegion::offset

Zero-based offset within the texture data.

Remarks

Any component of this field that is not meant for the respective texture type is ignored. All other components must be greater than or equal to zero.

10.120.3.3 TextureSubresource LLGL::TextureRegion::subresource

Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.

Remarks

For texture regions, `numMipLevels` of the TextureSubresource structure must always be 1, i.e. texture regions can only select a single MIP-map at a time.

See also

TextureSubresource::numMipLevels

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

- TextureFlags.h

10.121 LLGL::TextureSubresource Struct Reference

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

```
#include <TextureFlags.h>
```

Public Member Functions

- TextureSubresource ()=default
- TextureSubresource (const TextureSubresource &)=default
- TextureSubresource (std::uint32_t baseArrayLayer, std::uint32_t baseMipLevel)

  Constructor to initialize base MIP-map level and base array layer only.
- TextureSubresource (std::uint32_t baseArrayLayer, std::uint32_t numArrayLayers, std::uint32_t baseMip↩Level, std::uint32_t numMipLevels)

  Constructor to initialize all attributes.

Public Attributes

- std::uint32_t baseArrayLayer = 0

  Zero-based index of the first array layer. By default 0.
- std::uint32_t numArrayLayers = 1

  Number of array layers. By default 1.
- std::uint32_t baseMipLevel = 0

  MIP-map level for the sub-texture, where 0 is the base texture, and N > 0 is the N-th MIP-map level. By default 0.
- std::uint32_t numMipLevels = 1

  Number of MIP-map levels. By default 1.

10.121.1 Detailed Description

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

Remarks

The default values refer to the first array layer and the first MIP-map level.

See also

TextureRegion::subresource
TextureViewDescriptor::subresource

10.121.2 Constructor & Destructor Documentation

10.121.2.1 LLGL::TextureSubresource::TextureSubresource ( ) [default]

10.121.2.2 LLGL::TextureSubresource::TextureSubresource (const TextureSubresource &) [default]

10.121.2.3 LLGL::TextureSubresource::TextureSubresource (std::uint32_t *baseArrayLayer*, std::uint32_t *baseMipLevel*) [inline]

Constructor to initialize base MIP-map level and base array layer only.

10.121.2.4 LLGL::TextureSubresource::TextureSubresource (std::uint32_t *baseArrayLayer,* std::uint32_t *numArrayLayers,* std::uint32_t *baseMipLevel,* std::uint32_t *numMipLevels*) [inline]

Constructor to initialize all attributes.

10.121.3 Member Data Documentation

10.121.3.1 std::uint32_t LLGL::TextureSubresource::baseArrayLayer = 0

Zero-based index of the first array layer. By default 0.

Remarks

Only used by array texture types (i.e. TextureType::Texture1DArray, TextureType::Texture2DArray, TextureType::TextureCubeArray, and TextureType::Texture2DMSArray).
This field is ignored by all other texture types.

See also

TextureDescriptor::arrayLayers

10.121.3.2 std::uint32_t LLGL::TextureSubresource::baseMipLevel = 0

MIP-map level for the sub-texture, where 0 is the base texture, and N $>$ 0 is the N-th MIP-map level. By default 0.

See also

TextureDescriptor::mipLevels

10.121.3.3 std::uint32_t LLGL::TextureSubresource::numArrayLayers = 1

Number of array layers. By default 1.

Remarks

Must be greater than zero.

See also

TextureDescriptor::arrayLayers

10.121.3.4 std::uint32_t LLGL::TextureSubresource::numMipLevels = 1

Number of MIP-map levels. By default 1.

Remarks

Must be greater than zero.

See also

TextureDescriptor::mipLevels

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

- TextureFlags.h

10.122 LLGL::TextureSwizzleRGBA Struct Reference

Texture component swizzle structure for red, green, blue, and alpha components.

```
#include <TextureFlags.h>
```

Public Attributes

- TextureSwizzle r = TextureSwizzle::Red

  Red component swizzle. By default TextureSwizzle::Red.
- TextureSwizzle g = TextureSwizzle::Green

  Green component swizzle. By default TextureSwizzle::Green.
- TextureSwizzle b = TextureSwizzle::Blue

  Blue component swizzle. By default TextureSwizzle::Blue.
- TextureSwizzle a = TextureSwizzle::Alpha

  Alpha component swizzle. By default TextureSwizzle::Alpha.

10.122.1 Detailed Description

Texture component swizzle structure for red, green, blue, and alpha components.

Remarks

Can be used to change the order of texel components independently of a shader.

See also

TextureViewDescriptor::swizzle

10.122.2 Member Data Documentation

10.122.2.1 TextureSwizzle LLGL::TextureSwizzleRGBA::a = TextureSwizzle::Alpha

Alpha component swizzle. By default TextureSwizzle::Alpha.

10.122.2.2 TextureSwizzle LLGL::TextureSwizzleRGBA::b = TextureSwizzle::Blue

Blue component swizzle. By default TextureSwizzle::Blue.

10.122.2.3 TextureSwizzle LLGL::TextureSwizzleRGBA::g = TextureSwizzle::Green

Green component swizzle. By default TextureSwizzle::Green.

10.122.2.4 TextureSwizzle LLGL::TextureSwizzleRGBA::r = TextureSwizzle::Red

Red component swizzle. By default TextureSwizzle::Red.

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

- TextureFlags.h

10.123 LLGL::TextureViewDescriptor Struct Reference

Texture view descriptor structure.

```
#include <TextureFlags.h>
```

Public Attributes

- TextureType type = TextureType::Texture2D

  Hardware texture type. By default TextureType::Texture2D.
- Format format = Format::RGBA8UNorm

  Hardware texture format. By default Format::RGBA8UNorm.
- TextureSubresource subresource

  Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.
- TextureSwizzleRGBA swizzle

  Specifies the color component mapping. Each component is mapped to its identity by default.

10.123.1 Detailed Description

Texture view descriptor structure.

Remarks

Contains all information about type, format, and dimension to create a texture view that shares the image data of another texture resource.

See also

ResourceViewDescriptor::textureView
RenderingFeatures::hasTextureViews

10.123.2 Member Data Documentation

10.123.2.1 Format LLGL::TextureViewDescriptor::format = Format::RGBA8UNorm

Hardware texture format. By default Format::RGBA8UNorm.

Remarks

The format of the shared texture and the texture-view must be in the same format class:

Class	Compatible texture formats
128 Bits	Format::RGBA32UInt, Format::RGBA32SInt, Format::RGBA32Float
96 Bits	Format::RGB32UInt, Format::RGB32SInt, Format::RGB32Float
64 Bits	Format::RG32UInt, Format::RG32SInt, Format::RG32Float, Format::RGBA16UNorm, Format::RGBA16SNorm, Format::RGBA16UInt, Format::RGBA16SInt, Format::RGBA16Float
48 Bits	Format::RGB16UNorm, Format::RGB16SNorm, Format::RGB16UInt, Format::RGB16SInt, Format::RGB16Float
32 Bits	Format::RG16UNorm, Format::RG16SNorm, Format::RG16UInt, Format::RG16SInt, Format↩ ::RG16Float, Format::RGBA8UNorm, Format::RGBA8SNorm, Format::RGBA8UInt, Format↩ ::RGBA8SInt
24 Bits	Format::RGB8UNorm, Format::RGB8SNorm, Format::RGB8UInt, Format::RGB8SInt
16 Bits	Format::R16UNorm, Format::R16SNorm, Format::R16UInt, Format::R16SInt, Format::R16↩ Float, Format::RG8UNorm, Format::RG8SNorm, Format::RG8UInt, Format::RG8SInt
8 Bits	Format::R8UNorm, Format::R8SNorm, Format::R8UInt, Format::R8SInt

10.123.2.2 TextureSubresource LLGL::TextureViewDescriptor::subresource

Specifies the texture subresource, i.e. MIP-map level and array layer range. By default only the first MIP-map level and first array layer is addressed.

Remarks

For texture subresources that are bound with the binding flag BindFlags::Storage, `numMipLevels` **must** be 1.

10.123.2.3 TextureSwizzleRGBA LLGL::TextureViewDescriptor::swizzle

Specifies the color component mapping. Each component is mapped to its identity by default.

Remarks

If texture swizzling is not supported, this must be equal to the default value.

Note

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

See also

RenderingFeatures::hasTextureViewSwizzle
IsTextureSwizzleIdentity

10.123.2.4 TextureType LLGL::TextureViewDescriptor::type = TextureType::Texture2D

Hardware texture type. By default TextureType::Texture2D.

Remarks

The types of a shared texture can be mapped to the following type of texture-views:

Shared texture type	Compatible texture view types
TextureType::Texture1D	TextureType::Texture1D, TextureType::Texture1DArray
TextureType::Texture2D	TextureType::Texture2D, TextureType::Texture2DArray
TextureType::Texture3D	TextureType::Texture3D
TextureType::TextureCube	TextureType::Texture2D, TextureType::Texture2DArray, TextureType↩ ::TextureCube, TextureType::TextureCubeArray
TextureType::Texture1DArray	TextureType::Texture1D, TextureType::Texture1DArray
TextureType::Texture2DArray	TextureType::Texture2D, TextureType::Texture2DArray
TextureType::TextureCubeArray	TextureType::Texture2D, TextureType::Texture2DArray, TextureType↩ ::TextureCube, TextureType::TextureCubeArray
TextureType::Texture2DMS	TextureType::Texture2DMS, TextureType::Texture2DMSArray
TextureType::Texture2DMSArray	TextureType::Texture2DMS, TextureType::Texture2DMSArray

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

- TextureFlags.h

10.124 LLGL::UninitializeTag Struct Reference

Common uninitialize tag.

```
#include <Tags.h>
```

10.124.1 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 { LLGL::UninitializeTag{} };

// Explicitly uninitialized color elements in a container.
std::vector<LLGL::ColorRGBAf> color;
color.resize(1024, LLGL::ColorRGBAf { LLGL::UninitializeTag{} });
```

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

- Tags.h

10.125 LLGL::UTF8String Class Reference

Container class for UTF-8 encoded strings.

```
#include <UTF8String.h>
```

Public Types

- using value_type = char
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using reference = value_type &
- using const_reference = const value_type &
- using pointer = value_type *
- using const_pointer = const value_type *
- using iterator = value_type *
- using const_iterator = const value_type *
- using reverse_iterator = std::reverse_iterator< iterator >
- using const_reverse_iterator = std::reverse_iterator< const_iterator >

Public Member Functions

- UTF8String ()

  Initialize an empty string.
- UTF8String (const UTF8String &rhs)

  Initialies the UTF-8 string with a copy of the specified string.
- UTF8String (UTF8String &&rhs)

  Takes the ownership of the specified UTF-8 string.
- UTF8String (const StringView &str)

  Initializes the UTF-8 string with a copy of the specified string view.
- UTF8String (const WStringView &str)

  Initializes the UTF-8 string with a UTF-8 encoded conversion of the specified wide string view.

- UTF8String (const char *str)

  Initializes the UTF-8 string with a copy of the specified null-terminated string.
- UTF8String (const wchar_t *str)

  Initializes the UTF-8 string with a UTF-8 encoded conversion of the specified null-terminated wide string.
- template<template< class, class, class > class TString, class TChar , class Traits , class Allocator > UTF8String (const TString< TChar, Traits, Allocator > &str)

  Initializes the UTF-8 string with a copy of another templated string class. This would usually be std::basic_string.
- bool empty () const
- size_type size () const
- size_type length () const
- size_type capacity () const
- const_pointer data () const
- const_pointer c_str () const
- const_reference at (size_type pos) const
- const_reference front () const
- const_reference back () const
- const_iterator begin () const
- const_iterator cbegin () const
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const
- const_iterator end () const
- const_iterator cend () const
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const
- void clear ()
- int compare (const StringView &str) const
- int compare (size_type pos1, size_type count1, const StringView &str) const
- int compare (size_type pos1, size_type count1, const StringView &str, size_type pos2, size_type count2=npos) const
- int compare (const WStringView &str) const
- int compare (size_type pos1, size_type count1, const WStringView &str) const
- int compare (size_type pos1, size_type count1, const WStringView &str, size_type pos2, size_type count2=npos) const
- SmallVector< wchar_t > ToWCharArray () const
- UTF8String & operator= (const UTF8String &rhs)
- UTF8String & operator= (UTF8String &&rhs)
- UTF8String & operator+= (const UTF8String &rhs)
- UTF8String & operator+= (const StringView &rhs)
- UTF8String & operator+= (const WStringView &rhs)
- UTF8String & operator+= (const char *rhs)
- UTF8String & operator+= (const wchar_t *rhs)
- UTF8String & operator+= (char chr)
- UTF8String & operator+= (wchar_t chr)
- const_reference operator[ ] (size_type pos) const
- operator const_pointer () const

  Conversion operator to a null-terminated string.
- operator StringView () const

  Conversion operator to a string view.

Static Public Attributes

- static const size_type npos = -1

10.125.1 Detailed Description

Container class for UTF-8 encoded strings.

Remarks

This class converts between `char` and `wchar_t` strings automatically, but stores strings always as UTF-8 encoded `char` strings.

10.125.2 Member Typedef Documentation

10.125.2.1 using LLGL::UTF8String::const_iterator = const value_type*

10.125.2.2 using LLGL::UTF8String::const_pointer = const value_type*

10.125.2.3 using LLGL::UTF8String::const_reference = const value_type&

10.125.2.4 using LLGL::UTF8String::const_reverse_iterator = std::reverse_iterator<const_iterator>

10.125.2.5 using LLGL::UTF8String::difference_type = std::ptrdiff_t

10.125.2.6 using LLGL::UTF8String::iterator = value_type*

10.125.2.7 using LLGL::UTF8String::pointer = value_type*

10.125.2.8 using LLGL::UTF8String::reference = value_type&

10.125.2.9 using LLGL::UTF8String::reverse_iterator = std::reverse_iterator<iterator>

10.125.2.10 using LLGL::UTF8String::size_type = std::size_t

10.125.2.11 using LLGL::UTF8String::value_type = char

10.125.3 Constructor & Destructor Documentation

10.125.3.1 LLGL::UTF8String::UTF8String ( )

Initialize an empty string.

10.125.3.2 LLGL::UTF8String::UTF8String (const UTF8String & *rhs*)

Initialies the UTF-8 string with a copy of the specified string.

10.125.3.3 LLGL::UTF8String::UTF8String (UTF8String && *rhs*)

Takes the ownership of the specified UTF-8 string.

10.125.3.4 LLGL::UTF8String::UTF8String (const StringView & *str*)

Initializes the UTF-8 string with a copy of the specified string view.

10.125.3.5 LLGL::UTF8String::UTF8String (const WStringView & *str*)

Initializes the UTF-8 string with a UTF-8 encoded conversion of the specified wide string view.

10.125.3.6 LLGL::UTF8String::UTF8String (const char * *str*)

Initializes the UTF-8 string with a copy of the specified null-terminated string.

10.125.3.7 LLGL::UTF8String::UTF8String (const wchar_t * *str*)

Initializes the UTF-8 string with a UTF-8 encoded conversion of the specified null-terminated wide string.

10.125.3.8 template<template< class, class, class > class TString, class TChar , class Traits , class Allocator > LLGL::UTF8String::UTF8String (const TString< TChar, Traits, Allocator > & *str*) [inline]

Initializes the UTF-8 string with a copy of another templated string class. This would usually be std::basic_string.

10.125.4 Member Function Documentation

10.125.4.1 const_reference LLGL::UTF8String::at (size_type *pos*) const [inline]

10.125.4.2 const_reference LLGL::UTF8String::back ( ) const [inline]

10.125.4.3 const_iterator LLGL::UTF8String::begin ( ) const [inline]

10.125.4.4 const_pointer LLGL::UTF8String::c_str ( ) const [inline]

10.125.4.5 size_type LLGL::UTF8String::capacity ( ) const [inline]

10.125.4.6 const_iterator LLGL::UTF8String::cbegin ( ) const [inline]

10.125.4.7 const_iterator LLGL::UTF8String::cend ( ) const [inline]

10.125.4.8 void LLGL::UTF8String::clear ( )

10.125.4.9 int LLGL::UTF8String::compare (const StringView & *str*) const

10.125.4.10 int LLGL::UTF8String::compare (size_type *pos1*, size_type *count1*, const StringView & *str*) const

10.125.4.11 int LLGL::UTF8String::compare (size_type *pos1,* size_type *count1,* const StringView & *str,* size_type *pos2,* size_type *count2* = npos) const

10.125.4.12 int LLGL::UTF8String::compare (const WStringView & *str*) const

10.125.4.13 int LLGL::UTF8String::compare (size_type *pos1,* size_type *count1,* const WStringView & *str*) const

10.125.4.14 int LLGL::UTF8String::compare (size_type *pos1,* size_type *count1,* const WStringView & *str,* size_type *pos2,* size_type *count2* = npos) const

10.125.4.15 const_reverse_iterator LLGL::UTF8String::crbegin ( ) const [inline]

10.125.4.16 const_reverse_iterator LLGL::UTF8String::crend ( ) const [inline]

10.125.4.17 const_pointer LLGL::UTF8String::data ( ) const [inline]

10.125.4.18 bool LLGL::UTF8String::empty ( ) const [inline]

10.125.4.19 const_iterator LLGL::UTF8String::end ( ) const [inline]

10.125.4.20 const_reference LLGL::UTF8String::front ( ) const [inline]

10.125.4.21 size_type LLGL::UTF8String::length ( ) const [inline]

10.125.4.22 LLGL::UTF8String::operator const_pointer ( ) const [inline]

Conversion operator to a null-terminated string.

10.125.4.23 LLGL::UTF8String::operator StringView ( ) const [inline]

Conversion operator to a string view.

10.125.4.24 UTF8String& LLGL::UTF8String::operator+= (const UTF8String & *rhs*)

10.125.4.25 UTF8String& LLGL::UTF8String::operator+= (const StringView & *rhs*)

10.125.4.26 UTF8String& LLGL::UTF8String::operator+= (const WStringView & *rhs*)

10.125.4.27 UTF8String& LLGL::UTF8String::operator+= (const char * *rhs*)

10.125.4.28 UTF8String& LLGL::UTF8String::operator+= (const wchar_t * *rhs*)

10.125.4.29 UTF8String& LLGL::UTF8String::operator+= (char *chr*)

10.125.4.30 UTF8String& LLGL::UTF8String::operator+= (wchar_t *chr*)

10.125.4.31 UTF8String& LLGL::UTF8String::operator= (const UTF8String & *rhs*)

10.125.4.32 UTF8String& LLGL::UTF8String::operator= (UTF8String && *rhs*)

10.125.4.33 const_reference LLGL::UTF8String::operator[ ] (size_type *pos*) const [inline]

10.125.4.34 const_reverse_iterator LLGL::UTF8String::rbegin ( ) const [inline]

10.125.4.35 const_reverse_iterator LLGL::UTF8String::rend ( ) const [inline]

10.125.4.36 size_type LLGL::UTF8String::size ( ) const [inline]

10.125.4.37 SmallVector<wchar_t> LLGL::UTF8String::ToWCharArray ( ) const

10.125.5 Member Data Documentation

10.125.5.1 const size_type LLGL::UTF8String::npos = -1 [static]

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

- UTF8String.h

10.126 LLGL::VertexAttribute Struct Reference

Vertex input/output attribute structure.

```
#include <VertexAttribute.h>
```

Public Member Functions

- VertexAttribute ()=default
- VertexAttribute (const VertexAttribute &)=default
- VertexAttribute & operator= (const VertexAttribute &)=default
- VertexAttribute (const StringView &name, const Format format, std::uint32_t location=0, std::uint32_t instanceDivisor=0, const SystemValue systemValue=SystemValue::Undefined)

  Constructor for minimal vertex attribute information and system value semantics, e.g. `SV_VertexID` (HLSL) or `gl_VertexID` (GLSL).
- VertexAttribute (const StringView &semanticName, std::uint32_t semanticIndex, const Format format, std::uint32_t location=0, std::uint32_t instanceDivisor=0)

  Constructor for basic vertex attribute information.
- VertexAttribute (const StringView &name, const Format format, std::uint32_t location, std::uint32_t offset, std::uint32_t stride, std::uint32_t slot=0, std::uint32_t instanceDivisor=0)

  Constructor for common vertex attribute information.
- VertexAttribute (const StringView &semanticName, std::uint32_t semanticIndex, const Format format, std::uint32_t location, std::uint32_t offset, std::uint32_t stride, std::uint32_t slot=0, std::uint32_t instanceDivisor=0)

  Constructor for the most vertex attribute information, including semantic index.
- std::uint32_t GetSize () const

  Returns the size (in bytes) which is required for this vertex attribute, or zero if the format is not a valid vertex format.

Public Attributes

- UTF8String 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 location = 0

  Vertex attribute location (only for OpenGL, Vulkan, Metal) or stream-output number (only for Direct3D).
- std::uint32_t semanticIndex = 0

  Semantic index for HLSL.
- SystemValue systemValue = SystemValue::Undefined

  Specifies the system value type for this vertex attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.
- std::uint32_t slot = 0

  Vertex buffer binding slot. By default 0.
- std::uint32_t offset = 0

  Byte offset within each vertex and each buffer for input attributes, or component offset for output attributes. By default 0.
- std::uint32_t stride = 0

  Specifies the vertex data stride which describes the byte offset between consecutive vertices.
- std::uint32_t instanceDivisor = 0

  Instance data divisor (or instance data step rate).

10.126.1 Detailed Description

Vertex input/output attribute structure.

Remarks

For attributes within the same vertex buffer, the following members **must** have the same value:

- `slot`
- `stride`
- `instanceDivisor`

See also

VertexShaderAttributes::inputAttribs
VertexShaderAttributes::outputAttribs
BufferDescriptor::vertexAttribs
VertexFormat::attributes
FragmentAttribute

10.126.2 Constructor & Destructor Documentation

10.126.2.1 LLGL::VertexAttribute::VertexAttribute ( ) [default]

10.126.2.2 LLGL::VertexAttribute::VertexAttribute (const VertexAttribute &) [default]

10.126.2.3 LLGL::VertexAttribute::VertexAttribute (const StringView & *name*, const Format *format*, std::uint32_t *location* = 0, std::uint32_t *instanceDivisor* = 0, const SystemValue *systemValue* = SystemValue::Undefined)

Constructor for minimal vertex attribute information and system value semantics, e.g. `SV_VertexID` (HLSL) or `gl_VertexID` (GLSL).

10.126.2.4 LLGL::VertexAttribute::VertexAttribute (const StringView & *semanticName*, std::uint32_t *semanticIndex*, const Format *format*, std::uint32_t *location* = 0, std::uint32_t *instanceDivisor* = 0)

Constructor for basic vertex attribute information.

10.126.2.5 LLGL::VertexAttribute::VertexAttribute (const StringView & *name*, const Format *format*, std::uint32_t *location*, std::uint32_t *offset*, std::uint32_t *stride*, std::uint32_t *slot* = 0, std::uint32_t *instanceDivisor* = 0)

Constructor for common vertex attribute information.

10.126.2.6 LLGL::VertexAttribute::VertexAttribute (const StringView & *semanticName*, std::uint32_t *semanticIndex*, const Format *format*, std::uint32_t *location*, std::uint32_t *offset*, std::uint32_t *stride*, std::uint32_t *slot* = 0, std::uint32_t *instanceDivisor* = 0)

Constructor for the most vertex attribute information, including semantic index.

10.126.3 Member Function Documentation

10.126.3.1 std::uint32_t LLGL::VertexAttribute::GetSize ( ) const

Returns the size (in bytes) which is required for this vertex attribute, or zero if the format is not a valid vertex format.

See also

FormatAttributes::bitSize
FormatFlags::SupportsVertex
Format

10.126.3.2 VertexAttribute& LLGL::VertexAttribute::operator= (const VertexAttribute &) [default]

10.126.4 Member Data Documentation

10.126.4.1 Format LLGL::VertexAttribute::format = Format::RGBA32Float

Vertex attribute format. By default Format::RGBA32Float.

Remarks

Not all hardware formats are allowed for vertex attributes. In particular, depth-stencil formats and compressed formats are not allowed. To specify a vertex attribute of a matrix type, multiple attributes with ascending semantic indices must be used. Here is an example of a 4x4 matrix:

```
myVertexFormat.AppendAttribute({ "myMatrix4x4", 0, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 1, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 2, LLGL::Format::RGBA32Float });
myVertexFormat.AppendAttribute({ "myMatrix4x4", 3, LLGL::Format::RGBA32Float });
```

Here is an example of a 2x2 matrix:

```
myVertexFormat.AppendAttribute({ "myMatrix2x2", 0, LLGL::Format::RG32Float });
myVertexFormat.AppendAttribute({ "myMatrix2x2", 1, LLGL::Format::RG32Float });
```

10.126.4.2 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.

Note

For Vulkan, this must only be 0 or 1.

10.126.4.3 std::uint32_t LLGL::VertexAttribute::location = 0

Vertex attribute location (only for OpenGL, Vulkan, Metal) or stream-output number (only for Direct3D).

Remarks

This is only required for OpenGL, Vulkan, and Metal. For Direct3D, this is ignored and instead `semantic↩Index` is used.
The following example shows GLSL attribute locations from 0 to 4 inclusive:

```
layout(location = 0) in vec4 vertexPosition;   // location 0
layout(location = 1) in mat4 projectionMatrix; // location 1...4
```

The following example shows Metal attribute locations from 0 to 4 inclusive:

```
struct MyVertexInput
{
    float4 vertexPosition    [[attribute(0)]];
    float4 projectionMatrix0 [[attribute(1)]];
    float4 projectionMatrix1 [[attribute(2)]];
    float4 projectionMatrix2 [[attribute(3)]];
    float4 projectionMatrix3 [[attribute(4)]];
};
```

10.126.4.4 UTF8String LLGL::VertexAttribute::name

Vertex attribute name (for GLSL) or semantic name (for HLSL).

10.126.4.5 std::uint32_t LLGL::VertexAttribute::offset = 0

Byte offset within each vertex and each buffer for input attributes, or component offset for output attributes. By default 0.

Remarks

For vertex input attributes, this offset specifies the byte aligned offset within each vertex buffer.
For stream-output attributes, this offset specifies the first component that is to be written and must be either 0, 1, 2, or 3.

10.126.4.6 std::uint32_t LLGL::VertexAttribute::semanticIndex = 0

Semantic index for HLSL.

Remarks

This is only required for Direct3D when a semantic name is used multiple times. This happens when a matrix type is distributed over multiple vector attributes.
The following example uses semantic names `POS0`, `MATRIX0`, `MATRIX1`, `MATRIX2`, `MATRIX3`:

```
struct MyVertexInput
{
    float4   vertexPosition   : POS;
    float4x4 projectionMatrix : MATRIX;
};
```

10.126.4.7 std::uint32_t LLGL::VertexAttribute::slot = 0

Vertex buffer binding slot. By default 0.

Remarks

This is used when multiple vertex buffers are used simultaneously. This binding slot refers either to the input buffer indices (determined by `SetVertexBufferArray`), or stream-output buffer indices (determined by `BeginStreamOutput`).

Note

Only supported with: Direct3D 11, Direct3D 12, Vulkan, Metal.
For OpenGL, the vertex binding slots are automatically generated in ascending order and beginning with zero.

See also

CommandBuffer::SetVertexBufferArray
CommandBuffer::BeginStreamOutput

10.126.4.8 std::uint32_t LLGL::VertexAttribute::stride = 0

Specifies the vertex data stride which describes the byte offset between consecutive vertices.

10.126.4.9 SystemValue LLGL::VertexAttribute::systemValue = SystemValue::Undefined

Specifies the system value type for this vertex attribute or SystemValue::Undefined if this attribute is not a system value. By default SystemValue::Undefined.

Remarks

System value semantics are only used for shader code reflection. Examples of system value semantics are:

- Vertex ID: `SV_VertexID` (HLSL), `gl_VertexID` (GLSL), `gl_VertexIndex` (SPIR-V), `[[vertex_id]]` (Metal).
- Instance ID: `SV_InstanceID` (HLSL), `gl_InstanceID` (GLSL), `gl_InstanceIndex` (SPIR-V), `[[instance_id]]` (Metal).

See also

Shader::Reflect

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

- VertexAttribute.h

10.127 LLGL::VertexFormat Struct Reference

Utility structure to store vertex attributes.

```
#include <VertexFormat.h>
```

Public Member Functions

- void AppendAttribute (const VertexAttribute &attrib)

  Appends the specified vertex attribute to this vertex format.
- std::uint32_t GetStride () const

  Returns the stride (in bytes) of the first vertex.
- std::uint32_t GetStride (std::uint32_t slot) const

  Returns the stride (in bytes) of the first vertex with the specified buffer binding slot.
- void SetStride (std::uint32_t stride)

  Set the `stride` member for all vertex attributes to the specified value.
- void SetStride (std::uint32_t stride, std::uint32_t slot)

  Set the `stride` member for all vertex attributes with the specified buffer binding slot to the new value specified by `stride`.
- void SetSlot (std::uint32_t slot)

  Sets the `slot` member for all vertex attributes to the specified value.

Public Attributes

- std::vector< VertexAttribute > attributes

  Specifies the list of vertex attributes.

10.127.1 Detailed Description

Utility structure to store vertex attributes.

Remarks

A vertex format is required to describe how the vertex attributes are layed out inside a vertex buffer.

See also

BufferDescriptor::VertexBuffer::format

10.127.2 Member Function Documentation

10.127.2.1 void LLGL::VertexFormat::AppendAttribute (const VertexAttribute & *attrib*) [inline]

Appends the specified vertex attribute to this vertex format.

Parameters

in	*attrib*	Specifies the new attribute which is appended to this vertex format.

Remarks

This function modifies the `offset` member of specified attribute before adding it to the `attributes` list, and sets the `stride` member to the sum of the size of all attributes.

See also

VertexAttribute::offset

10.127.2.2 std::uint32_t LLGL::VertexFormat::GetStride ( ) const [inline]

Returns the stride (in bytes) of the first vertex.

Remarks

It is expected that all vertices with the same buffer binding slot have the same stride.

See also

VertexAttribute::GetSize

10.127.2.3 std::uint32_t LLGL::VertexFormat::GetStride (std::uint32_t *slot*) const [inline]

Returns the stride (in bytes) of the first vertex with the specified buffer binding slot.

Remarks

It is expected that all vertices with the same buffer binding slot have the same stride.

See also

VertexAttribute::GetSize

10.127.2.4 void LLGL::VertexFormat::SetSlot (std::uint32_t *slot*) [inline]

Sets the `slot` member for all vertex attributes to the specified value.

See also

VertexAttribute::slot

10.127.2.5 void LLGL::VertexFormat::SetStride (std::uint32_t *stride*) [inline]

Set the `stride` member for all vertex attributes to the specified value.

See also

VertexAttribute::stride

10.127.2.6 void LLGL::VertexFormat::SetStride (std::uint32_t *stride*, std::uint32_t *slot*) [inline]

Set the `stride` member for all vertex attributes with the specified buffer binding slot to the new value specified by `stride`.

See also

VertexAttribute::stride

10.127.3 Member Data Documentation

10.127.3.1 std::vector<VertexAttribute> LLGL::VertexFormat::attributes

Specifies the list of vertex attributes.

See also

AppendAttribute

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

- VertexFormat.h

10.128 LLGL::VertexShaderAttributes Struct Reference

Vertex (or geometry) shader specific structure.

```
#include <ShaderFlags.h>
```

Public Attributes

- std::vector< VertexAttribute > inputAttribs

  Vertex shader input attributes.
- std::vector< VertexAttribute > outputAttribs

  Vertex (or geometry) shader stream-output attributes.

10.128.1 Detailed Description

Vertex (or geometry) shader specific structure.

See also

ShaderDescriptor::vertex
ShaderReflection::vertex

10.128.2 Member Data Documentation

10.128.2.1 std::vector<VertexAttribute> LLGL::VertexShaderAttributes::inputAttribs

Vertex shader input attributes.

Remarks

All of these attributes must be contained in the `vertexAttribs` list of the vertex buffer that will be used in conjunction with the respective shader. In other words, a shader must not declare any vertex attributes that are not contained in the currently bound vertex buffer.

See also

BufferDescriptor::vertexAttribs

10.128.2.2 std::vector<VertexAttribute> LLGL::VertexShaderAttributes::outputAttribs

Vertex (or geometry) shader stream-output attributes.

Remarks

Some rendering APIs need the output stream attributes for the vertex shader and other APIs need them for the geometry shader. To keep the code logic simple, it is valid to declare the output attributes for both the vertex and geometry shader (or even all that will be used in the same shader program). Output attributes are ignored where they cannot be used.

See also

RenderingFeatures::hasStreamOutputs
CommandBuffer::BeginStreamOutput

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

- ShaderFlags.h

10.129 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.

10.129.1 Detailed Description

Video adapter descriptor structure.

Remarks

A video adapter determines the output capabilities of a GPU.

Todo Currently unused in the interface.

10.129.2 Member Data Documentation

10.129.2.1 std::wstring LLGL::VideoAdapterDescriptor::name

Hardware adapter name (name of the GPU).

10.129.2.2 std::vector<VideoOutputDescriptor> LLGL::VideoAdapterDescriptor::outputs

List of all adapter output descriptors.

10.129.2.3 std::string LLGL::VideoAdapterDescriptor::vendor

Vendor name (e.g. "NVIDIA Corporation", "Advanced Micro Devices, Inc." etc.).

10.129.2.4 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

10.130 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.

10.130.1 Detailed Description

Video output structure.

See also

VideoAdapterDescriptor::outputs

Todo Currently unused in the interface.

10.130.2 Member Data Documentation

10.130.2.1 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

10.131 LLGL::Viewport Struct Reference

Viewport dimensions.

```
#include <PipelineStateFlags.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.

10.131.1 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

10.131.2 Constructor & Destructor Documentation

10.131.2.1 LLGL::Viewport::Viewport ( ) [default]

10.131.2.2 LLGL::Viewport::Viewport (const Viewport &) [default]

10.131.2.3 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*) [inline]

Viewport constructor with default depth range of [0, 1].

10.131.2.4 LLGL::Viewport::Viewport (float *x*, float *y*, float *width*, float *height*, float *minDepth*, float *maxDepth*) [inline]

Viewport constructor with parameters for all attributes.

10.131.2.5 LLGL::Viewport::Viewport (const Extent2D & *extent*) [inline]

Viewport constructor with extent and default depth range of [0, 1].

10.131.2.6 LLGL::Viewport::Viewport (const Extent2D & *extent,* float *minDepth,* float *maxDepth*) [inline]

Viewport constructor with extent and explicit depth range.

10.131.2.7 LLGL::Viewport::Viewport (const Offset2D & *offset,* const Extent2D & *extent*) [inline]

Viewport constructor with offset, extent, and default depth range of [0, 1].

10.131.2.8 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.

10.131.3 Member Data Documentation

10.131.3.1 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.

10.131.3.2 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.

10.131.3.3 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.

10.131.3.4 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.

10.131.3.5 float LLGL::Viewport::x = 0.0f

Y coordinate of the left-top origin. By default 0.0.

10.131.3.6 float LLGL::Viewport::y = 0.0f

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

- PipelineStateFlags.h

10.132 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

- ~Window ()

  Releases the internal data.
- virtual void SetPosition (const Offset2D &position)=0

  Sets the window position relative to its parent.
- virtual Offset2D GetPosition () const =0

  Returns the window position relative to its parent (which can also be the display).
- virtual void SetSize (const Extent2D &size, bool useClientArea=true)=0

  Sets the either the overall window size or the client area size. By default the client area size is set.
- virtual Extent2D GetSize (bool useClientArea=true) const =0

  Returns either the overall window size or the client area size. By default the client area size is returned.
- virtual void SetTitle (const UTF8String &title)=0

  Sets the window title as UTF8 string.
- virtual UTF8String GetTitle () const =0

  Returns the window title as UTF8 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.
- bool AdaptForVideoMode (Extent2D *resolution, bool *fullscreen) override

  Adapts the window for the specified video mode.
- bool ProcessEvents () override final

  Processes the events for this window (i.e. mouse movement, key presses etc.).
- Display * FindResidentDisplay () const override final

  Searches the entire list of displays until a display is found where more than the half of this window's client area is visible.
- bool HasFocus () const

  Returns true if this window has the keyboard focus.
- bool HasQuit () const

  Returns true if this window is in the 'Quit' state.
- void SetBehavior (const WindowBehavior &behavior)

  Sets the new window behavior.
- const WindowBehavior & GetBehavior () const

  Returns the window heavior.
- void AddEventListener (const std::shared_ptr< EventListener > &eventListener)

  Adds the specified event listener to this window.
- void RemoveEventListener (const EventListener *eventListener)

  Removes the specified event listener from this window.
- void PostQuit ()

  Posts a 'Quit' event to all event listeners if the window is not yet in the 'Quit' state.
- void PostKeyDown (Key keyCode)

  Posts a 'KeyDown' event to all event listeners.
- void PostKeyUp (Key keyCode)
- void PostDoubleClick (Key keyCode)
- void PostChar (wchar_t chr)
- void PostWheelMotion (int motion)

- void PostLocalMotion (const Offset2D &position)
- void PostGlobalMotion (const Offset2D &motion)
- void PostResize (const Extent2D &clientAreaSize)
- void PostGetFocus ()

  Posts a 'GetFocus' event to all event listeners.
- void PostLostFocus ()

  Posts a 'LostFocus' event to all event listeners.
- void PostTimer (std::uint32_t timerID)

  Posts a timer event with the specified timer ID number.

Static Public Member Functions

- static std::unique_ptr< Window > Create (const WindowDescriptor &desc)

  Creates a platform specific instance of the Window interface.

Protected Member Functions

- Window ()

  Allocates the internal data.
- virtual void OnProcessEvents ()=0

  Called inside the "ProcessEvents" function after all event listeners received the same event.

10.132.1 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

10.132.2 Constructor & Destructor Documentation

10.132.2.1 LLGL::Window::~Window ( )

Releases the internal data.

10.132.2.2 LLGL::Window::Window ( ) [protected]

Allocates the internal data.

10.132.3 Member Function Documentation

10.132.3.1 bool LLGL::Window::AdaptForVideoMode (Extent2D * *resolution,* bool * *fullscreen*) [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.

10.132.3.2 void LLGL::Window::AdaptEventListener (const std::shared_ptr< EventListener > & *eventListener*)

Adds the specified event listener to this window.

10.132.3.3 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

10.132.3.4 Display* LLGL::Window::FindResidentDisplay ( ) const [final], [override], [virtual]

Searches the entire list of displays until a display is found where more than the half of this window's client area is visible.

Implements LLGL::Surface.

10.132.3.5 const WindowBehavior& LLGL::Window::GetBehavior ( ) const

Returns the window heavior.

10.132.3.6 virtual WindowDescriptor LLGL::Window::GetDesc ( ) const [pure virtual]

Queries a window descriptor, which describes the attributes of this window.

10.132.3.7 virtual Offset2D LLGL::Window::GetPosition ( ) const [pure virtual]

Returns the window position relative to its parent (which can also be the display).

10.132.3.8 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.

10.132.3.9 virtual UTF8String LLGL::Window::GetTitle ( ) const [pure virtual]

Returns the window title as UTF8 string.

10.132.3.10 bool LLGL::Window::HasFocus ( ) const

Returns true if this window has the keyboard focus.

10.132.3.11 bool LLGL::Window::HasQuit ( ) const

Returns true if this window is in the 'Quit' state.

See also

PostQuit
ProcessEvents

10.132.3.12 virtual bool LLGL::Window::IsShown ( ) const [pure virtual]

Returns true if this window is visible.

10.132.3.13 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

10.132.3.14 void LLGL::Window::PostChar (wchar_t *chr*)

See also

PostKeyDown

10.132.3.15 void LLGL::Window::PostDoubleClick (Key *keyCode*)

See also

PostKeyDown

10.132.3.16 void LLGL::Window::PostGetFocus ( )

Posts a 'GetFocus' event to all event listeners.

10.132.3.17 void LLGL::Window::PostGlobalMotion (const Offset2D & *motion*)

See also

PostKeyDown

10.132.3.18 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

10.132.3.19 void LLGL::Window::PostKeyUp (Key *keyCode*)

See also

PostKeyDown

10.132.3.20 void LLGL::Window::PostLocalMotion (const Offset2D & *position*)

See also

PostKeyDown

10.132.3.21 void LLGL::Window::PostLostFocus ( )

Posts a 'LostFocus' event to all event listeners.

10.132.3.22 void LLGL::Window::PostQuit ( )

Posts a 'Quit' event to all event listeners if the window is not yet in the 'Quit' state.

Remarks

If one or more event listeners set the `veto` parameter to true in the `OnQuit` callback, the window will not quit. Otherwise, the `ProcessEvents` function will return false from then on.

See also

EventListener::OnQuit
ProcessEvents
HasQuit

10.132.3.23 void LLGL::Window::PostResize (const Extent2D & *clientAreaSize*)

See also

PostKeyDown

10.132.3.24 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.

10.132.3.25 void LLGL::Window::PostWheelMotion (int *motion*)

See also

PostKeyDown

10.132.3.26 bool LLGL::Window::ProcessEvents ( ) [final], [override], [virtual]

Processes the events for this window (i.e. mouse movement, key presses etc.).

Returns

True, as long as the window can process events. Once the `PostQuit` function has set this window to the 'Quit' state, this function returns false. This happens when the user clicks on the close button.

See also

PostQuit

Implements LLGL::Surface.

10.132.3.27 void LLGL::Window::RemoveEventListener (const EventListener * *eventListener*)

Removes the specified event listener from this window.

10.132.3.28 void LLGL::Window::SetBehavior (const WindowBehavior & *behavior*)

Sets the new window behavior.

10.132.3.29 virtual void LLGL::Window::SetDesc (const WindowDescriptor & *desc*) [pure virtual]

Sets the window attributes according to the specified window descriptor.

10.132.3.30 virtual void LLGL::Window::SetPosition (const Offset2D & *position*) [pure virtual]

Sets the window position relative to its parent.

10.132.3.31 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.

10.132.3.32 virtual void LLGL::Window::SetTitle (const UTF8String & *title*) [pure virtual]

Sets the window title as UTF8 string.

10.132.3.33 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

10.133 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 = Constants::invalidTimerID

  Specifies an ID for a timer which will be activated when the window is moved or sized. By default invalidWindow↩ TimerID.

10.133.1 Detailed Description

Window behavior structure.

See also

Window::SetBehavior

10.133.2 Member Data Documentation

10.133.2.1 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.

10.133.2.2 std::uint32_t LLGL::WindowBehavior::moveAndResizeTimerID = Constants::invalidTimerID

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

10.134 LLGL::WindowDescriptor Struct Reference

Window descriptor structure.

```
#include <WindowFlags.h>
```

Public Attributes

- UTF8String 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 swap-chain. 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 centered = false

  Specifies whether the window is centered within the desktop screen. By default false.
- const void * windowContext = nullptr

  Window context handle.

10.134.1 Detailed Description

Window descriptor structure.

10.134.2 Member Data Documentation

10.134.2.1 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.

10.134.2.2 bool LLGL::WindowDescriptor::borderless = false

Specifies whether the window is borderless. This is required for a fullscreen swap-chain. By default false.

10.134.2.3 bool LLGL::WindowDescriptor::centered = false

Specifies whether the window is centered within the desktop screen. By default false.

10.134.2.4 Offset2D LLGL::WindowDescriptor::position

Window position (relative to the client area).

10.134.2.5 bool LLGL::WindowDescriptor::resizable = false

Specifies whether the window can be resized. By default false.

Remarks

For every window representing the surface for a SwapChain which has been resized, the video mode of that SwapChain 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 swap-chain is bound 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 {
        mySwapChain->ResizeBuffers(clientAreaSize);
    }
};
myWindow->AddEventListener(std::make_shared<MyEventListener>());

// Alternative 2
mySwapChain->ResizeBuffers(myWindow->GetContentSize());
myCmdBuffer->BeginRenderPass(*mySwapChain);
```

Note

Not updating the swap-chain on a resized window is undefined behavior.

See also

Surface::GetContentSize
Window::EventListener::OnResize

10.134.2.6 Extent2D LLGL::WindowDescriptor::size

Window size (this should be the client area size).

10.134.2.7 UTF8String LLGL::WindowDescriptor::title

Window title as unicode string.

10.134.2.8 bool LLGL::WindowDescriptor::visible = false

Specifies whether the window is visible at creation time. By default false.

10.134.2.9 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 AndroidNativeHandle.h File Reference

```
#include <android/native_window.h>
```

Classes

- struct LLGL::NativeHandle

  Android native handle structure.
- struct LLGL::NativeContextHandle

  Android native context handle structure.

Namespaces

- LLGL

11.2 ArrayView.h File Reference

```
#include <LLGL/Export.h>
#include <cstddef>
#include <iterator>
#include <initializer_list>
```

Classes

- class LLGL::ArrayView< T >

  Constant array view container template. Keeps pointer to array and length.

Namespaces

- LLGL

11.3 Blob.h File Reference

```
#include <LLGL/NonCopyable.h>
#include <LLGL/ImageFlags.h>
#include <memory>
#include <vector>
#include <string>
```

Classes

- class LLGL::Blob

  CPU read-only buffer of arbitrary size.

Namespaces

- LLGL

11.4 Buffer.h File Reference

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

Classes

- class LLGL::Buffer

  Hardware buffer interface.

Namespaces

- LLGL

11.5 BufferArray.h File Reference

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

Classes

- class LLGL::BufferArray

  Hardware buffer container interface.

Namespaces

- LLGL

11.6 BufferFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/ResourceFlags.h>
#include <LLGL/VertexAttribute.h>
#include <LLGL/RenderSystemFlags.h>
#include <LLGL/Constants.h>
#include <LLGL/Container/ArrayView.h>
#include <cstdint>
```

Classes

- struct LLGL::BufferDescriptor

  Hardware buffer descriptor structure.
- struct LLGL::BufferViewDescriptor

  Buffer view descriptor structure.

Namespaces

- LLGL

Functions

- LLGL_EXPORT bool LLGL::IsTypedBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a typed buffer, i.e. `Buffer` or `RWBuffer` in HLSL.
- LLGL_EXPORT bool LLGL::IsStructuredBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a structured buffer, i.e. `StructuredBuffer`, `RWStructuredBuffer`, `AppendStructuredBuffer`, or `ConsumeStructuredBuffer` in HLSL.
- LLGL_EXPORT bool LLGL::IsByteAddressBuffer (const BufferDescriptor &desc)

  Returns true if the buffer descriptor denotes a byte addresse buffer, i.e. `ByteAddressBuffer` or `RWByteAddressBuffer` in HLSL.

11.7 Canvas.h File Reference

```
#include <LLGL/CanvasFlags.h>
#include <LLGL/Surface.h>
#include <LLGL/Types.h>
#include <LLGL/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

11.8 CanvasFlags.h File Reference

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

Classes

- struct LLGL::CanvasDescriptor

  Canvas descriptor structure.

Namespaces

- LLGL

11.9 Color.h File Reference

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

11.10 ColorRGB.h File Reference

```
#include <LLGL/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 >

11.11 ColorRGBA.h File Reference

```
#include <LLGL/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 >

11.12 CommandBuffer.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/CommandBufferFlags.h>
#include <LLGL/RenderSystemFlags.h>
#include <LLGL/ColorRGBA.h>
#include <LLGL/Buffer.h>
#include <LLGL/BufferArray.h>
#include <LLGL/ResourceHeap.h>
#include <LLGL/PipelineLayoutFlags.h>
#include <LLGL/Constants.h>
#include <LLGL/RenderPass.h>
#include <LLGL/RenderTarget.h>
#include <LLGL/Shader.h>
#include <LLGL/PipelineState.h>
#include <LLGL/QueryHeap.h>
#include <cstdint>
```

Classes

- class LLGL::CommandBuffer

  Command buffer interface used for storing and encoding GPU commands.

Namespaces

- LLGL

11.13 CommandBufferFlags.h File Reference

```
#include <LLGL/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::MetalDependentStateDescriptor

  Graphics API dependent state descriptor for the Metal 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.
- enum LLGL::StencilFace { LLGL::StencilFace::FrontAndBack, LLGL::StencilFace::Front, LLGL::StencilFace::Back }

  Stencil face enumeration.
- enum LLGL::PipelineBindPoint { LLGL::PipelineBindPoint::Undefined, LLGL::PipelineBindPoint::Graphics, LLGL::PipelineBindPoint::Compute }

  Pipeline binding point enumeration.

11.14 CommandQueue.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/ForwardDecls.h>
#include <cstdint>
#include <cstddef>
```

Classes

- class LLGL::CommandQueue

  Command queue interface.

Namespaces

- LLGL

11.15 Constants.h File Reference

```
#include <cstdint>
#include <cstddef>
```

Namespaces

- LLGL
- LLGL::Constants

  Namespace with all constants used as default arguments.

Variables

- constexpr unsigned LLGL::Constants::maxThreadCount = -1

  Specifies the maximal number of threads the host system supports.
- constexpr std::uint32_t LLGL::Constants::ignoreOffset = -1

  Offset value to determine the offset automatically, e.g. to append a vertex attribute at the end of a vertex format.
- constexpr std::uint32_t LLGL::Constants::invalidSlot = -1

  Specifies an invalid binding slot for shader resources.
- constexpr std::uint32_t LLGL::Constants::invalidLocation = -1

  Specifies an invalid shader uniform location.
- constexpr std::uint64_t LLGL::Constants::wholeSize = -1

  Specifies to use the whole size of a resource.
- constexpr std::uint32_t LLGL::Constants::invalidTimerID = 0u

  Specifies an invalid timer ID for window events.

11.16 Display.h File Reference

```
#include <LLGL/Interface.h>
#include <LLGL/DisplayFlags.h>
#include <LLGL/Container/Strings.h>
#include <vector>
```

Classes

- class LLGL::Display

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

Namespaces

- LLGL

11.17 DisplayFlags.h File Reference

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

Classes

- class LLGL::Fence

  Fence interface for CPU/GPU synchronization.

Namespaces

- LLGL

11.20 Format.h File Reference

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

Classes

- struct LLGL::FormatFlags

  Format attribute flags enumeration.
- struct LLGL::FormatAttributes

  Vertex and texture format attributes structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::Format {
  LLGL::Format::Undefined, LLGL::Format::A8UNorm, LLGL::Format::R8UNorm, LLGL::Format::R8SNorm, LLGL::Format::R8UInt, LLGL::Format::R8SInt, LLGL::Format::R16UNorm, LLGL::Format::R16SNorm, LLGL::Format::R16UInt, LLGL::Format::R16SInt, LLGL::Format::R16Float, LLGL::Format::R32UInt, LLGL::Format::R32SInt, LLGL::Format::R32Float, LLGL::Format::R64Float, LLGL::Format::RG8UNorm, LLGL::Format::RG8SNorm, LLGL::Format::RG8UInt, LLGL::Format::RG8SInt, LLGL::Format::RG16UNorm, LLGL::Format::RG16SNorm, LLGL::Format::RG16UInt, LLGL::Format::RG16SInt, LLGL::Format::RG16Float,
  LLGL::Format::RG32UInt, LLGL::Format::RG32SInt, LLGL::Format::RG32Float, LLGL::Format::RG64Float, LLGL::Format::RGB8UNorm, LLGL::Format::RGB8UNorm_sRGB, LLGL::Format::RGB8SNorm, LLGL::Format::RGB8UInt,
  LLGL::Format::RGB8SInt, LLGL::Format::RGB16UNorm, LLGL::Format::RGB16SNorm, LLGL::Format::RGB16UInt,
  LLGL::Format::RGB16SInt, LLGL::Format::RGB16Float, LLGL::Format::RGB32UInt, LLGL::Format::RGB32SInt,
  LLGL::Format::RGB32Float, LLGL::Format::RGB64Float, LLGL::Format::RGBA8UNorm, LLGL::Format::RGBA8UNorm_sRGB,
  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::RGBA64Float,
  LLGL::Format::BGRA8UNorm, LLGL::Format::BGRA8UNorm_sRGB, LLGL::Format::BGRA8SNorm, LLGL::Format::BGRA8UInt,
  LLGL::Format::BGRA8SInt, LLGL::Format::RGB10A2UNorm, LLGL::Format::RGB10A2UInt, LLGL::Format::RG11B10Float,
  LLGL::Format::RGB9E5Float, LLGL::Format::D16UNorm, LLGL::Format::D24UNormS8UInt, LLGL::Format::D32Float,
  LLGL::Format::D32FloatS8X24UInt, LLGL::Format::BC1UNorm, LLGL::Format::BC1UNorm_sRGB, LLGL::Format::BC2UNorm,
  LLGL::Format::BC2UNorm_sRGB, LLGL::Format::BC3UNorm, LLGL::Format::BC3UNorm_sRGB, LLGL::Format::BC4UNorm,
  LLGL::Format::BC4SNorm, LLGL::Format::BC5UNorm, LLGL::Format::BC5SNorm }

  Texture and vertex attribute format enumeration used for GPU side operations.
- enum LLGL::ImageFormat {
  LLGL::ImageFormat::Alpha, LLGL::ImageFormat::R, LLGL::ImageFormat::RG, LLGL::ImageFormat::RGB, LLGL::ImageFormat::BGR, LLGL::ImageFormat::RGBA, LLGL::ImageFormat::BGRA, LLGL::ImageFormat::ARGB,
  LLGL::ImageFormat::ABGR, LLGL::ImageFormat::Depth, LLGL::ImageFormat::DepthStencil, LLGL::ImageFormat::BC1,
  LLGL::ImageFormat::BC2, LLGL::ImageFormat::BC3, LLGL::ImageFormat::BC4, LLGL::ImageFormat::BC5
  }

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

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

  Data types enumeration used for CPU side operations.

Functions

- LLGL_EXPORT const FormatAttributes & LLGL::GetFormatAttribs (const Format format)

  Returns the attributes for the specified hardware format or a zero-initialized structure in case of an invalid argument.
- 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::GetMemoryFootprint (const Format format, std::uint32_t count)

  Returns the memory footprint (in bytes) of a texture or buffer with the specified hardware format and number of elements.
- LLGL_EXPORT std::uint32_t LLGL::GetMemoryFootprint (const ImageFormat imageFormat, const DataType dataType, std::uint32_t count)

  Returns the memory footprint (in bytes) of an image with the specified format, data type, and number of elements.
- LLGL_EXPORT bool LLGL::IsCompressedFormat (const Format format)

  Returns true if the specified hardware format is a compressed format, e.g. Format::BC1UNorm, Format::BC2UNorm_sRGB, Format::BC4SNorm, etc.
- 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 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::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::IsDepthFormat (const Format format)

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

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

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

  Returns true if the specified hardware format is an integral format (like Format::RGBA8UInt, Format::RGBA8UNorm, 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 ForRange.h File Reference

```
#include <type_traits>
```

Macros

- #define for_range_end(INDEX) (INDEX ## _End)
- #define for_subrange(INDEX, BEGIN, END)
- #define for_range(INDEX, END) for_subrange(INDEX, 0, END)
- #define for_subrange_reverse(INDEX, START, END)
- #define for_range_reverse(INDEX, END) for_subrange_reverse(INDEX, 0, END)
- #define for_range_reverse_iter(INDEX) (INDEX ## _Iter)

11.21.1 Macro Definition Documentation

11.21.1.1 #define for_range(*INDEX, END*) for_subrange(INDEX, 0, END)

11.21.1.2 #define for_range_end(*INDEX*) (INDEX ## _End)

11.21.1.3 #define for_range_reverse(*INDEX, END*) for_subrange_reverse(INDEX, 0, END)

11.21.1.4 #define for_range_reverse_iter(*INDEX*) (INDEX ## _Iter)

11.21.1.5 #define for_subrange(*INDEX, BEGIN, END*)

Value:

```
for
                                \
    (
                                    \
        typename std::remove_cv<typename std::remove_reference<decltype(END)>::type>::type INDEX ## _End =
      (END), INDEX = (BEGIN);  \
        INDEX < for_range_end(INDEX);
                                                \
        ++INDEX
                                    \
    )
```

11.21.1.6 #define for_subrange_reverse(*INDEX, START, END*)

Value:

```
for
                                                \
    (
                                                    \
        typename std::remove_cv<typename std::remove_reference<decltype(END)>::type>::type INDEX ## _End =
      (END), INDEX ## _Iter = (START), INDEX;  \
        (INDEX = for_range_end(INDEX) - (INDEX ## _Iter) - 1), (INDEX ## _Iter) <
      for_range_end(INDEX);                                                 \
        ++(INDEX ## _Iter)
                                                    \
    )
```

11.22 ForwardDecls.h File Reference

Namespaces

- LLGL

11.23 FragmentAttribute.h File Reference

```
#include <LLGL/Format.h>
#include <LLGL/SystemValue.h>
#include <LLGL/Container/StringView.h>
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::FragmentAttribute

  Fragment output attribute structure.

Namespaces

- LLGL

11.24 Image.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Types.h>
#include <LLGL/ImageFlags.h>
#include <LLGL/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 <LLGL/Export.h>
#include <LLGL/Format.h>
#include <LLGL/Tags.h>
#include <LLGL/RenderSystemFlags.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/ColorRGBA.h>
#include <memory>
#include <cstdint>
```

Classes

- struct LLGL::SrcImageDescriptor

  Descriptor structure for an image that is used as source for reading the image data.
- struct LLGL::DstImageDescriptor

  Descriptor structure for an image that is used as destination for writing the image data.

Namespaces

- LLGL

Typedefs

- using LLGL::ByteBuffer = std::unique_ptr< char[ ]>

  Common byte buffer type.

Functions

- LLGL_EXPORT bool LLGL::ConvertImageBuffer (const SrcImageDescriptor &srcImageDesc, const Dst↩ImageDescriptor &dstImageDesc, unsigned 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, unsigned 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::DecompressImageBufferToRGBA8UNorm (const SrcImageDescriptor &srcImageDesc, const Extent2D &extent, unsigned threadCount=0)

  Decompresses the specified image buffer to RGBA format with 8-bit unsigned normalized integers.
- LLGL_EXPORT void LLGL::CopyImageBufferRegion (const DstImageDescriptor &dstImageDesc, const Offset3D &dstOffset, std::uint32_t dstRowStride, std::uint32_t dstSliceStride, const SrcImageDescriptor &src↩ImageDesc, const Offset3D &srcOffset, std::uint32_t srcRowStride, std::uint32_t srcSliceStride, const Extent3D &extent)

  Copies an image buffer region from the source buffer to the destination 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::AllocateByteBuffer (std::size_t bufferSize)

  Generates a new byte buffer with zeros in each byte.
- LLGL_EXPORT ByteBuffer LLGL::AllocateByteBuffer (std::size_t bufferSize, UninitializeTag)

  Generates a new and uninitialized byte buffer.

11.26 IndirectArguments.h File Reference

```
#include <cstdint>
```

Classes

- struct LLGL::DrawIndirectArguments

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

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

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

  Format structure for the arguments of an indirect compute command.

Namespaces

- LLGL

11.27 Input.h File Reference

```
#include <LLGL/Surface.h>
#include <LLGL/Types.h>
#include <LLGL/Key.h>
```

Classes

- class LLGL::Input

  Default window event listener to receive user input.

Namespaces

- LLGL

11.28 Interface.h File Reference

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

Classes

- class LLGL::Interface

  Base class for all interfaces in LLGL.

Namespaces

- LLGL

Macros

- #define LLGL_DECLARE_INTERFACE(ID)

  Declares the base functions for all LLGL interfaces.
- #define LLGL_IMPLEMENT_INTERFACE(SELF, BASE)

  Implements the base functions for all LLGL interfaces.

11.28.1 Macro Definition Documentation

11.28.1.1 #define LLGL_DECLARE_INTERFACE(*ID*)

Value:

```
public:                                         \
        inline static int GetInterfaceID()          \
        {                                           \
            return (ID);                            \
        }                                           \
        bool IsInstanceOf(int id) const override
```

Declares the base functions for all LLGL interfaces.

11.28.1.2 #define LLGL_IMPLEMENT_INTERFACE(*SELF*, *BASE*)

Value:

```
bool SELF::IsInstanceOf(int id) const
                    \
    {
       \
        return (id == SELF::GetInterfaceID() || (id < SELF::GetInterfaceID() &&
      BASE::IsInstanceOf(id)));    \
    }
```

Implements the base functions for all LLGL interfaces.

11.29 InterfaceID.h File Reference

Classes

- struct LLGL::InterfaceID

  Enumeration of all interfaces in LLGL.

Namespaces

- LLGL

11.30 IOSNativeHandle.h File Reference

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

Classes

- struct LLGL::NativeHandle

  Android native handle structure.
- struct LLGL::NativeContextHandle

  Android native context handle structure.

Namespaces

- LLGL

11.31 Key.h File Reference

Namespaces

- LLGL

Enumerations

- enum LLGL::Key {
  LLGL::Key::LButton, LLGL::Key::RButton, LLGL::Key::Cancel, LLGL::Key::MButton,
  LLGL::Key::XButton1, LLGL::Key::XButton2, LLGL::Key::Back, LLGL::Key::Tab,
  LLGL::Key::Clear, LLGL::Key::Return, LLGL::Key::Shift, LLGL::Key::Control,
  LLGL::Key::Menu, LLGL::Key::Pause, LLGL::Key::Capital, LLGL::Key::Escape,
  LLGL::Key::Space, LLGL::Key::PageUp, LLGL::Key::PageDown, LLGL::Key::End,
  LLGL::Key::Home, LLGL::Key::Left, LLGL::Key::Up, LLGL::Key::Right,
  LLGL::Key::Down, LLGL::Key::Select, LLGL::Key::Print, LLGL::Key::Exe,
  LLGL::Key::Snapshot, LLGL::Key::Insert, LLGL::Key::Delete, LLGL::Key::Help,
  LLGL::Key::D0, LLGL::Key::D1, LLGL::Key::D2, LLGL::Key::D3,
  LLGL::Key::D4, LLGL::Key::D5, LLGL::Key::D6, LLGL::Key::D7,
  LLGL::Key::D8, LLGL::Key::D9, LLGL::Key::A, LLGL::Key::B,
  LLGL::Key::C, LLGL::Key::D, LLGL::Key::E, LLGL::Key::F,
  LLGL::Key::G, LLGL::Key::H, LLGL::Key::I, LLGL::Key::J,
  LLGL::Key::K, LLGL::Key::L, LLGL::Key::M, LLGL::Key::N,
  LLGL::Key::O, LLGL::Key::P, LLGL::Key::Q, LLGL::Key::R,
  LLGL::Key::S, LLGL::Key::T, LLGL::Key::U, LLGL::Key::V,
  LLGL::Key::W, LLGL::Key::X, LLGL::Key::Y, LLGL::Key::Z,
  LLGL::Key::LWin, LLGL::Key::RWin, LLGL::Key::Apps, LLGL::Key::Sleep,
  LLGL::Key::Keypad0, LLGL::Key::Keypad1, LLGL::Key::Keypad2, LLGL::Key::Keypad3,
  LLGL::Key::Keypad4, LLGL::Key::Keypad5, LLGL::Key::Keypad6, LLGL::Key::Keypad7,
  LLGL::Key::Keypad8, LLGL::Key::Keypad9, LLGL::Key::KeypadMultiply, LLGL::Key::KeypadPlus,
  LLGL::Key::KeypadSeparator, LLGL::Key::KeypadMinus, LLGL::Key::KeypadDecimal, LLGL::Key::Keypad↩

Divide,
LLGL::Key::F1, LLGL::Key::F2, LLGL::Key::F3, LLGL::Key::F4,
LLGL::Key::F5, LLGL::Key::F6, LLGL::Key::F7, LLGL::Key::F8,
LLGL::Key::F9, LLGL::Key::F10, LLGL::Key::F11, LLGL::Key::F12,
LLGL::Key::F13, LLGL::Key::F14, LLGL::Key::F15, LLGL::Key::F16,
LLGL::Key::F17, LLGL::Key::F18, LLGL::Key::F19, LLGL::Key::F20,
LLGL::Key::F21, LLGL::Key::F22, LLGL::Key::F23, LLGL::Key::F24,
LLGL::Key::NumLock, LLGL::Key::ScrollLock, LLGL::Key::LShift, LLGL::Key::RShift,
LLGL::Key::LControl, LLGL::Key::RControl, LLGL::Key::LMenu, LLGL::Key::RMenu,
LLGL::Key::BrowserBack, LLGL::Key::BrowserForward, LLGL::Key::BrowserRefresh, LLGL::Key::Browser↩
Stop,
LLGL::Key::BrowserSearch, LLGL::Key::BrowserFavorits, LLGL::Key::BrowserHome, LLGL::Key::Volume↩
Mute,
LLGL::Key::VolumeDown, LLGL::Key::VolumeUp, LLGL::Key::MediaNextTrack, LLGL::Key::MediaPrevTrack,
LLGL::Key::MediaStop, LLGL::Key::MediaPlayPause, LLGL::Key::LaunchMail, LLGL::Key::LaunchMedia↩
Select,
LLGL::Key::LaunchApp1, LLGL::Key::LaunchApp2, LLGL::Key::Plus, LLGL::Key::Comma,
LLGL::Key::Minus, LLGL::Key::Period, LLGL::Key::Exponent, LLGL::Key::Attn,
LLGL::Key::CrSel, LLGL::Key::ExSel, LLGL::Key::ErEOF, LLGL::Key::Play,
LLGL::Key::Zoom, LLGL::Key::NoName, LLGL::Key::PA1, LLGL::Key::OEMClear,
LLGL::Key::Any }

Input key codes.

11.32 LinuxNativeHandle.h File Reference

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

Classes

- struct LLGL::NativeHandle

  Android native handle structure.
- struct LLGL::NativeContextHandle

  Android native context handle structure.

Namespaces

- LLGL

11.33 LLGL.h File Reference

```
#include <LLGL/Version.h>
```

```
#include <LLGL/Window.h>
#include <LLGL/Canvas.h>
#include <LLGL/Display.h>
#include <LLGL/Input.h>
#include <LLGL/Timer.h>
#include <LLGL/TypeInfo.h>
#include <LLGL/ColorRGB.h>
#include <LLGL/ColorRGBA.h>
#include <LLGL/RenderSystem.h>
#include <LLGL/Log.h>
#include <LLGL/IndirectArguments.h>
#include <LLGL/ImageFlags.h>
```

11.34 Log.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Container/StringView.h>
#include <functional>
#include <ostream>
```

Namespaces

- LLGL
- LLGL::Log

Typedefs

- using LLGL::Log::ReportCallback = std::function< void(ReportType type, const StringView &message, const StringView &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 StringView &message, const StringView &contextInfo={})

  Posts a report to the currently set report callback.
- LLGL_EXPORT void LLGL::Log::SetReportCallback (const ReportCallback &callback, void *userData=nullptr)

  Sets the new report callback. No report callback is specified by default, in which case the reports are ignored.
- LLGL_EXPORT void LLGL::Log::SetReportCallbackStd (std::ostream *stream)

  Sets the new report callback to the standard output streams.
- LLGL_EXPORT void LLGL::Log::SetReportLimit (std::size_t maxCount)

  Sets the maximum number of reports that will be triggered. All remaining reports will be ignored.

11.35 MacOSNativeHandle.h File Reference

```
#include <Cocoa/Cocoa.h>
```

Classes

- struct LLGL::NativeHandle

  Android native handle structure.
- struct LLGL::NativeContextHandle

  Android native context handle structure.

Namespaces

- LLGL

11.36 NativeHandle.h File Reference

```
#include "Platform.h"
```

11.37 NonCopyable.h File Reference

```
#include <LLGL/Export.h>
```

Classes

- class LLGL::NonCopyable

  Base class for all classes with virtual function tables in LLGL.

Namespaces

- LLGL

11.38 PipelineLayout.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <cstdint>
```

Classes

- class LLGL::PipelineLayout

  Pipeline layout interface.

Namespaces

- LLGL

11.39 PipelineLayoutFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/ResourceFlags.h>
#include <LLGL/BufferFlags.h>
#include <LLGL/ShaderFlags.h>
#include <vector>
```

Classes

- struct LLGL::BindingDescriptor

  Layout structure for a single binding point of the pipeline layout descriptor.
- struct LLGL::PipelineLayoutDescriptor

  Pipeline layout descritpor structure.

Namespaces

- LLGL

11.40 PipelineState.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/Report.h>
```

Classes

- class LLGL::PipelineState

  Graphics and compute pipeline state interface.

Namespaces

- LLGL

11.41 PipelineStateFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/ColorRGBA.h>
#include <LLGL/Types.h>
#include <LLGL/Format.h>
#include <LLGL/ForwardDecls.h>
#include <LLGL/StaticLimits.h>
#include <vector>
#include <cstdint>
```

Classes

- struct LLGL::Viewport

  Viewport dimensions.
- struct LLGL::Scissor

  Scissor dimensions.
- 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::TessellationDescriptor

  Tessellation descriptor structure for the graphics pipeline.
- struct LLGL::GraphicsPipelineDescriptor

  Graphics pipeline state descriptor structure.
- struct LLGL::ComputePipelineDescriptor

  Compute pipeline state descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::PrimitiveTopology { LLGL::PrimitiveTopology::PointList, LLGL::PrimitiveTopology::LineList, LLGL::PrimitiveTopology::LineStrip, LLGL::PrimitiveTopology::LineListAdjacency, LLGL::PrimitiveTopology::LineStripAdjacency, LLGL::PrimitiveTopology::TriangleList, LLGL::PrimitiveTopology::TriangleStrip, LLGL::PrimitiveTopology::TriangleListAdjacency, LLGL::PrimitiveTopology::TriangleStripAdjacency, LLGL::PrimitiveTopology::Patches1, LLGL::PrimitiveTopology::Patches2, LLGL::PrimitiveTopology::Patches3, LLGL::PrimitiveTopology::Patches4, LLGL::PrimitiveTopology::Patches5, LLGL::PrimitiveTopology::Patches6, LLGL::PrimitiveTopology::Patches7, LLGL::PrimitiveTopology::Patches8, LLGL::PrimitiveTopology::Patches9, LLGL::PrimitiveTopology::Patches10, LLGL::PrimitiveTopology::Patches11, LLGL::PrimitiveTopology::Patches12, LLGL::PrimitiveTopology::Patches13, LLGL::PrimitiveTopology::Patches14, LLGL::PrimitiveTopology::Patches15, LLGL::PrimitiveTopology::Patches16, LLGL::PrimitiveTopology::Patches17, LLGL::PrimitiveTopology::Patches18, LLGL::PrimitiveTopology::Patches19, LLGL::PrimitiveTopology::Patches20, LLGL::PrimitiveTopology::Patches21, LLGL::PrimitiveTopology::Patches22, LLGL::PrimitiveTopology::Patches23, LLGL::PrimitiveTopology::Patches24, LLGL::PrimitiveTopology::Patches25, LLGL::PrimitiveTopology::Patches26, LLGL::PrimitiveTopology::Patches27, LLGL::PrimitiveTopology::Patches28, LLGL::PrimitiveTopology::Patches29, LLGL::PrimitiveTopology::Patches30, LLGL::PrimitiveTopology::Patches31, LLGL::PrimitiveTopology::Patches32 }

  Primitive topology enumeration.
- enum LLGL::CompareOp { LLGL::CompareOp::NeverPass, LLGL::CompareOp::Less, LLGL::CompareOp::Equal, LLGL::CompareOp::LessEqual, LLGL::CompareOp::Greater, LLGL::CompareOp::NotEqual, LLGL::CompareOp::GreaterEqual, LLGL::CompareOp::AlwaysPass }

  Compare operations enumeration.
- enum LLGL::StencilOp { LLGL::StencilOp::Keep, LLGL::StencilOp::Zero, LLGL::StencilOp::Replace, LLGL::StencilOp::IncClamp, LLGL::StencilOp::DecClamp, LLGL::StencilOp::Invert, LLGL::StencilOp::IncWrap, LLGL::StencilOp::DecWrap }

  Stencil operations enumeration.
- enum LLGL::BlendOp { LLGL::BlendOp::Zero, LLGL::BlendOp::One, LLGL::BlendOp::SrcColor, LLGL::BlendOp::InvSrcColor, LLGL::BlendOp::SrcAlpha, LLGL::BlendOp::InvSrcAlpha, LLGL::BlendOp::DstColor, LLGL::BlendOp::InvDstColor, LLGL::BlendOp::DstAlpha, LLGL::BlendOp::InvDstAlpha, LLGL::BlendOp::SrcAlphaSaturate, LLGL::BlendOp::BlendFactor, LLGL::BlendOp::InvBlendFactor, LLGL::BlendOp::Src1Color, LLGL::BlendOp::InvSrc1Color, LLGL::BlendOp::Src1Alpha, LLGL::BlendOp::InvSrc1Alpha }

  Blending operations enumeration.
- enum LLGL::BlendArithmetic { LLGL::BlendArithmetic::Add, LLGL::BlendArithmetic::Subtract, LLGL::BlendArithmetic::RevSubtract, LLGL::BlendArithmetic::Min, LLGL::BlendArithmetic::Max }

  Blending arithmetic operations enumeration.
- enum LLGL::PolygonMode { LLGL::PolygonMode::Fill, LLGL::PolygonMode::Wireframe, LLGL::PolygonMode::Points }

  Polygon filling modes enumeration.
- enum LLGL::CullMode { LLGL::CullMode::Disabled, LLGL::CullMode::Front, LLGL::CullMode::Back }

  Polygon culling modes enumeration.

- enum LLGL::LogicOp {
  LLGL::LogicOp::Disabled, LLGL::LogicOp::Clear, LLGL::LogicOp::Set, LLGL::LogicOp::Copy, LLGL::LogicOp::CopyInverted, LLGL::LogicOp::NoOp, LLGL::LogicOp::Invert, LLGL::LogicOp::AND, LLGL::LogicOp::ANDReverse, LLGL::LogicOp::ANDInverted, LLGL::LogicOp::NAND, LLGL::LogicOp::OR, LLGL::LogicOp::ORReverse, LLGL::LogicOp::ORInverted, LLGL::LogicOp::NOR, LLGL::LogicOp::XOR, LLGL::LogicOp::Equiv }

  Logical pixel operation enumeration.
- enum LLGL::TessellationPartition {
  LLGL::TessellationPartition::Undefined, LLGL::TessellationPartition::Integer, LLGL::TessellationPartition::Pow2, LLGL::TessellationPartition::FractionalOdd, LLGL::TessellationPartition::FractionalEven }

  Tessellation partition mode enumeration.

Functions

- LLGL_EXPORT bool LLGL::IsPrimitiveTopologyPatches (const PrimitiveTopology primitiveTopology)

  Returns true if the specified primitive topology is a patch list.
- LLGL_EXPORT bool LLGL::IsPrimitiveTopologyStrip (const PrimitiveTopology primitiveTopology)

  Returns true if the specified primitive topology is a strip that generates a new primitive with each new vertex.
- LLGL_EXPORT std::uint32_t LLGL::GetPrimitiveTopologyPatchSize (const PrimitiveTopology primitiveTopology)

  Returns the number of patch control points of the specified primitive topology (in range [1, 32]), or 0 if the topology is not a patch list.

11.42 Platform.h File Reference

Macros

- #define LLGL_API
- #define LLGL_API_CDECL
- #define LLGL_API_STDCALL
- #define LLGL_API_THISCALL

11.42.1 Macro Definition Documentation

11.42.1.1 #define LLGL_API

11.42.1.2 #define LLGL_API_CDECL

11.42.1.3 #define LLGL_API_STDCALL

11.42.1.4 #define LLGL_API_THISCALL

11.43 QueryHeap.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/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

11.44 QueryHeapFlags.h File Reference

```
#include <LLGL/Constants.h>
#include <cstdint>
```

Classes

- struct LLGL::QueryPipelineStatistics

  Query data structure for pipeline statistics.
- struct LLGL::QueryHeapDescriptor

  Query heap descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::QueryType { LLGL::QueryType::SamplesPassed, LLGL::QueryType::AnySamplesPassed, LLGL::QueryType::AnySamplesPassedConservative, LLGL::QueryType::TimeElapsed, LLGL::QueryType::StreamOutPrimitivesWritten, LLGL::QueryType::StreamOutOverflow, LLGL::QueryType::PipelineStatistics }

  Query type enumeration.

11.45 RendererConfiguration.h File Reference

```
#include <LLGL/Container/ArrayView.h>
#include <cstdint>
```

Classes

- struct LLGL::ApplicationDescriptor

  Application descriptor structure.
- struct LLGL::RendererConfigurationVulkan

  Structure for a Vulkan renderer specific configuration.
- struct LLGL::RendererConfigurationOpenGL

  OpenGL profile descriptor structure.
- struct LLGL::RendererConfigurationOpenGLES3

  OpenGL ES 3 profile descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::OpenGLContextProfile { LLGL::OpenGLContextProfile::CompatibilityProfile, LLGL::OpenGL↩ ContextProfile::CoreProfile, LLGL::OpenGLContextProfile::ESProfile }

  OpenGL context profile enumeration.

11.46 RenderingDebugger.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Container/Strings.h>
```

Classes

- class LLGL::RenderingDebugger

  Rendering debugger interface.
- class LLGL::RenderingDebugger::Message

  Rendering debugger message class.

Namespaces

- LLGL

Enumerations

- enum LLGL::ErrorType { LLGL::ErrorType::InvalidArgument, LLGL::ErrorType::InvalidState, LLGL::Error↩ Type::UnsupportedFeature, LLGL::ErrorType::UndefinedBehavior }

  Rendering debugger error types enumeration.
- enum LLGL::WarningType { LLGL::WarningType::ImproperArgument, LLGL::WarningType::ImproperState, LLGL::WarningType::PointlessOperation, LLGL::WarningType::VaryingBehavior }

  Rendering debugger warning types enumeration.

11.47 RenderingProfiler.h File Reference

```
#include <LLGL/SwapChainFlags.h>
#include <LLGL/PipelineStateFlags.h>
#include <cstdint>
#include <string.h>
```

Classes

- struct LLGL::ProfileTimeRecord

  Structure with annotation and elapsed time for a timer profile.
- struct LLGL::FrameProfile

  Profile of a rendered frame.
- class LLGL::RenderingProfiler

  Rendering profiler model class.

Namespaces

- LLGL

11.48 RenderPass.h File Reference

```
#include <LLGL/RenderSystemChild.h>
```

Classes

- class LLGL::RenderPass

  Render pass interface.

Namespaces

- LLGL

11.49 RenderPassFlags.h File Reference

```
#include <LLGL/Format.h>
#include <LLGL/StaticLimits.h>
```

Classes

- struct LLGL::AttachmentFormatDescriptor

  Render pass attachment descriptor structure.

- struct LLGL::RenderPassDescriptor

  Render pass descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::AttachmentLoadOp { LLGL::AttachmentLoadOp::Undefined, LLGL::AttachmentLoadOp::Load, LLGL::AttachmentLoadOp::Clear }

  Enumeration for render pass attachment load operations.

- enum LLGL::AttachmentStoreOp { LLGL::AttachmentStoreOp::Undefined, LLGL::AttachmentStoreOp::Store }

  Enumeration for render pass attachment store operations.

11.50 RenderSystem.h File Reference

```
#include <LLGL/Interface.h>
```

```
#include <LLGL/SwapChain.h>
#include <LLGL/CommandQueue.h>
#include <LLGL/CommandBuffer.h>
#include <LLGL/RenderSystemFlags.h>
#include <LLGL/RenderingProfiler.h>
#include <LLGL/RenderingDebugger.h>
#include <LLGL/Blob.h>
#include <LLGL/Container/ArrayView.h>
#include <LLGL/Buffer.h>
#include <LLGL/BufferFlags.h>
#include <LLGL/BufferArray.h>
#include <LLGL/Texture.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/Sampler.h>
#include <LLGL/SamplerFlags.h>
#include <LLGL/ResourceHeap.h>
#include <LLGL/ResourceHeapFlags.h>
#include <LLGL/RenderPass.h>
#include <LLGL/RenderPassFlags.h>
#include <LLGL/RenderTarget.h>
#include <LLGL/RenderTargetFlags.h>
#include <LLGL/Shader.h>
#include <LLGL/ShaderFlags.h>
#include <LLGL/ShaderReflection.h>
#include <LLGL/PipelineLayout.h>
#include <LLGL/PipelineLayoutFlags.h>
#include <LLGL/PipelineState.h>
#include <LLGL/PipelineStateFlags.h>
#include <LLGL/QueryHeap.h>
#include <LLGL/QueryHeapFlags.h>
#include <LLGL/Fence.h>
#include <string>
#include <memory>
#include <vector>
#include <cstdint>
```

Classes

- class LLGL::RenderSystemDeleter

  Delegate to delete an instance of the RenderSystem interface.
- class LLGL::RenderSystem

  Render system interface.

Namespaces

- LLGL

Typedefs

- using LLGL::RenderSystemPtr = std::unique_ptr< RenderSystem, RenderSystemDeleter >

  Unique pointer type for the RenderSystem interface with a custom deleter.

11.51 RenderSystemChild.h File Reference

```
#include <LLGL/Interface.h>
```

Classes

- class LLGL::RenderSystemChild

  Base class for all interfaces whoes instances are owned by the RenderSystem.

Namespaces

- LLGL

11.52 RenderSystemFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/CommandBufferFlags.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/Constants.h>
#include <LLGL/RendererConfiguration.h>
#include <LLGL/Platform/Platform.h>
#include <cstddef>
#include <cstdint>
#include <string>
#include <vector>
#include <functional>
```

Classes

- struct LLGL::RendererID

  Renderer identification number enumeration.
- struct LLGL::RendererInfo

  Renderer basic information structure.
- struct LLGL::RenderSystemDescriptor

  Render system descriptor structure.
- struct LLGL::RenderingFeatures

  Contains the attributes for all supported rendering features.
- struct LLGL::RenderingLimits

  Contains all rendering limitations such as maximum buffer size, maximum texture resolution etc.
- struct LLGL::RenderingCapabilities

  Structure with all attributes describing the rendering capabilities of the render system.

Namespaces

- LLGL

Typedefs

- using LLGL::DebugCallback = std::function< void(const std::string &type, const std::string &message)>

  Debug callback function interface.
- using LLGL::ValidateRenderingCapsFunc = std::function< bool(const std::string &info, const std::string &attrib)>

  Callback interface for the ValidateRenderingCaps function.

Enumerations

- enum LLGL::ShadingLanguage {
  LLGL::ShadingLanguage::GLSL = (0x10000), LLGL::ShadingLanguage::GLSL_110 = (0x10000 | 110), LLGL::ShadingLanguage::GLSL_120 = (0x10000 | 120), LLGL::ShadingLanguage::GLSL_130 = (0x10000 | 130),
  LLGL::ShadingLanguage::GLSL_140 = (0x10000 | 140), LLGL::ShadingLanguage::GLSL_150 = (0x10000 | 150), LLGL::ShadingLanguage::GLSL_330 = (0x10000 | 330), LLGL::ShadingLanguage::GLSL_400 = (0x10000 | 400),
  LLGL::ShadingLanguage::GLSL_410 = (0x10000 | 410), LLGL::ShadingLanguage::GLSL_420 = (0x10000 | 420), LLGL::ShadingLanguage::GLSL_430 = (0x10000 | 430), LLGL::ShadingLanguage::GLSL_440 = (0x10000 | 440),
  LLGL::ShadingLanguage::GLSL_450 = (0x10000 | 450), LLGL::ShadingLanguage::GLSL_460 = (0x10000 | 460), LLGL::ShadingLanguage::ESSL = (0x20000), LLGL::ShadingLanguage::ESSL_100 = (0x20000 | 100), LLGL::ShadingLanguage::ESSL_300 = (0x20000 | 300), LLGL::ShadingLanguage::ESSL_310 = (0x20000 | 310), LLGL::ShadingLanguage::ESSL_320 = (0x20000 | 320), LLGL::ShadingLanguage::HLSL = (0x30000), LLGL::ShadingLanguage::HLSL_2_0 = (0x30000 | 200), LLGL::ShadingLanguage::HLSL_2_0a = (0x30000 | 201), LLGL::ShadingLanguage::HLSL_2_0b = (0x30000 | 202), LLGL::ShadingLanguage::HLSL_3_0 = (0x30000 | 300),
  LLGL::ShadingLanguage::HLSL_4_0 = (0x30000 | 400), LLGL::ShadingLanguage::HLSL_4_1 = (0x30000 | 410), LLGL::ShadingLanguage::HLSL_5_0 = (0x30000 | 500), LLGL::ShadingLanguage::HLSL_5_1 = (0x30000 | 510),
  LLGL::ShadingLanguage::HLSL_6_0 = (0x30000 | 600), LLGL::ShadingLanguage::HLSL_6_1 = (0x30000 | 601), LLGL::ShadingLanguage::HLSL_6_2 = (0x30000 | 602), LLGL::ShadingLanguage::HLSL_6_3 = (0x30000 | 603),
  LLGL::ShadingLanguage::HLSL_6_4 = (0x30000 | 604), LLGL::ShadingLanguage::Metal = (0x40000), LLGL::ShadingLanguage::Metal_1_0 = (0x40000 | 100), LLGL::ShadingLanguage::Metal_1_1 = (0x40000 | 110),
  LLGL::ShadingLanguage::Metal_1_2 = (0x40000 | 120), LLGL::ShadingLanguage::Metal_2_0 = (0x40000 | 200), LLGL::ShadingLanguage::Metal_2_1 = (0x40000 | 210), LLGL::ShadingLanguage::SPIRV = (0x50000), LLGL::ShadingLanguage::SPIRV_100 = (0x50000 | 100), LLGL::ShadingLanguage::VersionBitmask = 0x0000ffff }

  Shading language version enumation.
- enum LLGL::ScreenOrigin { LLGL::ScreenOrigin::LowerLeft, LLGL::ScreenOrigin::UpperLeft }

  Screen coordinate system origin enumeration.
- enum LLGL::ClippingRange { LLGL::ClippingRange::MinusOneToOne, LLGL::ClippingRange::ZeroToOne }

  Clipping depth range enumeration.
- enum LLGL::CPUAccess { LLGL::CPUAccess::ReadOnly, LLGL::CPUAccess::WriteOnly, LLGL::CPUAccess::WriteDiscard, LLGL::CPUAccess::ReadWrite }

  Classifications of CPU access to mapped resources.

Functions

- LLGL_EXPORT bool LLGL::ValidateRenderingCaps (const RenderingCapabilities &presentCaps, const RenderingCapabilities &requiredCaps, const ValidateRenderingCapsFunc &callback={})

  Validates the presence of the specified required rendering capabilities.

11.53 RenderTarget.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/RenderTargetFlags.h>
#include <LLGL/Types.h>
```

Classes

- class LLGL::RenderTarget

  Render target interface.

Namespaces

- LLGL

11.54 RenderTargetFlags.h File Reference

```
#include <LLGL/TextureFlags.h>
#include <LLGL/ForwardDecls.h>
#include <LLGL/PipelineStateFlags.h>
#include <vector>
#include <cstdint>
```

Classes

- struct LLGL::AttachmentDescriptor

  Render target attachment descriptor structure.
- struct LLGL::RenderTargetDescriptor

  Render target descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::AttachmentType { LLGL::AttachmentType::Color, LLGL::AttachmentType::Depth, LLGL::↩ AttachmentType::DepthStencil, LLGL::AttachmentType::Stencil }

  Render target attachment type enumeration.

11.55 Report.h File Reference

```
#include <LLGL/Interface.h>
```

Classes

- class LLGL::Report

  Error and warning report interface.

Namespaces

- LLGL

11.56 Resource.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/ResourceFlags.h>
```

Classes

- class LLGL::Resource

  Base class for all hardware resource interfaces.

Namespaces

- LLGL

11.57 ResourceFlags.h File Reference

Classes

- struct LLGL::BindFlags

  Flags for Buffer and Texture resources that describe for which purposes they will be used.
- struct LLGL::CPUAccessFlags

  CPU read/write access flag enumeration for mapping GPU into CPU memory space.
- struct LLGL::MiscFlags

  Miscellaneous resource flags.

Namespaces

- LLGL

Enumerations

- enum LLGL::ResourceType { LLGL::ResourceType::Undefined, LLGL::ResourceType::Buffer, LLGL::↩ ResourceType::Texture, LLGL::ResourceType::Sampler }

  Hardware resource type enumeration.

11.58 ResourceHeap.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <cstdint>
```

Classes

- class LLGL::ResourceHeap

  Resource heap interface.

Namespaces

- LLGL

11.59 ResourceHeapFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Texture.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/Buffer.h>
#include <LLGL/BufferFlags.h>
#include <vector>
```

Classes

- struct LLGL::BarrierFlags

  Flags for memory barriers in resource heaps.
- struct LLGL::ResourceViewDescriptor

  Resource view descriptor structure.
- struct LLGL::ResourceHeapDescriptor

  Resource heap descriptor structure.

Namespaces

- LLGL

11.60 Sampler.h File Reference

```
#include <LLGL/Resource.h>
#include <LLGL/SamplerFlags.h>
```

Classes

- class LLGL::Sampler

  Sampler interface.

Namespaces

- LLGL

11.61 SamplerFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/PipelineStateFlags.h>
#include <LLGL/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.

11.62 Shader.h File Reference

```
#include <LLGL/RenderSystemChild.h>
#include <LLGL/ShaderFlags.h>
#include <LLGL/ShaderReflection.h>
#include <LLGL/Report.h>
```

Classes

- class LLGL::Shader

  Shader interface.

Namespaces

- LLGL

11.63 ShaderFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Types.h>
#include <LLGL/VertexAttribute.h>
#include <LLGL/FragmentAttribute.h>
#include <cstddef>
#include <vector>
```

Classes

- struct LLGL::ShaderCompileFlags

  Shader compilation flags enumeration.
- struct LLGL::StageFlags

  Shader stage flags enumeration.
- struct LLGL::ShaderMacro

  Shader macro structure with name and optional body definition.
- struct LLGL::VertexShaderAttributes

  Vertex (or geometry) shader specific structure.
- struct LLGL::FragmentShaderAttributes

  Fragment shader specific descriptor structure.
- struct LLGL::ComputeShaderAttributes

  Compute shader specific descriptor structure.
- struct LLGL::ShaderDescriptor

  Shader source and binary code descriptor structure.

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.

11.64 ShaderReflection.h File Reference

```
#include <LLGL/ForwardDecls.h>
#include <LLGL/ShaderFlags.h>
#include <LLGL/ResourceFlags.h>
#include <LLGL/BufferFlags.h>
#include <LLGL/PipelineLayoutFlags.h>
#include <vector>
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::ShaderResource

  Shader reflection resource structure.

- struct LLGL::ShaderUniform

  Shader reflection uniform structure.

- struct LLGL::ShaderReflection

  Shader reflection structure.

Namespaces

- LLGL

Typedefs

- using LLGL::UniformLocation = std::int32_t

  Shader uniform location type, as zero-based index in 32-bit signed integer format.

Enumerations

- enum LLGL::UniformType {
  LLGL::UniformType::Undefined, LLGL::UniformType::Float1, LLGL::UniformType::Float2, LLGL::Uniform↩Type::Float3,
  LLGL::UniformType::Float4, LLGL::UniformType::Double1, LLGL::UniformType::Double2, LLGL::Uniform↩Type::Double3,
  LLGL::UniformType::Double4, LLGL::UniformType::Int1, LLGL::UniformType::Int2, LLGL::UniformType::Int3,
  LLGL::UniformType::Int4, LLGL::UniformType::UInt1, LLGL::UniformType::UInt2, LLGL::UniformType::UInt3,
  LLGL::UniformType::UInt4, LLGL::UniformType::Bool1, LLGL::UniformType::Bool2, LLGL::UniformType::↩Bool3,
  LLGL::UniformType::Bool4, LLGL::UniformType::Float2x2, LLGL::UniformType::Float2x3, LLGL::Uniform↩Type::Float2x4,
  LLGL::UniformType::Float3x2, LLGL::UniformType::Float3x3, LLGL::UniformType::Float3x4, LLGL::↩UniformType::Float4x2,
  LLGL::UniformType::Float4x3, LLGL::UniformType::Float4x4, LLGL::UniformType::Double2x2, LLGL::↩UniformType::Double2x3,
  LLGL::UniformType::Double2x4, LLGL::UniformType::Double3x2, LLGL::UniformType::Double3x3, LLGL::↩UniformType::Double3x4,
  LLGL::UniformType::Double4x2, LLGL::UniformType::Double4x3, LLGL::UniformType::Double4x4, LLGL::↩UniformType::Sampler,
  LLGL::UniformType::Image, LLGL::UniformType::AtomicCounter }

  Shader uniform type enumeration.
- enum LLGL::StorageBufferType {
  LLGL::StorageBufferType::Undefined, LLGL::StorageBufferType::TypedBuffer, LLGL::StorageBufferType::↩StructuredBuffer, LLGL::StorageBufferType::ByteAddressBuffer,
  LLGL::StorageBufferType::RWTypedBuffer, LLGL::StorageBufferType::RWStructuredBuffer, LLGL::↩StorageBufferType::RWByteAddressBuffer, LLGL::StorageBufferType::AppendStructuredBuffer,
  LLGL::StorageBufferType::ConsumeStructuredBuffer }

  Storage buffer type enumeration for shader reflection.

11.65 SmallVector.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Container/ArrayView.h>
#include <memory>
#include <cstddef>
#include <iterator>
#include <type_traits>
#include <algorithm>
#include <initializer_list>
```

Classes

- struct LLGL::GrowStrategyAddHalf

  Container capacity grow strategy to increase the capacity to 150%.
- struct LLGL::GrowStrategyDouble

  Container capacity grow strategy to increase the capacity to 200%.
- struct LLGL::GrowStrategyRoundUpPow2

  Container capacity grow strategy to increase the capacity to the next higher power of two.
- class LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >

  Generic container class for consecutive arrays optimized for small sizes.

Namespaces

- LLGL

11.66 StaticLimits.h File Reference

Macros

- #define LLGL_MAX_NUM_COLOR_ATTACHMENTS (8u)

  Maximum number of color attachments allowed for render targets.
- #define LLGL_MAX_NUM_ATTACHMENTS ((LLGL_MAX_NUM_COLOR_ATTACHMENTS) + 1u)

  Maximum number of attachments allowed for render targets (color attachments and depth-stencil attachment).
- #define LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS (16u)

  Maximum number of viewports and scissors.
- #define LLGL_MAX_NUM_SAMPLES (64u)

  Maximum number of samples for multi-sampled rendering.
- #define LLGL_MAX_NUM_SO_BUFFERS (4u)

  Maximum number of stream-output buffers.

11.66.1 Macro Definition Documentation

11.66.1.1 #define LLGL_MAX_NUM_ATTACHMENTS ((LLGL_MAX_NUM_COLOR_ATTACHMENTS) + 1u)

Maximum number of attachments allowed for render targets (color attachments and depth-stencil attachment).

11.66.1.2 #define LLGL_MAX_NUM_COLOR_ATTACHMENTS (8u)

Maximum number of color attachments allowed for render targets.

11.66.1.3 #define LLGL_MAX_NUM_SAMPLES (64u)

Maximum number of samples for multi-sampled rendering.

11.66.1.4 #define LLGL_MAX_NUM_SO_BUFFERS (4u)

Maximum number of stream-output buffers.

11.66.1.5 #define LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS (16u)

Maximum number of viewports and scissors.

11.67 Strings.h File Reference

```
#include <LLGL/Container/UTF8String.h>
#include <LLGL/Container/StringView.h>
```

Namespaces

- LLGL

Functions

- UTF8String LLGL::operator+ (const UTF8String &lhs, const UTF8String &rhs)

  Concatenates the two UTF-8 string.
- UTF8String LLGL::operator+ (const UTF8String &lhs, const StringView &rhs)
- UTF8String LLGL::operator+ (const UTF8String &lhs, const WStringView &rhs)
- UTF8String LLGL::operator+ (const UTF8String &lhs, const char *rhs)
- UTF8String LLGL::operator+ (const UTF8String &lhs, const wchar_t *rhs)
- UTF8String LLGL::operator+ (const StringView &lhs, const UTF8String &rhs)
- UTF8String LLGL::operator+ (const WStringView &lhs, const UTF8String &rhs)
- UTF8String LLGL::operator+ (const char *lhs, const UTF8String &rhs)
- UTF8String LLGL::operator+ (const wchar_t *lhs, const UTF8String &rhs)
- bool LLGL::operator== (const UTF8String &lhs, const UTF8String &rhs)

  Returns true if the two strings are equal.
- bool LLGL::operator== (const UTF8String &lhs, const StringView &rhs)
- bool LLGL::operator== (const UTF8String &lhs, const WStringView &rhs)
- bool LLGL::operator== (const UTF8String &lhs, const char *rhs)
- bool LLGL::operator== (const UTF8String &lhs, const wchar_t *rhs)
- bool LLGL::operator== (const StringView &lhs, const StringView &rhs)
- bool LLGL::operator== (const StringView &lhs, const UTF8String &rhs)
- bool LLGL::operator== (const StringView &lhs, const char *rhs)
- bool LLGL::operator== (const WStringView &lhs, const WStringView &rhs)
- bool LLGL::operator== (const WStringView &lhs, const UTF8String &rhs)
- bool LLGL::operator== (const WStringView &lhs, const wchar_t *rhs)
- bool LLGL::operator== (const char *lhs, const UTF8String &rhs)
- bool LLGL::operator== (const char *lhs, const StringView &rhs)
- bool LLGL::operator== (const wchar_t *lhs, const UTF8String &rhs)
- bool LLGL::operator== (const wchar_t *lhs, const WStringView &rhs)
- bool LLGL::operator!= (const UTF8String &lhs, const UTF8String &rhs)

  Returns true if the two strings are not equal.
- bool LLGL::operator!= (const UTF8String &lhs, const StringView &rhs)
- bool LLGL::operator!= (const UTF8String &lhs, const WStringView &rhs)
- bool LLGL::operator!= (const UTF8String &lhs, const char *rhs)
- bool LLGL::operator!= (const UTF8String &lhs, const wchar_t *rhs)
- bool LLGL::operator!= (const StringView &lhs, const StringView &rhs)
- bool LLGL::operator!= (const StringView &lhs, const UTF8String &rhs)
- bool LLGL::operator!= (const StringView &lhs, const char *rhs)
- bool LLGL::operator!= (const WStringView &lhs, const WStringView &rhs)
- bool LLGL::operator!= (const WStringView &lhs, const UTF8String &rhs)
- bool LLGL::operator!= (const WStringView &lhs, const wchar_t *rhs)
- bool LLGL::operator!= (const char *lhs, const UTF8String &rhs)
- bool LLGL::operator!= (const char *lhs, const StringView &rhs)
- bool LLGL::operator!= (const wchar_t *lhs, const UTF8String &rhs)
- bool LLGL::operator!= (const wchar_t *lhs, const WStringView &rhs)

11.68 StringView.h File Reference

```
#include <LLGL/Export.h>
#include <string>
#include <cstddef>
#include <iterator>
#include <algorithm>
#include <stdexcept>
```

Classes

- class LLGL::BasicStringView< TChar, Traits >

  Constant string view container template. Keeps pointer to string and its length.

Namespaces

- LLGL

Typedefs

- using LLGL::StringView = BasicStringView< char >

  ANSI character string view (char).
- using LLGL::WStringView = BasicStringView< wchar_t >

  Wide character string view (wchar_t).

11.69 Surface.h File Reference

```
#include <LLGL/Interface.h>
#include <LLGL/Types.h>
#include <LLGL/SwapChainFlags.h>
#include <LLGL/Display.h>
```

Classes

- class LLGL::Surface

  The Surface interface is the base interface for Window (on Desktop platforms) and Canvas (on mobile platforms).

Namespaces

- LLGL

11.70 SwapChain.h File Reference

```
#include <LLGL/RenderTarget.h>
#include <LLGL/SwapChainFlags.h>
#include <LLGL/RenderSystemFlags.h>
#include <LLGL/Surface.h>
#include <LLGL/Buffer.h>
#include <LLGL/BufferArray.h>
#include <LLGL/Texture.h>
#include <LLGL/PipelineState.h>
#include <LLGL/Sampler.h>
#include <LLGL/QueryHeap.h>
#include <string>
#include <memory>
#include <map>
```

Classes

- class LLGL::SwapChain

  Swap-chain interface.

Namespaces

- LLGL

11.71 SwapChainFlags.h File Reference

```
#include <LLGL/Types.h>
#include <cstdint>
```

Classes

- struct LLGL::ResizeBuffersFlags

  Swap-chain resize buffers flags.
- struct LLGL::SwapChainDescriptor

  Swap chain descriptor structure.

Namespaces

- LLGL

11.72 SystemValue.h File Reference

Namespaces

- LLGL

Enumerations

- enum LLGL::SystemValue {
  LLGL::SystemValue::Undefined, LLGL::SystemValue::ClipDistance, LLGL::SystemValue::Color, LLGL::↩
  SystemValue::CullDistance,
  LLGL::SystemValue::Depth, LLGL::SystemValue::DepthGreater, LLGL::SystemValue::DepthLess, LLGL::↩
  SystemValue::FrontFacing,
  LLGL::SystemValue::InstanceID, LLGL::SystemValue::Position, LLGL::SystemValue::PrimitiveID, LLGL::↩
  SystemValue::RenderTargetIndex,
  LLGL::SystemValue::SampleMask, LLGL::SystemValue::SampleID, LLGL::SystemValue::Stencil, LLGL::↩
  SystemValue::VertexID,
  LLGL::SystemValue::ViewportIndex }

  Enumeration of system values for vertex input or fragment output attributes.

11.73 Tags.h File Reference

Classes

- struct LLGL::UninitializeTag

  Common uninitialize tag.

Namespaces

- LLGL

11.74 Texture.h File Reference

```
#include <LLGL/Resource.h>
#include <LLGL/Types.h>
#include <LLGL/TextureFlags.h>
#include <cstdint>
```

Classes

- class LLGL::Texture

  Texture interface.

Namespaces

- LLGL

11.75 TextureFlags.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Types.h>
#include <LLGL/Format.h>
#include <LLGL/ResourceFlags.h>
#include <LLGL/CommandBufferFlags.h>
#include <cstddef>
#include <cstdint>
```

Classes

- struct LLGL::TextureSwizzleRGBA

  Texture component swizzle structure for red, green, blue, and alpha components.
- struct LLGL::TextureSubresource

  Texture subresource descriptor which specifies the array layer and MIP-map level range of a texture resource.
- struct LLGL::TextureLocation

  Texture location structure: MIP-map level and offset.
- struct LLGL::TextureRegion

  Texture region structure: Subresource (MIP-map level and array layer range), offset, and extent.
- struct LLGL::TextureDescriptor

  Texture descriptor structure.
- struct LLGL::TextureViewDescriptor

  Texture view descriptor structure.

Namespaces

- LLGL

Enumerations

- enum LLGL::TextureType {
  LLGL::TextureType::Texture1D, LLGL::TextureType::Texture2D, LLGL::TextureType::Texture3D, LLGL::↩
  TextureType::TextureCube,
  LLGL::TextureType::Texture1DArray, LLGL::TextureType::Texture2DArray, LLGL::TextureType::Texture↩
  CubeArray, LLGL::TextureType::Texture2DMS,
  LLGL::TextureType::Texture2DMSArray }

  Texture type enumeration.
- enum LLGL::TextureSwizzle : std::uint8_t {
  LLGL::TextureSwizzle::Zero, LLGL::TextureSwizzle::One, LLGL::TextureSwizzle::Red, LLGL::Texture↩
  Swizzle::Green,
  LLGL::TextureSwizzle::Blue, LLGL::TextureSwizzle::Alpha }

  Texture component swizzle enumeration.

Functions

- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (std::uint32_t width, std::uint32_t height=1, std::uint32_t depth=1)

  Returns the number of MIP-map levels for a texture with the specified size.
- LLGL_EXPORT std::uint32_t LLGL::NumMipLevels (const TextureType type, const Extent3D &extent)

  Returns the number of MIP-map levels for the specified texture attributes.
- 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::NumMipTexels (const TextureType type, const Extent3D &extent, std↩::uint32_t mipLevel)

  Returns the number of texture elements (texels) for the specified texture attributes, or zero extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureType type, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the number of texture elements (texels) for the specified texture subresource range.
- LLGL_EXPORT std::uint32_t LLGL::NumMipTexels (const TextureDescriptor &textureDesc, std::uint32_↩t mipLevel=~0u)

  Returns the number of texture elements (texels) for the specified texture descriptor.
- LLGL_EXPORT std::uint32_t LLGL::NumMipDimensions (const TextureType type)

  Returns the number of MIP-map dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.
- LLGL_EXPORT std::uint32_t LLGL::NumTextureDimensions (const TextureType type)

  Returns the number of texture dimensions for the specified texture type. This is either 1, 2, 3, or 0 if the input is invalid.
- LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureType type, const Extent3D &extent, std↩::uint32_t mipLevel)

  Returns the MIP-map extent (including array layers) for the specified texture type, or an empty extent if `mipLevel` is out of bounds (see `NumMipLevels`).
- LLGL_EXPORT Extent3D LLGL::GetMipExtent (const TextureDescriptor &textureDesc, std::uint32_t mip↩Level=0)

  Returns the MIP-map extent (including array layers) for the specified texture descriptor.
- std::uint32_t LLGL::GetMemoryFootprint (const TextureType type, const Format format, const Extent3D &extent, const TextureSubresource &subresource)

  Returns the memory footprint (in bytes) of a texture subresource with the specified hardware format and extent.
- LLGL_EXPORT bool LLGL::IsMipMappedTexture (const TextureDescriptor &textureDesc)

  Returns true if the specified texture descriptor describes a texture with MIP-mapping enabled.
- LLGL_EXPORT bool LLGL::IsArrayTexture (const TextureType type)

  Returns true if the specified texture type is an array texture.
- LLGL_EXPORT bool LLGL::IsMultiSampleTexture (const TextureType type)

  Returns true if the specified texture type is a multi-sample texture.
- LLGL_EXPORT bool LLGL::IsCubeTexture (const TextureType type)

  Returns true if the specified texture type is a cube texture.
- LLGL_EXPORT bool LLGL::IsTextureSwizzleIdentity (const TextureSwizzleRGBA &swizzle)

  Returns true if the specified texture swizzling is equal to the identity mapping.

11.76 Timer.h File Reference

```
#include <LLGL/Export.h>
#include <cstdint>
```

Namespaces

- LLGL
- LLGL::Timer

Functions

- LLGL_EXPORT std::uint64_t LLGL::Timer::Frequency ()

  Returns the timer frequency of the OS. This is the number of ticks per seconds, e.g. 1,000,000 for microseconds.
- LLGL_EXPORT std::uint64_t LLGL::Timer::Tick ()

  Returns the current 'tick' of a high resolution timer that is OS dependent. The tick frequency can be queried with Frequency.

11.77 TypeInfo.h File Reference

```
#include <LLGL/Interface.h>
```

Namespaces

- LLGL

Functions

- template<typename T >
  bool LLGL::IsInstanceOf (const Interface *obj)

  Returns whether the specified object is an instance of the interface specified by template parameter <T>.
- template<typename T >
  bool LLGL::IsInstanceOf (const Interface &obj)

  Const-reference version of IsInstanceOf.
- template<typename T >
  const T * LLGL::CastTo (const Interface *obj)

  Returns a constant pointer to this instance of type `const T*` if it implements the specified interface.
- template<typename T >
  T * LLGL::CastTo (Interface *obj)

  Returns a pointer to this instance of type `T*` if it implements the specified interface.
- template<typename T >
  const T & LLGL::CastTo (const Interface &obj)

  Returns a constant reference to this instance of type `const T&` if it implements the specified interface.
- template<typename T >
  T & LLGL::CastTo (Interface &obj)

  Returns a reference to this instance of type `T&` if it implements the specified interface.

11.78 TypeNames.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/ShaderFlags.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/RenderingDebugger.h>
#include <LLGL/RenderSystemFlags.h>
```

Namespaces

- LLGL

Functions

- LLGL_EXPORT const char * LLGL::ToString (const ShaderType t)

  Returns a string representation for the spcified ShaderType value, or null if the input type is invalid.
- LLGL_EXPORT const char * LLGL::ToString (const ErrorType t)

  Returns a string representation for the specified ErrorType value, or null if the input type is invalid.
- LLGL_EXPORT const char * LLGL::ToString (const WarningType t)

  Returns a string representation for the specified WarningType value, or null if the input type is invalid.
- LLGL_EXPORT const char * LLGL::ToString (const ShadingLanguage t)

  Returns a string representation for the specified ShadingLanguage value, or null if the input type is invalid.
- LLGL_EXPORT const char * LLGL::ToString (const Format t)

  Returns a string representation for the specified Format value, or null if the input type is invalid.
- LLGL_EXPORT const char * LLGL::ToString (const TextureType t)

  Returns a string representation for the specified TextureType value, or null if the input type is invalid.

11.79 Types.h File Reference

```
#include <LLGL/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'.

11.80 UTF8String.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Container/SmallVector.h>
#include <LLGL/Container/StringView.h>
```

Classes

- class LLGL::UTF8String

  Container class for UTF-8 encoded strings.

Namespaces

- LLGL

11.81 Utility.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/ForwardDecls.h>
#include <LLGL/TextureFlags.h>
#include <LLGL/BufferFlags.h>
#include <LLGL/RenderTargetFlags.h>
#include <LLGL/RenderPassFlags.h>
#include <LLGL/ResourceHeapFlags.h>
#include <LLGL/ShaderFlags.h>
#include <LLGL/ShaderReflection.h>
#include <LLGL/PipelineLayoutFlags.h>
```

Namespaces

- LLGL

Functions

- LLGL_EXPORT TextureDescriptor LLGL::Texture1DDesc (Format format, std::uint32_t width, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture1D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture2D type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture3DDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t depth, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture3D type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeDesc (Format format, std::uint32_t width, std::uint32_t height, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::TextureCube type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture1DArrayDesc (Format format, std::uint32_t width, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture1DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture2DArray type.
- LLGL_EXPORT TextureDescriptor LLGL::TextureCubeArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::TextureCubeArray type.
- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture2DMS type.

- LLGL_EXPORT TextureDescriptor LLGL::Texture2DMSArrayDesc (Format format, std::uint32_t width, std::uint32_t height, std::uint32_t arrayLayers, std::uint32_t samples, long bindFlags=(BindFlags::ColorAttachment|BindFlags::Sampled))

  Returns a TextureDescriptor structure with the TextureType::Texture2DMSArray type.
- LLGL_EXPORT BufferDescriptor LLGL::VertexBufferDesc (uint64_t size, const VertexFormat &vertexFormat, long cpuAccessFlags=0)

  Returns a BufferDescriptor structure for a vertex buffer.
- LLGL_EXPORT BufferDescriptor LLGL::IndexBufferDesc (uint64_t size, const Format format, long cpuAccessFlags=0)

  Returns a BufferDescriptor structure for an index buffer.
- LLGL_EXPORT BufferDescriptor LLGL::ConstantBufferDesc (uint64_t size, long cpuAccessFlags=0)

  Returns a BufferDescriptor structure for a constant buffer.
- LLGL_EXPORT BufferDescriptor LLGL::StorageBufferDesc (uint64_t size, const StorageBufferType storageType, std::uint32_t stride, long cpuAccessFlags=CPUAccessFlags::ReadWrite)

  Returns a BufferDescriptor structure for a storage buffer.
- LLGL_EXPORT ShaderDescriptor LLGL::ShaderDescFromFile (const ShaderType type, const char *filename, const char *entryPoint=nullptr, const char *profile=nullptr, long flags=0)

  Returns a ShaderDescriptor structure.
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const ShaderReflection &reflection)

  Converts the specified shader reflection descriptor into a pipeline layout descriptor.
- LLGL_EXPORT PipelineLayoutDescriptor LLGL::PipelineLayoutDesc (const char *layoutSignature)

  Generates a pipeline layout descriptor by parsing the specified string.
- LLGL_EXPORT RenderPassDescriptor LLGL::RenderPassDesc (const RenderTargetDescriptor &renderTargetDesc)

  Converts the specified render target descriptor into a render pass descriptor with default settings.

11.82 Version.h File Reference

```
#include <LLGL/Export.h>
```

Namespaces

- LLGL
- LLGL::Version

  Namespace with functions to determine LLGL version.

Functions

- LLGL_EXPORT unsigned LLGL::Version::GetMajor ()

  Returns the major LLGL version (e.g. 1 stands for "1.00").
- LLGL_EXPORT unsigned LLGL::Version::GetMinor ()

  Returns the minor LLGL version (e.g. 1 stands for "0.01"). Must be less than 100.
- LLGL_EXPORT unsigned LLGL::Version::GetRevision ()

  Returns the revision version number. Must be less than 100.
- LLGL_EXPORT const char * LLGL::Version::GetStatus ()

  Returns the LLGL version status (either "Alpha", "Beta", or empty).
- LLGL_EXPORT unsigned LLGL::Version::GetID ()

  Returns the full LLGL version as an ID number (e.g. 200317 stands for "2.03 (Rev. 17)").
- LLGL_EXPORT const char * LLGL::Version::GetString ()

  Returns the full LLGL version as a string (e.g. "0.01 Beta (Rev. 1)").

11.83 VertexAttribute.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Format.h>
#include <LLGL/SystemValue.h>
#include <LLGL/Container/Strings.h>
#include <cstdint>
```

Classes

- struct LLGL::VertexAttribute

  Vertex input/output 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).

11.84 VertexFormat.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/Constants.h>
#include <LLGL/VertexAttribute.h>
#include <vector>
#include <cstdint>
#include <algorithm>
```

Classes

- struct LLGL::VertexFormat

  Utility structure to store vertex attributes.

Namespaces

- LLGL

11.85 VideoAdapter.h File Reference

```
#include <LLGL/Export.h>
#include <LLGL/DisplayFlags.h>
#include <vector>
#include <string>
#include <cstdint>
```

Classes

- struct LLGL::VideoOutputDescriptor

  Video output structure.
- struct LLGL::VideoAdapterDescriptor

  Video adapter descriptor structure.

Namespaces

- LLGL

11.86 Win32NativeHandle.h File Reference

```
#include <Windows.h>
```

Classes

- struct LLGL::NativeHandle

  Android native handle structure.
- struct LLGL::NativeContextHandle

  Android native context handle structure.

Namespaces

- LLGL

11.87 Window.h File Reference

```
#include <LLGL/Container/Strings.h>
#include <LLGL/Surface.h>
#include <LLGL/WindowFlags.h>
#include <LLGL/Key.h>
#include <memory>
```

Classes

- class LLGL::Window

  Window interface for desktop platforms.
- class LLGL::Window::EventListener

  Interface for all window event listeners.

Namespaces

- LLGL

11.88 WindowFlags.h File Reference

```
#include <LLGL/Types.h>
#include <LLGL/Constants.h>
#include <LLGL/Container/Strings.h>
#include <cstdint>
```

Classes

- struct LLGL::WindowDescriptor

  Window descriptor structure.
- struct LLGL::WindowBehavior

  Window behavior structure.

Namespaces

- LLGL

Index

~Canvas
LLGL::Canvas, 158
~Input
LLGL::Input, 270
~NonCopyable
LLGL::NonCopyable, 285
~RenderSystem
LLGL::RenderSystem, 331
~RenderingDebugger
LLGL::RenderingDebugger, 308
~SmallVector
LLGL::SmallVector, 390
~SwapChain
LLGL::SwapChain, 405
~Window
LLGL::Window, 451

A
LLGL, 86
a
LLGL::Color< T, 4u >, 176
LLGL::TextureSwizzleRGBA, 428
A8UNorm
LLGL, 81
ABGR
LLGL, 84
ANDInverted
LLGL, 89
ANDReverse
LLGL, 89
AND
LLGL, 89
ARGB
LLGL, 84
acceptDropFiles
LLGL::WindowDescriptor, 459
Accumulate
LLGL::FrameProfile, 246
LLGL::RenderingProfiler, 324
AdaptForVideoMode
LLGL::Canvas, 158
LLGL::Surface, 401
LLGL::Window, 452
AdaptSurface
LLGL::ResizeBuffersFlags, 358
Add
LLGL, 75
AddEventListener
LLGL::Canvas, 158
LLGL::Window, 452
addressModeU
LLGL::SamplerDescriptor, 367
addressModeV
LLGL::SamplerDescriptor, 367
addressModeW
LLGL::SamplerDescriptor, 368
All
LLGL::ClearFlags, 161
AllGraphicsStages
LLGL::StageFlags, 396
AllStages
LLGL::StageFlags, 396
AllTessStages
LLGL::StageFlags, 396
AllocateByteBuffer
Image utility functions to classify and convert image data., 33
Alpha
LLGL, 84, 101
alphaArithmetic
LLGL::BlendTargetDescriptor, 144
alphaToCoverageEnabled
LLGL::BlendDescriptor, 141
AlwaysPass
LLGL, 77
AndroidNativeHandle.h, 461
annotation
LLGL::ProfileTimeRecord, 291
antiAliasedLineEnabled
LLGL::RasterizerDescriptor, 296
Any
LLGL, 88
AnySamplesPassed
LLGL, 92
AnySamplesPassedConservative
LLGL, 92
Append
LLGL::MiscFlags, 280
AppendAttribute
LLGL::VertexFormat, 441
AppendStructuredBuffer
LLGL, 99
application
LLGL::RendererConfigurationVulkan, 301
applicationName
LLGL::ApplicationDescriptor, 119
applicationVersion
LLGL::ApplicationDescriptor, 119
Apps

LLGL, 86
arrayLayer
LLGL::AttachmentDescriptor, 127
LLGL::TextureLocation, 422
arrayLayers
LLGL::TextureDescriptor, 418
arraySize
LLGL::BindingDescriptor, 139
ArrayView
LLGL::ArrayView, 122
ArrayView.h, 461
AssertCreateBuffer
LLGL::RenderSystem, 331
AssertCreateBufferArray
LLGL::RenderSystem, 331
AssertCreateRenderTarget
LLGL::RenderSystem, 331
AssertCreateShader
LLGL::RenderSystem, 331
AssertImageDataSize
LLGL::RenderSystem, 331
at
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::UTF8String, 434
AtomicCounter
LLGL, 103
AttachmentClear
LLGL::AttachmentClear, 124, 125
attachmentClears
LLGL::FrameProfile, 246
AttachmentDescriptor
LLGL::AttachmentDescriptor, 126
AttachmentFormatDescriptor
LLGL::AttachmentFormatDescriptor, 129
AttachmentLoadOp
LLGL, 74
AttachmentStoreOp
LLGL, 74
AttachmentType
LLGL, 74
attachments
LLGL::RenderTargetDescriptor, 355
Attn
LLGL, 88
attributes
LLGL::VertexFormat, 443

B
LLGL, 86
b
LLGL::Color< T, 3u >, 171
LLGL::Color< T, 4u >, 176
LLGL::TextureSwizzleRGBA, 428
BC1
LLGL, 84
BC1UNorm
LLGL, 83
BC1UNorm_sRGB
LLGL, 83
BC2
LLGL, 84
BC2UNorm
LLGL, 83
BC2UNorm_sRGB
LLGL, 83
BC3
LLGL, 84
BC3UNorm
LLGL, 84
BC3UNorm_sRGB
LLGL, 84
BC4
LLGL, 84
BC4SNorm
LLGL, 84
BC4UNorm
LLGL, 84
BC5
LLGL, 84
BC5SNorm
LLGL, 84
BC5UNorm
LLGL, 84
BGRA8SInt
LLGL, 83
BGRA8SNorm
LLGL, 83
BGRA8UInt
LLGL, 83
BGRA8UNorm
LLGL, 83
BGRA8UNorm_sRGB
LLGL, 83
BGRA
LLGL, 84
BGR
LLGL, 84
Back
LLGL, 78, 85, 97
back
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::StencilDescriptor, 397
LLGL::UTF8String, 434
barrierFlags
LLGL::ResourceHeapDescriptor, 362
baseArrayLayer
LLGL::TextureSubresource, 426
baseMipLevel
LLGL::TextureSubresource, 426
BasicStringView
LLGL::BasicStringView, 133, 134
Begin
LLGL::CommandBuffer, 180

begin
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::UTF8String, 434
BeginQuery
LLGL::CommandBuffer, 180
BeginRenderCondition
LLGL::CommandBuffer, 181
BeginRenderPass
LLGL::CommandBuffer, 181
BeginStreamOutput
LLGL::CommandBuffer, 182
BinaryBuffer
LLGL, 95
BinaryFile
LLGL, 95
bindFlags
LLGL::BindingDescriptor, 139
LLGL::BufferDescriptor, 152
LLGL::TextureDescriptor, 419
binding
LLGL::ShaderResource, 385
BindingDescriptor
LLGL::BindingDescriptor, 139
bindings
LLGL::PipelineLayoutDescriptor, 289
bitSize
LLGL::FormatAttributes, 238
blend
LLGL::GraphicsPipelineDescriptor, 254
BlendArithmetic
LLGL, 75
blendEnabled
LLGL::BlendTargetDescriptor, 144
BlendFactor
LLGL, 76
blendFactor
LLGL::BlendDescriptor, 141
blendFactorDynamic
LLGL::BlendDescriptor, 142
BlendOp
LLGL, 75
Blit
LLGL::Image, 262
Blob.h, 462
Block
LLGL::RenderingDebugger::Message, 277
BlockAfter
LLGL::RenderingDebugger::Message, 277
blockHeight
LLGL::FormatAttributes, 238
blockWidth
LLGL::FormatAttributes, 238
Blue
LLGL, 101
Bool1
LLGL, 103
Bool2
LLGL, 103
Bool3
LLGL, 103
Bool4
LLGL, 103
Border
LLGL, 94
borderColor
LLGL::SamplerDescriptor, 368
borderless
LLGL::CanvasDescriptor, 160
LLGL::WindowDescriptor, 459
BrowserBack
LLGL, 87
BrowserFavorits
LLGL, 87
BrowserForward
LLGL, 87
BrowserHome
LLGL, 87
BrowserRefresh
LLGL, 87
BrowserSearch
LLGL, 87
BrowserStop
LLGL, 87
Buffer
LLGL::Buffer, 149
LLGL::InterfaceID, 274
LLGL, 93
Buffer argument structures for indirect draw and compute commands., 38
Buffer utility functions to determine buffer types., 27
IsByteAddressBuffer, 27
IsStructuredBuffer, 27
IsTypedBuffer, 27
Buffer.h, 462
BufferArray
LLGL::BufferArray, 151
LLGL::InterfaceID, 274
BufferArray.h, 462
bufferCopies
LLGL::FrameProfile, 246
bufferFills
LLGL::FrameProfile, 247
BufferFlags.h, 463
bufferMappings
LLGL::FrameProfile, 247
bufferReads
LLGL::FrameProfile, 247
bufferUpdates
LLGL::FrameProfile, 247
bufferView
LLGL::ResourceViewDescriptor, 364
BufferViewDescriptor
LLGL::BufferViewDescriptor, 155
bufferWrites

LLGL::FrameProfile, 247
ByRegionNoWait
LLGL, 92
ByRegionNoWaitInverted
LLGL, 92
ByRegionWait
LLGL, 92
ByRegionWaitInverted
LLGL, 92
ByteAddressBuffer
LLGL, 98
ByteBuffer
LLGL, 72

C
LLGL, 86
c_str
LLGL::UTF8String, 434
CPUAccess
LLGL, 77
Cancel
LLGL, 85
Canvas
LLGL::Canvas, 158
LLGL::Canvas::EventListener, 234
LLGL::InterfaceID, 274
Canvas.h, 463
Canvas_EventListener
LLGL::InterfaceID, 274
CanvasFlags.h, 464
capacity
LLGL::SmallVector, 390
LLGL::UTF8String, 434
Capital
LLGL, 85
Cast
LLGL::Color, 165
LLGL::Color< T, 3u >, 169
LLGL::Color< T, 4u >, 174
CastColorValue
LLGL, 104
CastColorValue< bool, bool >
LLGL, 104
CastColorValue< double, double >
LLGL, 104
CastColorValue< float, float >
LLGL, 104
CastColorValue< std::uint8_t, std::uint8_t >
LLGL, 104
CastTo
LLGL, 104, 105
cbegin
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::UTF8String, 434
cend
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::UTF8String, 434
centered
LLGL::WindowDescriptor, 459
Clamp
LLGL, 94
clamp
LLGL::DepthBiasDescriptor, 214
Clear
LLGL::CommandBuffer, 183
LLGL::FrameProfile, 246
LLGL, 74, 85, 89
clear
LLGL::SmallVector, 390
LLGL::UTF8String, 434
ClearAttachments
LLGL::CommandBuffer, 183
ClearValue
LLGL::ClearValue, 162, 163
clearValue
LLGL::AttachmentClear, 125
LLGL::TextureDescriptor, 419
ClipDistance
LLGL, 100
clippingInvocations
LLGL::QueryPipelineStatistics, 294
clippingPrimitives
LLGL::QueryPipelineStatistics, 294
ClippingRange
LLGL, 76
clippingRange
LLGL::RenderingCapabilities, 306
CodeFile
LLGL, 95
CodeString
LLGL, 95
Color
LLGL::ClearFlags, 161
LLGL::Color, 165
LLGL::Color< T, 3u >, 169
LLGL::Color< T, 4u >, 173, 174
LLGL, 74, 100
color
LLGL::ClearValue, 163
Color.h, 464
colorArithmetic
LLGL::BlendTargetDescriptor, 144
ColorAttachment
LLGL::BindFlags, 137
colorAttachment
LLGL::AttachmentClear, 125
colorAttachments
LLGL::RenderPassDescriptor, 326
colorBits
LLGL::SwapChainDescriptor, 411
ColorDepth
LLGL::ClearFlags, 161
colorMap

LLGL::NativeContextHandle, 282
colorMask
LLGL::BlendTargetDescriptor, 144
ColorRGB.h, 465
ColorRGBA.h, 466
ColorRGBAb
LLGL, 72
ColorRGBAd
LLGL, 73
ColorRGBAf
LLGL, 73
ColorRGBAT
LLGL, 73
ColorRGBAub
LLGL, 73
ColorRGBb
LLGL, 73
ColorRGBd
LLGL, 73
ColorRGBf
LLGL, 73
ColorRGBT
LLGL, 73
ColorRGBub
LLGL, 73
CombinedSampler
LLGL::BindFlags, 137
Comma
LLGL, 88
CommandBuffer
LLGL::CommandBuffer, 180
LLGL::InterfaceID, 275
CommandBuffer.h, 467
CommandBufferDescriptor
LLGL::CommandBufferDescriptor, 205
commandBufferEncodings
LLGL::FrameProfile, 247
CommandBufferFlags.h, 467
commandBufferSubmittions
LLGL::FrameProfile, 248
CommandQueue
LLGL::CommandQueue, 208
LLGL::InterfaceID, 275
CommandQueue.h, 468
compare
LLGL::BasicStringView, 134
LLGL::UTF8String, 434, 435
compareEnabled
LLGL::SamplerDescriptor, 368
CompareOp
LLGL, 76
compareOp
LLGL::DepthDescriptor, 215
LLGL::SamplerDescriptor, 368
LLGL::StencilFaceDescriptor, 398
CompareSWO
Global functions for Strict-Weak-Order (SWO) comparisons., 21
CompatibilityProfile
LLGL, 89
components
LLGL::Color, 167
LLGL::Color< T, 3u >, 171
LLGL::Color< T, 4u >, 176
LLGL::FormatAttributes, 238
Compute
LLGL, 89, 96
compute
LLGL::ShaderDescriptor, 378
LLGL::ShaderReflection, 383
computePipelineBindings
LLGL::FrameProfile, 248
computeShader
LLGL::ComputePipelineDescriptor, 211
computeShaderInvocations
LLGL::QueryPipelineStatistics, 295
ComputeStage
LLGL::StageFlags, 396
conservativeRasterization
LLGL::RasterizerDescriptor, 296
const_iterator
LLGL::ArrayView, 121
LLGL::BasicStringView, 132
LLGL::SmallVector, 388
LLGL::UTF8String, 433
const_pointer
LLGL::ArrayView, 121
LLGL::BasicStringView, 133
LLGL::SmallVector, 388
LLGL::UTF8String, 433
const_reference
LLGL::ArrayView, 121
LLGL::BasicStringView, 133
LLGL::SmallVector, 388
LLGL::UTF8String, 433
const_reverse_iterator
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 388
LLGL::UTF8String, 433
ConstantBuffer
LLGL::BindFlags, 136
constantBufferBindings
LLGL::FrameProfile, 248
ConstantBufferDesc
Global utility functions, especially to fill descriptor structures., 42
constantBufferSize
LLGL::ShaderResource, 385
constantFactor
LLGL::DepthBiasDescriptor, 214
Constants.h, 468
ConsumeStructuredBuffer
LLGL, 99
contextProfile
LLGL::RendererConfigurationOpenGL, 299

Control
LLGL, 85
Convert
LLGL::Image, 263
ConvertImageBuffer
Image utility functions to classify and convert image data., 34, 35
Copy
LLGL, 89
CopyBuffer
LLGL::CommandBuffer, 184
CopyBufferFromTexture
LLGL::CommandBuffer, 184
CopyDst
LLGL::BindFlags, 138
CopyImageBufferRegion
Image utility functions to classify and convert image data., 35
CopyInverted
LLGL, 89
CopySrc
LLGL::BindFlags, 138
CopyTexture
LLGL::CommandBuffer, 185
CopyTextureFromBuffer
LLGL::CommandBuffer, 186
CopyTextureImageData
LLGL::RenderSystem, 331
CoreProfile
LLGL, 89
Count
LLGL::Display, 218
Counter
LLGL::MiscFlags, 280
cpuAccessFlags
LLGL::BufferDescriptor, 152
CrSel
LLGL, 88
crbegin
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 390
LLGL::UTF8String, 435
Create
LLGL::Canvas, 158
LLGL::Window, 452
CreateBuffer
LLGL::RenderSystem, 331
CreateBufferArray
LLGL::RenderSystem, 332
CreateCommandBuffer
LLGL::RenderSystem, 332
CreateCopy
LLGL::Blob, 146
CreateFence
LLGL::RenderSystem, 333
CreateFromFile
LLGL::Blob, 146, 147
CreatePipelineLayout
LLGL::RenderSystem, 333
CreatePipelineState
LLGL::RenderSystem, 333, 334
CreateQueryHeap
LLGL::RenderSystem, 335
CreateRenderPass
LLGL::RenderSystem, 335
CreateRenderTarget
LLGL::RenderSystem, 335
CreateResourceHeap
LLGL::RenderSystem, 336
CreateSampler
LLGL::RenderSystem, 336
CreateShader
LLGL::RenderSystem, 336
CreateStrongRef
LLGL::Blob, 147
CreateSwapChain
LLGL::RenderSystem, 337
CreateTexture
LLGL::RenderSystem, 337
CreateWeakRef
LLGL::Blob, 148
crend
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 391
LLGL::UTF8String, 435
CullDistance
LLGL, 100
CullMode
LLGL, 77
cullMode
LLGL::RasterizerDescriptor, 297
customMultiSampling
LLGL::RenderTargetDescriptor, 355

D
LLGL, 86
D0
LLGL, 85
D1
LLGL, 85
D16UNorm
LLGL, 83
D2
LLGL, 85
D24UNormS8UInt
LLGL, 83
D3
LLGL, 85
D32Float
LLGL, 83
D32FloatS8X24UInt
LLGL, 83
D4
LLGL, 85
D5

LLGL, 85
D6
LLGL, 86
D7
LLGL, 86
D8
LLGL, 86
D9
LLGL, 86
data
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::DstImageDescriptor, 228
LLGL::SmallVector, 391
LLGL::SrcImageDescriptor, 394
LLGL::UTF8String, 435
Data type utility functions., 32
DataTypeSize, 32
IsFloatDataType, 32
IsIntDataType, 32
IsUIntDataType, 32
dataSize
LLGL::DstImageDescriptor, 228
LLGL::SrcImageDescriptor, 394
DataType
LLGL, 78
dataType
LLGL::DstImageDescriptor, 229
LLGL::FormatAttributes, 239
LLGL::SrcImageDescriptor, 395
DataTypeSize
Data type utility functions., 32
Debug
LLGL::ShaderCompileFlags, 375
DebugCallback
Global type aliases to callback interfaces., 25
debugCallback
LLGL::RenderSystemDescriptor, 350
DecClamp
LLGL, 98
DecWrap
LLGL, 98
DecompressImageBufferToRGBA8UNorm
Image utility functions to classify and convert image data., 36
defines
LLGL::ShaderDescriptor, 378
definition
LLGL::ShaderMacro, 382
Delete
LLGL, 85
Depth
LLGL::ClearFlags, 161
LLGL, 75, 84, 100
depth
LLGL::ClearValue, 163
LLGL::Extent3D, 236
LLGL::GraphicsPipelineDescriptor, 254
depthAttachment
LLGL::RenderPassDescriptor, 326
depthBias
LLGL::RasterizerDescriptor, 297
depthBits
LLGL::SwapChainDescriptor, 411
depthClampEnabled
LLGL::RasterizerDescriptor, 297
depthFailOp
LLGL::StencilFaceDescriptor, 398
DepthGreater
LLGL, 100
DepthLess
LLGL, 100
depthPassOp
LLGL::StencilFaceDescriptor, 398
DepthStencil
LLGL::ClearFlags, 161
LLGL, 75, 84
DepthStencilAttachment
LLGL::BindFlags, 137
deviceName
LLGL::RendererInfo, 305
difference_type
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 388
LLGL::UTF8String, 433
Direct3D10
LLGL::RendererID, 303
Direct3D11
LLGL::RendererID, 303
Direct3D12
LLGL::RendererID, 303
Direct3D9
LLGL::RendererID, 303
disableClearOnResize
LLGL::WindowBehavior, 457
Disabled
LLGL, 78, 89
discardEnabled
LLGL::RasterizerDescriptor, 297
Dispatch
LLGL::CommandBuffer, 187
dispatchCommands
LLGL::FrameProfile, 248
DispatchIndirect
LLGL::CommandBuffer, 187
Display
LLGL::InterfaceID, 274
display
LLGL::NativeContextHandle, 282
LLGL::NativeHandle, 283
Display.h, 469
DisplayFlags.h, 469
displayModes
LLGL::VideoOutputDescriptor, 446
Double1

LLGL, 103
Double2
LLGL, 103
Double2x2
LLGL, 103
Double2x3
LLGL, 103
Double2x4
LLGL, 103
Double3
LLGL, 103
Double3x2
LLGL, 103
Double3x3
LLGL, 103
Double3x4
LLGL, 103
Double4
LLGL, 103
Double4x2
LLGL, 103
Double4x3
LLGL, 103
Double4x4
LLGL, 103
Down
LLGL, 85
Draw
LLGL::CommandBuffer, 187
drawCommands
LLGL::FrameProfile, 248
DrawIndexed
LLGL::CommandBuffer, 188
DrawIndexedIndirect
LLGL::CommandBuffer, 188, 189
DrawIndexedInstanced
LLGL::CommandBuffer, 189
DrawIndirect
LLGL::CommandBuffer, 190
DrawInstanced
LLGL::CommandBuffer, 191
Drop
LLGL::Input, 270
DstAlpha
LLGL, 76
dstAlpha
LLGL::BlendTargetDescriptor, 145
DstColor
LLGL, 76
dstColor
LLGL::BlendTargetDescriptor, 145
DstImageDescriptor
LLGL::DstImageDescriptor, 228
DynamicUsage
LLGL::MiscFlags, 280

E
LLGL, 86
ESProfile
LLGL, 89
ESSL_100
LLGL, 96
ESSL_300
LLGL, 97
ESSL_310
LLGL, 97
ESSL_320
LLGL, 97
ESSL
LLGL, 96
elapsedTime
LLGL::ProfileTimeRecord, 291
empty
LLGL::ArrayView, 123
LLGL::BasicStringView, 134
LLGL::SmallVector, 391
LLGL::UTF8String, 435
enabledLayers
LLGL::RendererConfigurationVulkan, 301
End
LLGL::CommandBuffer, 192
LLGL, 85
end
LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::SmallVector, 391
LLGL::UTF8String, 435
EndQuery
LLGL::CommandBuffer, 192
EndRenderCondition
LLGL::CommandBuffer, 192
EndRenderPass
LLGL::CommandBuffer, 192
EndStreamOutput
LLGL::CommandBuffer, 193
engineName
LLGL::ApplicationDescriptor, 120
engineVersion
LLGL::ApplicationDescriptor, 120
entryPoint
LLGL::ShaderDescriptor, 379
Equal
LLGL, 77
Equiv
LLGL, 89
ErEOF
LLGL, 88
erase
LLGL::SmallVector, 391
Error
LLGL::Log, 114
ErrorType
LLGL, 78
Escape
LLGL, 85
ExSel
LLGL, 88

Exe
LLGL, 85
Execute
LLGL::CommandBuffer, 193
Exponent
LLGL, 88
Export.h, 470
LLGL_EXPORT, 470
extensionNames
LLGL::RendererInfo, 305
extent
LLGL::TextureDescriptor, 419
LLGL::TextureRegion, 424
Extent2D
LLGL::Extent2D, 235
Extent3D
LLGL::Extent3D, 236

F
LLGL, 86
F1
LLGL, 87
F10
LLGL, 87
F11
LLGL, 87
F12
LLGL, 87
F13
LLGL, 87
F14
LLGL, 87
F15
LLGL, 87
F16
LLGL, 87
F17
LLGL, 87
F18
LLGL, 87
F19
LLGL, 87
F2
LLGL, 87
F20
LLGL, 87
F21
LLGL, 87
F22
LLGL, 87
F23
LLGL, 87
F24
LLGL, 87
F3
LLGL, 87
F4
LLGL, 87
F5
LLGL, 87
F6
LLGL, 87
F7
LLGL, 87
F8
LLGL, 87
F9
LLGL, 87
features
LLGL::RenderingCapabilities, 306
Fence
LLGL::InterfaceID, 275
Fence.h, 470
fenceSubmissions
LLGL::FrameProfile, 249
Fill
LLGL::Image, 263
LLGL, 90
FillBuffer
LLGL::CommandBuffer, 193
FinalizeDisplayModes
LLGL::Display, 218
FindModules
LLGL::RenderSystem, 338
FindResidentDisplay
LLGL::Canvas, 158
LLGL::Surface, 402
LLGL::Window, 452
FindUniformLocation
LLGL::Shader, 372
firstIndex
LLGL::DrawIndexedIndirectArguments, 224
firstInstance
LLGL::DrawIndexedIndirectArguments, 224
LLGL::DrawIndirectArguments, 225
LLGL::DrawPatchIndirectArguments, 227
firstPatch
LLGL::DrawPatchIndirectArguments, 227
firstVertex
LLGL::DrawIndirectArguments, 225
FixedSamples
LLGL::MiscFlags, 280
flags
LLGL::AttachmentClear, 125
LLGL::CommandBufferDescriptor, 206
LLGL::FormatAttributes, 239
LLGL::ShaderDescriptor, 379
Float1
LLGL, 103
Float16
LLGL, 78
Float2
LLGL, 103
Float2x2
LLGL, 103
Float2x3
LLGL, 103

Float2x4
LLGL, 103
Float3
LLGL, 103
Float32
LLGL, 78
Float3x2
LLGL, 103
Float3x3
LLGL, 103
Float3x4
LLGL, 103
Float4
LLGL, 103
Float4x2
LLGL, 103
Float4x3
LLGL, 103
Float4x4
LLGL, 103
Float64
LLGL, 78
for_range
ForRange.h, 473
for_range_end
ForRange.h, 473
for_range_reverse
ForRange.h, 473
for_range_reverse_iter
ForRange.h, 473
for_subrange
ForRange.h, 473
for_subrange_reverse
ForRange.h, 473
ForRange.h, 473
for_range, 473
for_range_end, 473
for_range_reverse, 473
for_range_reverse_iter, 473
for_subrange, 473
for_subrange_reverse, 473
Format
LLGL, 79
format
LLGL::AttachmentFormatDescriptor, 129
LLGL::BufferDescriptor, 152
LLGL::BufferViewDescriptor, 155
LLGL::DstImageDescriptor, 229
LLGL::FormatAttributes, 239
LLGL::FragmentAttribute, 242
LLGL::SrcImageDescriptor, 395
LLGL::TextureDescriptor, 420
LLGL::TextureViewDescriptor, 429
LLGL::VertexAttribute, 438
Format.h, 470
ForwardDecls.h, 474
FractionalEven
LLGL, 101
FractionalOdd
LLGL, 101
Fragment
LLGL, 96
fragment
LLGL::ShaderDescriptor, 379
LLGL::ShaderReflection, 383
FragmentAttribute
LLGL::FragmentAttribute, 242
FragmentAttribute.h, 474
fragmentShader
LLGL::GraphicsPipelineDescriptor, 254
fragmentShaderInvocations
LLGL::QueryPipelineStatistics, 295
FragmentStage
LLGL::StageFlags, 396
FrameProfile
LLGL::FrameProfile, 246
frameProfile
LLGL::RenderingProfiler, 324
Frequency
LLGL::Timer, 116
Front
LLGL, 78, 97
front
LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::SmallVector, 391
LLGL::StencilDescriptor, 397
LLGL::UTF8String, 435
FrontAndBack
LLGL, 97
frontCCW
LLGL::RasterizerDescriptor, 297
FrontFacing
LLGL, 100
fullscreen
LLGL::SwapChainDescriptor, 411
FullscreenMode
LLGL::ResizeBuffersFlags, 358

G
LLGL, 86
g
LLGL::Color< T, 3u >, 171
LLGL::Color< T, 4u >, 176
LLGL::TextureSwizzleRGBA, 428
GLSL_110
LLGL, 96
GLSL_120
LLGL, 96
GLSL_130
LLGL, 96
GLSL_140
LLGL, 96
GLSL_150
LLGL, 96
GLSL_330
LLGL, 96

GLSL_400
LLGL, 96
GLSL_410
LLGL, 96
GLSL_420
LLGL, 96
GLSL_430
LLGL, 96
GLSL_440
LLGL, 96
GLSL_450
LLGL, 96
GLSL_460
LLGL, 96
GLSL
LLGL, 96
GenerateImageBuffer
Image utility functions to classify and convert image data., 37
GenerateMips
LLGL::CommandBuffer, 194
LLGL::MiscFlags, 280
Geometry
LLGL, 96
geometryShader
LLGL::GraphicsPipelineDescriptor, 254
geometryShaderInvocations
LLGL::QueryPipelineStatistics, 295
geometryShaderPrimitives
LLGL::QueryPipelineStatistics, 295
GeometryStage
LLGL::StageFlags, 396
Get
LLGL::Display, 218
GetAnyKeyCount
LLGL::Input, 270
GetBehavior
LLGL::Window, 452
GetBindFlags
LLGL::Buffer, 149
LLGL::BufferArray, 151
LLGL::Texture, 415
GetBytesPerPixel
LLGL::Image, 263
GetColorFormat
LLGL::SwapChain, 406
GetCommandQueue
LLGL::RenderSystem, 338
GetContentSize
LLGL::Surface, 402
GetCursorPosition
LLGL::Display, 219
GetData
LLGL::Blob, 148
LLGL::Image, 263
GetDataSize
LLGL::Image, 263
GetDataType
LLGL::Image, 264
GetDepthStencilFormat
LLGL::SwapChain, 406
GetDepthStride
LLGL::Image, 264
GetDesc
LLGL::Buffer, 149
LLGL::Texture, 415
LLGL::Window, 453
GetDeviceName
LLGL::Display, 219
GetDisplayMode
LLGL::Display, 219
GetDstDesc
LLGL::Image, 264
GetEnteredChars
LLGL::Input, 270
GetExtent
LLGL::Image, 264
GetExtentRatio
LLGL, 105
GetFormat
LLGL::Image, 264
LLGL::Texture, 416
GetFormatAttribs
Hardware format utility functions., 28
GetGroupName
LLGL::RenderingDebugger::Message, 277
GetID
LLGL::Version, 117
GetList
LLGL::Display, 219
GetMajor
LLGL::Version, 117
GetMemoryFootprint
Hardware format utility functions., 28, 29
LLGL::Texture, 416
Texture utility functions to determine texture dimension and buffer sizes., 48
GetMinor
LLGL::Version, 117
GetMipExtent
LLGL::Texture, 416
Texture utility functions to determine texture dimension and buffer sizes., 49
GetMouseMotion
LLGL::Input, 270
GetMousePosition
LLGL::Input, 270
GetName
LLGL::RenderSystem, 338
GetNativeHandle
LLGL::Surface, 402
GetNumBindings
LLGL::PipelineLayout, 288
GetNumColorAttachments
LLGL::RenderTarget, 352
LLGL::SwapChain, 406

GetNumDescriptorSets
LLGL::ResourceHeap, 361
GetNumPixels
LLGL::Image, 264
GetOccurrences
LLGL::RenderingDebugger::Message, 277
GetOffset
LLGL::Display, 219
GetPosition
LLGL::Window, 453
GetPrimary
LLGL::Display, 220
GetPrimitiveTopologyPatchSize
LLGL, 106
GetRenderPass
LLGL::RenderTarget, 352
GetRendererID
LLGL::RenderSystem, 338
GetRendererInfo
LLGL::RenderSystem, 338
GetRenderingCaps
LLGL::RenderSystem, 338
GetReport
LLGL::PipelineState, 290
LLGL::Shader, 372
GetResolution
LLGL::RenderTarget, 352
LLGL::SwapChain, 406
GetResourceType
LLGL::Buffer, 149
LLGL::Resource, 360
LLGL::Sampler, 366
LLGL::Texture, 417
GetRevision
LLGL::Version, 118
GetRowStride
LLGL::Image, 264
GetSamples
LLGL::RenderTarget, 353
GetSize
LLGL::Blob, 148
LLGL::VertexAttribute, 438
LLGL::Window, 453
GetSource
LLGL::RenderingDebugger::Message, 277
GetSrcDesc
LLGL::Image, 265
GetStageFlags
LLGL::Shader, 373
GetStatus
LLGL::Version, 118
GetStride
LLGL::VertexFormat, 442
GetString
LLGL::Version, 118
GetSupportedDisplayModes
LLGL::Display, 220
GetSurface
LLGL::SwapChain, 406
GetText
LLGL::RenderingDebugger::Message, 277
LLGL::Report, 358
GetTitle
LLGL::Canvas, 158
LLGL::Window, 453
GetType
LLGL::QueryHeap, 292
LLGL::Shader, 373
LLGL::Texture, 417
GetWheelMotion
LLGL::Input, 270
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., 39
ToString, 39, 40
Global utility functions, especially to fill descriptor structures., 41
ConstantBufferDesc, 42
IndexBufferDesc, 42
PipelineLayoutDesc, 42
RenderPassDesc, 44
ShaderDescFromFile, 44
StorageBufferDesc, 44
Texture1DArrayDesc, 45
Texture1DDesc, 45
Texture2DArrayDesc, 45
Texture2DDesc, 45
Texture2DMSArrayDesc, 45
Texture2DMSDesc, 46
Texture3DDesc, 46
TextureCubeArrayDesc, 46
TextureCubeDesc, 46
VertexBufferDesc, 46
Graphics
LLGL, 89
graphicsPipelineBindings
LLGL::FrameProfile, 249
Greater
LLGL, 77
GreaterEqual
LLGL, 77
Green
LLGL, 101

H
LLGL, 86
HLSL_2_0
LLGL, 97

HLSL_2_0a
LLGL, 97
HLSL_2_0b
LLGL, 97
HLSL_3_0
LLGL, 97
HLSL_4_0
LLGL, 97
HLSL_4_1
LLGL, 97
HLSL_5_0
LLGL, 97
HLSL_5_1
LLGL, 97
HLSL_6_0
LLGL, 97
HLSL_6_1
LLGL, 97
HLSL_6_2
LLGL, 97
HLSL_6_3
LLGL, 97
HLSL_6_4
LLGL, 97
HLSL
LLGL, 97
Hardware format utility functions., 28
GetFormatAttribs, 28
GetMemoryFootprint, 28, 29
ImageFormatSize, 29
IsCompressedFormat, 30
IsDepthFormat, 30
IsDepthStencilFormat, 30
IsFloatFormat, 30
IsIntegralFormat, 31
IsNormalizedFormat, 31
IsStencilFormat, 31
has3DTextures
LLGL::RenderingFeatures, 312
hasArrayTextures
LLGL::RenderingFeatures, 312
hasBufferViews
LLGL::RenderingFeatures, 312
hasComputeShaders
LLGL::RenderingFeatures, 312
hasConservativeRasterization
LLGL::RenderingFeatures, 313
hasConstantBuffers
LLGL::RenderingFeatures, 313
hasCubeArrayTextures
LLGL::RenderingFeatures, 313
hasCubeTextures
LLGL::RenderingFeatures, 313
HasDepth
LLGL::FormatFlags, 240
HasDepthAttachment
LLGL::RenderTarget, 353
LLGL::SwapChain, 407
HasDepthStencil
LLGL::FormatFlags, 241
hasDirectResourceBinding
LLGL::RenderingFeatures, 313
HasErrors
LLGL::Report, 358
HasFocus
LLGL::Window, 453
hasGeometryShaders
LLGL::RenderingFeatures, 314
hasIndirectDrawing
LLGL::RenderingFeatures, 314
hasInstancing
LLGL::RenderingFeatures, 314
hasLogicOp
LLGL::RenderingFeatures, 314
hasMultiSampleTextures
LLGL::RenderingFeatures, 314
hasOffsetInstancing
LLGL::RenderingFeatures, 315
hasPipelineStatistics
LLGL::RenderingFeatures, 315
HasQuit
LLGL::Canvas, 159
LLGL::Window, 453
hasRenderCondition
LLGL::RenderingFeatures, 315
hasRenderTargets
LLGL::RenderingFeatures, 315
hasSamplers
LLGL::RenderingFeatures, 315
HasStencil
LLGL::FormatFlags, 240
HasStencilAttachment
LLGL::RenderTarget, 353
LLGL::SwapChain, 407
hasStorageBuffers
LLGL::RenderingFeatures, 315
hasStreamOutputs
LLGL::RenderingFeatures, 316
hasTessellationShaders
LLGL::RenderingFeatures, 316
hasTessellatorStage
LLGL::RenderingFeatures, 316
hasTextureViewSwizzle
LLGL::RenderingFeatures, 317
hasTextureViews
LLGL::RenderingFeatures, 316
hasUniforms
LLGL::RenderingFeatures, 317
hasViewportArrays
LLGL::RenderingFeatures, 317
height
LLGL::Extent2D, 235
LLGL::Extent3D, 236
LLGL::Scissor, 371
LLGL::Viewport, 448
Help

LLGL, 85
Home
LLGL, 85

I
LLGL, 86
IOSNativeHandle.h, 478
ignoreOffset
LLGL::Constants, 112
Image
LLGL::Image, 261, 262
LLGL, 103
Image utility functions to classify and convert image data., 33
AllocateByteBuffer, 33
ConvertImageBuffer, 34, 35
CopyImageBufferRegion, 35
DecompressImageBufferToRGBA8UNorm, 36
GenerateImageBuffer, 37
Image.h, 474
ImageFlags.h, 475
ImageFormat
LLGL, 84
ImageFormatSize
Hardware format utility functions., 29
ImmediateSubmit
LLGL::CommandBufferFlags, 207
ImproperArgument
LLGL, 104
ImproperState
LLGL, 104
IncClamp
LLGL, 98
IncOccurrence
LLGL::RenderingDebugger::Message, 278
IncWrap
LLGL, 98
IncreaseToFit
LLGL::GrowStrategyAddHalf, 257
LLGL::GrowStrategyDouble, 258
LLGL::GrowStrategyRoundUpPow2, 258
independentBlendEnabled
LLGL::BlendDescriptor, 142
IndexBuffer
LLGL::BindFlags, 136
indexBufferBindings
LLGL::FrameProfile, 249
IndexBufferDesc
Global utility functions, especially to fill descriptor structures., 42
indexFormat
LLGL::TessellationDescriptor, 413
IndirectArguments.h, 476
IndirectBuffer
LLGL::BindFlags, 137
Information
LLGL::Log, 115
initialCount
LLGL::ResourceViewDescriptor, 364
Input
LLGL::Input, 270
LLGL::InterfaceID, 274
Input.h, 476
inputAssemblyPrimitives
LLGL::QueryPipelineStatistics, 295
inputAssemblyVertices
LLGL::QueryPipelineStatistics, 295
inputAttribs
LLGL::VertexShaderAttributes, 444
Insert
LLGL, 85
insert
LLGL::SmallVector, 391
instanceDivisor
LLGL::VertexAttribute, 438
InstanceID
LLGL, 100
Int1
LLGL, 103
Int16
LLGL, 78
Int2
LLGL, 103
Int3
LLGL, 103
Int32
LLGL, 78
Int4
LLGL, 103
Int8
LLGL, 78
Integer
LLGL, 101
Interface
LLGL::InterfaceID, 274
Interface.h, 476
LLGL_DECLARE_INTERFACE, 477
LLGL_IMPLEMENT_INTERFACE, 477
InterfaceID.h, 477
InvBlendFactor
LLGL, 76
InvDstAlpha
LLGL, 76
InvDstColor
LLGL, 76
InvSrc1Alpha
LLGL, 76
InvSrc1Color
LLGL, 76
InvSrcAlpha
LLGL, 76
InvSrcColor
LLGL, 75
InvalidArgument
LLGL, 79
invalidLocation
LLGL::Constants, 112

invalidSlot
LLGL::Constants, 112
InvalidState
LLGL, 79
invalidTimerID
LLGL::Constants, 112
Invert
LLGL, 89, 98
IsArrayTexture
Texture utility functions to determine texture dimension and buffer sizes., 50
IsBlocked
LLGL::RenderingDebugger::Message, 278
IsByteAddressBuffer
Buffer utility functions to determine buffer types., 27
IsColorSpace_sRGB
LLGL::FormatFlags, 240
IsCompressed
LLGL::FormatFlags, 240
IsCompressedFormat
Hardware format utility functions., 30
IsCubeTexture
Texture utility functions to determine texture dimension and buffer sizes., 50
IsCursorShown
LLGL::Display, 220
IsDepthFormat
Hardware format utility functions., 30
IsDepthStencilFormat
Hardware format utility functions., 30
IsFloatDataType
Data type utility functions., 32
IsFloatFormat
Hardware format utility functions., 30
IsInstanceOf
LLGL::Interface, 273
LLGL, 106
IsIntDataType
Data type utility functions., 32
IsInteger
LLGL::FormatFlags, 240
IsIntegralFormat
Hardware format utility functions., 31
IsMipMappedTexture
Texture utility functions to determine texture dimension and buffer sizes., 50
IsMultiSampleTexture
Texture utility functions to determine texture dimension and buffer sizes., 50
IsNormalized
LLGL::FormatFlags, 240
IsNormalizedFormat
Hardware format utility functions., 31
IsPacked
LLGL::FormatFlags, 240
IsPostTessellationVertex
LLGL::Shader, 373
IsPrimary
LLGL::Display, 220
IsPrimitiveTopologyPatches
LLGL, 106
IsPrimitiveTopologyStrip
LLGL, 106
IsRegionInside
LLGL::Image, 265
IsShaderSourceBinary
LLGL, 107
IsShaderSourceCode
LLGL, 107
IsShown
LLGL::Window, 453
IsStencilFormat
Hardware format utility functions., 31
IsStructuredBuffer
Buffer utility functions to determine buffer types., 27
IsTextureSwizzleIdentity
Texture utility functions to determine texture dimension and buffer sizes., 50
IsTypedBuffer
Buffer utility functions to determine buffer types., 27
IsUIntDataType
Data type utility functions., 32
IsUnsigned
LLGL::FormatFlags, 240
IsUnsignedInteger
LLGL::FormatFlags, 241
iterator
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::UTF8String, 433

J
LLGL, 86

K
LLGL, 86
Keep
LLGL, 98
Key
LLGL, 84
Key.h, 478
KeyDoubleClick
LLGL::Input, 270
KeyDown
LLGL::Input, 271
KeyDownRepeated
LLGL::Input, 271
KeyPressed
LLGL::Input, 271
KeyUp
LLGL::Input, 271
Keypad0
LLGL, 86
Keypad1
LLGL, 86
Keypad2

LLGL, 86
Keypad3
LLGL, 86
Keypad4
LLGL, 86
Keypad5
LLGL, 86
Keypad6
LLGL, 86
Keypad7
LLGL, 86
Keypad8
LLGL, 86
Keypad9
LLGL, 86
KeypadDecimal
LLGL, 87
KeypadDivide
LLGL, 87
KeypadMinus
LLGL, 87
KeypadMultiply
LLGL, 86
KeypadPlus
LLGL, 87
KeypadSeparator
LLGL, 87

L
LLGL, 86
LButton
LLGL, 85
LControl
LLGL, 87
LLGL.h, 479
LLGL::ApplicationDescriptor, 119
applicationName, 119
applicationVersion, 119
engineName, 120
engineVersion, 120
LLGL::ArrayView
ArrayView, 122
at, 123
back, 123
begin, 123
cbegin, 123
cend, 123
const_iterator, 121
const_pointer, 121
const_reference, 121
const_reverse_iterator, 122
crbegin, 123
crend, 123
data, 123
difference_type, 122
empty, 123
end, 123
front, 123
iterator, 122
operator=, 123
operator[], 123
pointer, 122
rbegin, 123
reference, 122
rend, 123
reverse_iterator, 122
size, 123
size_type, 122
value_type, 122
LLGL::ArrayView< T >, 120
LLGL::AttachmentClear, 124
AttachmentClear, 124, 125
clearValue, 125
colorAttachment, 125
flags, 125
operator=, 125
LLGL::AttachmentDescriptor, 126
arrayLayer, 127
AttachmentDescriptor, 126
mipLevel, 127
texture, 127
type, 127
LLGL::AttachmentFormatDescriptor, 128
AttachmentFormatDescriptor, 129
format, 129
loadOp, 129
storeOp, 129
LLGL::BarrierFlags, 130
Storage, 131
StorageBuffer, 130
StorageTexture, 130
LLGL::BasicStringView
at, 134
back, 134
BasicStringView, 133, 134
begin, 134
cbegin, 134
cend, 134
compare, 134
const_iterator, 132
const_pointer, 133
const_reference, 133
const_reverse_iterator, 133
crbegin, 134
crend, 134
data, 134
difference_type, 133
empty, 134
end, 135
front, 135
iterator, 133
length, 135
npos, 135
operator=, 135
operator[], 135
pointer, 133
rbegin, 135

reference, 133
rend, 135
reverse_iterator, 133
size, 135
size_type, 133
substr, 135
value_type, 133
LLGL::BasicStringView< TChar, Traits >, 131
LLGL::BindFlags, 135
ColorAttachment, 137
CombinedSampler, 137
ConstantBuffer, 136
CopyDst, 138
CopySrc, 138
DepthStencilAttachment, 137
IndexBuffer, 136
IndirectBuffer, 137
Sampled, 137
Storage, 137
StreamOutputBuffer, 137
VertexBuffer, 136
LLGL::BindingDescriptor, 138
arraySize, 139
bindFlags, 139
BindingDescriptor, 139
name, 139
slot, 140
stageFlags, 140
type, 140
LLGL::BlendDescriptor, 140
alphaToCoverageEnabled, 141
blendFactor, 141
blendFactorDynamic, 142
independentBlendEnabled, 142
logicOp, 142
sampleMask, 143
targets, 143
LLGL::BlendTargetDescriptor, 144
alphaArithmetic, 144
blendEnabled, 144
colorArithmetic, 144
colorMask, 144
dstAlpha, 145
dstColor, 145
srcAlpha, 145
srcColor, 145
LLGL::Blob, 145
CreateCopy, 146
CreateFromFile, 146, 147
CreateStrongRef, 147
CreateWeakRef, 148
GetData, 148
GetSize, 148
LLGL::Buffer, 148
Buffer, 149
GetBindFlags, 149
GetDesc, 149
GetResourceType, 149
LLGL::BufferArray, 150
BufferArray, 151
GetBindFlags, 151
LLGL::BufferDescriptor, 151
bindFlags, 152
cpuAccessFlags, 152
format, 152
miscFlags, 152
size, 153
stride, 153
vertexAttribs, 153
LLGL::BufferViewDescriptor, 154
BufferViewDescriptor, 155
format, 155
offset, 155
size, 155
LLGL::CPUAccessFlags, 212
Read, 213
ReadWrite, 213
Write, 213
LLGL::Canvas, 156
~Canvas, 158
AdaptForVideoMode, 158
AddEventListener, 158
Canvas, 158
Create, 158
FindResidentDisplay, 158
GetTitle, 158
HasQuit, 159
OnProcessEvents, 159
PostQuit, 159
ProcessEvents, 159
RemoveEventListener, 159
SetTitle, 160
LLGL::Canvas::EventListener, 233
Canvas, 234
OnProcessEvents, 233
OnQuit, 234
LLGL::CanvasDescriptor, 160
borderless, 160
title, 160
LLGL::ClearFlags, 161
All, 161
Color, 161
ColorDepth, 161
Depth, 161
DepthStencil, 161
Stencil, 161
LLGL::ClearValue, 161
ClearValue, 162, 163
color, 163
depth, 163
operator=, 163
stencil, 163
LLGL::Color
Cast, 165
Color, 165
components, 167

operator∗=, 166
operator+=, 166
operator-, 166
operator-=, 166
operator/=, 166
operator[], 166, 167
Ptr, 167
LLGL::Color< T, 3u >, 167
b, 171
Cast, 169
Color, 169
components, 171
g, 171
operator∗=, 170
operator+=, 170
operator-, 170
operator-=, 170
operator/=, 170
operator[], 170, 171
Ptr, 171
r, 172
ToRGBA, 171
LLGL::Color< T, 4u >, 172
a, 176
b, 176
Cast, 174
Color, 173, 174
components, 176
g, 176
operator∗=, 175
operator+=, 175
operator-, 175
operator-=, 175
operator/=, 175
operator[], 175, 176
Ptr, 176
r, 177
ToRGB, 176
LLGL::Color< T, N >, 163
LLGL::CommandBuffer, 177
Begin, 180
BeginQuery, 180
BeginRenderCondition, 181
BeginRenderPass, 181
BeginStreamOutput, 182
Clear, 183
ClearAttachments, 183
CommandBuffer, 180
CopyBuffer, 184
CopyBufferFromTexture, 184
CopyTexture, 185
CopyTextureFromBuffer, 186
Dispatch, 187
DispatchIndirect, 187
Draw, 187
DrawIndexed, 188
DrawIndexedIndirect, 188, 189
DrawIndexedInstanced, 189
DrawIndirect, 190
DrawInstanced, 191
End, 192
EndQuery, 192
EndRenderCondition, 192
EndRenderPass, 192
EndStreamOutput, 193
Execute, 193
FillBuffer, 193
GenerateMips, 194
PopDebugGroup, 195
PushDebugGroup, 195
ResetResourceSlots, 195
SetBlendFactor, 196
SetGraphicsAPIDependentState, 197
SetIndexBuffer, 197, 198
SetPipelineState, 198
SetResource, 199
SetResourceHeap, 199
SetScissor, 200
SetScissors, 200
SetStencilReference, 201
SetUniform, 201
SetUniforms, 202
SetVertexBuffer, 202
SetVertexBufferArray, 203
SetViewport, 203
SetViewports, 203
UpdateBuffer, 204
LLGL::CommandBufferDescriptor, 204
CommandBufferDescriptor, 205
flags, 206
numNativeBuffers, 206
operator=, 206
LLGL::CommandBufferFlags, 206
ImmediateSubmit, 207
MultiSubmit, 207
Secondary, 207
LLGL::CommandQueue, 208
CommandQueue, 208
QueryResult, 208
Submit, 209, 210
WaitFence, 210
WaitIdle, 210
LLGL::ComputePipelineDescriptor, 211
computeShader, 211
pipelineLayout, 211
LLGL::ComputeShaderAttributes, 212
workGroupSize, 212
LLGL::Constants, 111
ignoreOffset, 112
invalidLocation, 112
invalidSlot, 112
invalidTimerID, 112
maxThreadCount, 113
wholeSize, 113
LLGL::DepthBiasDescriptor, 213
clamp, 214

constantFactor, 214
slopeFactor, 214
LLGL::DepthDescriptor, 214
compareOp, 215
testEnabled, 215
writeEnabled, 215
LLGL::DispatchIndirectArguments, 215
numThreadGroups, 216
LLGL::Display, 216
Count, 218
FinalizeDisplayModes, 218
Get, 218
GetCursorPosition, 219
GetDeviceName, 219
GetDisplayMode, 219
GetList, 219
GetOffset, 219
GetPrimary, 220
GetSupportedDisplayModes, 220
IsCursorShown, 220
IsPrimary, 220
ResetDisplayMode, 220
SetCursorPosition, 221
SetDisplayMode, 221
ShowCursor, 221
LLGL::DisplayModeDescriptor, 222
refreshRate, 222
resolution, 222
LLGL::DrawIndexedIndirectArguments, 223
firstIndex, 224
firstInstance, 224
numIndices, 224
numInstances, 224
vertexOffset, 224
LLGL::DrawIndirectArguments, 224
firstInstance, 225
firstVertex, 225
numInstances, 226
numVertices, 226
LLGL::DrawPatchIndirectArguments, 226
firstInstance, 227
firstPatch, 227
numInstances, 227
numPatches, 227
LLGL::DstImageDescriptor, 227
data, 228
dataSize, 228
dataType, 229
DstImageDescriptor, 228
format, 229
LLGL::Extent2D, 234
Extent2D, 235
height, 235
width, 235
LLGL::Extent3D, 235
depth, 236
Extent3D, 236
height, 236
width, 236
LLGL::Fence, 237
LLGL::FormatAttributes, 237
bitSize, 238
blockHeight, 238
blockWidth, 238
components, 238
dataType, 239
flags, 239
format, 239
LLGL::FormatFlags, 239
HasDepth, 240
HasDepthStencil, 241
HasStencil, 240
IsColorSpace_sRGB, 240
IsCompressed, 240
IsInteger, 240
IsNormalized, 240
IsPacked, 240
IsUnsigned, 240
IsUnsignedInteger, 241
SupportsGenerateMips, 240
SupportsMips, 240
SupportsRenderTarget, 240
SupportsTexture1D, 240
SupportsTexture2D, 241
SupportsTexture3D, 241
SupportsTextureCube, 241
SupportsVertex, 241
LLGL::FragmentAttribute, 241
format, 242
FragmentAttribute, 242
location, 242
name, 243
operator=, 242
systemValue, 243
LLGL::FragmentShaderAttributes, 244
outputAttribs, 244
LLGL::FrameProfile, 244
Accumulate, 246
attachmentClears, 246
bufferCopies, 246
bufferFills, 247
bufferMappings, 247
bufferReads, 247
bufferUpdates, 247
bufferWrites, 247
Clear, 246
commandBufferEncodings, 247
commandBufferSubmittions, 248
computePipelineBindings, 248
constantBufferBindings, 248
dispatchCommands, 248
drawCommands, 248
fenceSubmissions, 249
FrameProfile, 246
graphicsPipelineBindings, 249
indexBufferBindings, 249

mipMapsGenerations, 249
querySections, 249
renderConditionSections, 250
renderPassSections, 250
resourceHeapBindings, 250
sampledBufferBindings, 250
sampledTextureBindings, 250
samplerBindings, 251
storageBufferBindings, 251
storageTextureBindings, 251
streamOutputSections, 251
textureCopies, 251
textureReads, 252
textureWrites, 252
timeRecords, 252
values, 252
vertexBufferBindings, 252
LLGL::GraphicsPipelineDescriptor, 253
blend, 254
depth, 254
fragmentShader, 254
geometryShader, 254
pipelineLayout, 254
primitiveTopology, 254
rasterizer, 254
renderPass, 255
scissors, 255
stencil, 255
tessControlShader, 255
tessEvaluationShader, 256
tessellation, 255
vertexShader, 256
viewports, 256
LLGL::GrowStrategyAddHalf, 257
IncreaseToFit, 257
LLGL::GrowStrategyDouble, 257
IncreaseToFit, 258
LLGL::GrowStrategyRoundUpPow2, 258
IncreaseToFit, 258
Round, 258, 259
LLGL::Image, 259
Blit, 262
Convert, 263
Fill, 263
GetBytesPerPixel, 263
GetData, 263
GetDataSize, 263
GetDataType, 264
GetDepthStride, 264
GetDstDesc, 264
GetExtent, 264
GetFormat, 264
GetNumPixels, 264
GetRowStride, 264
GetSrcDesc, 265
Image, 261, 262
IsRegionInside, 265
MirrorXYPlane, 265
MirrorXZPlane, 265
MirrorYZPlane, 265
operator=, 265
ReadPixels, 265
Release, 266
Reset, 266
Resize, 266, 267
Swap, 267
WritePixels, 268
LLGL::Input, 268
~Input, 270
Drop, 270
GetAnyKeyCount, 270
GetEnteredChars, 270
GetMouseMotion, 270
GetMousePosition, 270
GetWheelMotion, 270
Input, 270
KeyDoubleClick, 270
KeyDown, 271
KeyDownRepeated, 271
KeyPressed, 271
KeyUp, 271
Listen, 271
LLGL::Interface, 272
IsInstanceOf, 273
LLGL::InterfaceID, 273
Buffer, 274
BufferArray, 274
Canvas, 274
Canvas_EventListener, 274
CommandBuffer, 275
CommandQueue, 275
Display, 274
Fence, 275
Input, 274
Interface, 274
PipelineLayout, 275
PipelineState, 275
QueryHeap, 275
RenderPass, 275
RenderSystem, 274
RenderSystemChild, 274
RenderTarget, 275
Report, 274
Reserved, 276
Resource, 275
ResourceHeap, 275
Sampler, 275
Shader, 275
Surface, 274
SwapChain, 275
Texture, 275
Window, 274
Window_EventListener, 274
LLGL::Log, 113
Error, 114
Information, 115

Performance, 115
PostReport, 115
ReportCallback, 114
ReportType, 114
SetReportCallback, 115
SetReportCallbackStd, 115
SetReportLimit, 116
Warning, 114
LLGL::MetalDependentStateDescriptor, 278
tessFactorBufferSlot, 279
LLGL::MiscFlags, 279
Append, 280
Counter, 280
DynamicUsage, 280
FixedSamples, 280
GenerateMips, 280
NoInitialData, 280
LLGL::NativeContextHandle, 281
colorMap, 282
display, 282
parentView, 282
parentWindow, 282
screen, 282
visual, 282
LLGL::NativeHandle, 282
display, 283
view, 283
visual, 283
window, 283
LLGL::NonCopyable, 284
~NonCopyable, 285
NonCopyable, 285
operator=, 285
LLGL::Offset2D, 285
Offset2D, 286
x, 286
y, 286
LLGL::Offset3D, 286
Offset3D, 287
x, 287
y, 287
z, 287
LLGL::PipelineLayout, 287
GetNumBindings, 288
LLGL::PipelineLayoutDescriptor, 288
bindings, 289
LLGL::PipelineState, 289
GetReport, 290
LLGL::ProfileTimeRecord, 290
annotation, 291
elapsedTime, 291
LLGL::QueryHeap, 291
GetType, 292
QueryHeap, 292
LLGL::QueryHeapDescriptor, 292
numQueries, 292
renderCondition, 292
type, 293
LLGL::QueryPipelineStatistics, 293
clippingInvocations, 294
clippingPrimitives, 294
computeShaderInvocations, 295
fragmentShaderInvocations, 295
geometryShaderInvocations, 295
geometryShaderPrimitives, 295
inputAssemblyPrimitives, 295
inputAssemblyVertices, 295
tessControlShaderInvocations, 295
tessEvaluationShaderInvocations, 295
vertexShaderInvocations, 295
LLGL::RasterizerDescriptor, 296
antiAliasedLineEnabled, 296
conservativeRasterization, 296
cullMode, 297
depthBias, 297
depthClampEnabled, 297
discardEnabled, 297
frontCCW, 297
lineWidth, 297
multiSampleEnabled, 298
polygonMode, 298
scissorTestEnabled, 298
LLGL::RenderPass, 325
LLGL::RenderPassDescriptor, 326
colorAttachments, 326
depthAttachment, 326
samples, 327
stencilAttachment, 327
LLGL::RenderSystem, 327
~RenderSystem, 331
AssertCreateBuffer, 331
AssertCreateBufferArray, 331
AssertCreateRenderTarget, 331
AssertCreateShader, 331
AssertImageDataSize, 331
CopyTextureImageData, 331
CreateBuffer, 331
CreateBufferArray, 332
CreateCommandBuffer, 332
CreateFence, 333
CreatePipelineLayout, 333
CreatePipelineState, 333, 334
CreateQueryHeap, 335
CreateRenderPass, 335
CreateRenderTarget, 335
CreateResourceHeap, 336
CreateSampler, 336
CreateShader, 336
CreateSwapChain, 337
CreateTexture, 337
FindModules, 338
GetCommandQueue, 338
GetName, 338
GetRendererID, 338
GetRendererInfo, 338
GetRenderingCaps, 338

Load, 338
MapBuffer, 339, 340
ReadBuffer, 340
ReadTexture, 341
Release, 342, 343
RenderSystem, 331
SetRendererInfo, 343
SetRenderingCaps, 343
Unload, 343
UnmapBuffer, 344
WriteBuffer, 344
WriteResourceHeap, 344
WriteTexture, 345
LLGL::RenderSystemChild, 346
SetName, 346
LLGL::RenderSystemDeleter, 347
operator(), 348
operator=, 348
RenderSystemDeleter, 348
RenderSystemDeleterFuncPtr, 348
LLGL::RenderSystemDescriptor, 349
debugCallback, 350
moduleName, 350
operator=, 350
RenderSystemDescriptor, 349
rendererConfig, 350
rendererConfigSize, 350
LLGL::RenderTarget, 351
GetNumColorAttachments, 352
GetRenderPass, 352
GetResolution, 352
GetSamples, 353
HasDepthAttachment, 353
HasStencilAttachment, 353
ValidateMipResolution, 354
ValidateResolution, 354
LLGL::RenderTargetDescriptor, 355
attachments, 355
customMultiSampling, 355
renderPass, 356
resolution, 356
samples, 356
LLGL::RendererConfigurationOpenGLES3, 300
majorVersion, 300
minorVersion, 300
LLGL::RendererConfigurationOpenGL, 298
contextProfile, 299
majorVersion, 299
minorVersion, 299
LLGL::RendererConfigurationVulkan, 300
application, 301
enabledLayers, 301
minDeviceMemoryAllocationSize, 301
reduceDeviceMemoryFragmentation, 301
LLGL::RendererID, 302
Direct3D10, 303
Direct3D11, 303
Direct3D12, 303
Direct3D9, 303
Metal, 303
Null, 303
OpenGLES1, 303
OpenGLES2, 304
OpenGLES3, 304
OpenGL, 303
Reserved, 304
Undefined, 304
Vulkan, 304
LLGL::RendererInfo, 304
deviceName, 305
extensionNames, 305
rendererName, 305
shadingLanguageName, 305
vendorName, 305
LLGL::RenderingCapabilities, 305
clippingRange, 306
features, 306
limits, 306
screenOrigin, 306
shadingLanguages, 307
textureFormats, 307
LLGL::RenderingDebugger, 307
~RenderingDebugger, 308
OnError, 309
OnWarning, 309
PostError, 309
PostWarning, 309
RenderingDebugger, 308
SetDebugGroup, 310
SetSource, 310
LLGL::RenderingDebugger::Message, 276
Block, 277
BlockAfter, 277
GetGroupName, 277
GetOccurrences, 277
GetSource, 277
GetText, 277
IncOccurrence, 278
IsBlocked, 278
Message, 277
operator=, 278
RenderingDebugger, 278
ToReportString, 278
LLGL::RenderingFeatures, 310
has3DTextures, 312
hasArrayTextures, 312
hasBufferViews, 312
hasComputeShaders, 312
hasConservativeRasterization, 313
hasConstantBuffers, 313
hasCubeArrayTextures, 313
hasCubeTextures, 313
hasDirectResourceBinding, 313
hasGeometryShaders, 314
hasIndirectDrawing, 314
hasInstancing, 314

hasLogicOp, 314
hasMultiSampleTextures, 314
hasOffsetInstancing, 315
hasPipelineStatistics, 315
hasRenderCondition, 315
hasRenderTargets, 315
hasSamplers, 315
hasStorageBuffers, 315
hasStreamOutputs, 316
hasTessellationShaders, 316
hasTessellatorStage, 316
hasTextureViewSwizzle, 317
hasTextureViews, 316
hasUniforms, 317
hasViewportArrays, 317
LLGL::RenderingLimits, 318
lineWidthRange, 319
max1DTextureSize, 319
max2DTextureSize, 319
max3DTextureSize, 319
maxAnisotropy, 319
maxBufferSize, 320
maxColorAttachments, 320
maxComputeShaderWorkGroupSize, 320
maxComputeShaderWorkGroups, 320
maxConstantBufferSize, 320
maxCubeTextureSize, 321
maxPatchVertices, 321
maxStreamOutputs, 321
maxTessFactor, 321
maxTextureArrayLayers, 322
maxViewportSize, 322
maxViewports, 322
minConstantBufferAlignment, 322
minSampledBufferAlignment, 323
minStorageBufferAlignment, 323
LLGL::RenderingProfiler, 323
Accumulate, 324
frameProfile, 324
NextProfile, 324
timeRecordingEnabled, 324
LLGL::Report, 357
GetText, 358
HasErrors, 358
LLGL::ResizeBuffersFlags, 358
AdaptSurface, 358
FullscreenMode, 358
WindowedMode, 359
LLGL::Resource, 359
GetResourceType, 360
LLGL::ResourceHeap, 360
GetNumDescriptorSets, 361
LLGL::ResourceHeapDescriptor, 361
barrierFlags, 362
numResourceViews, 362
pipelineLayout, 363
ResourceHeapDescriptor, 362
LLGL::ResourceViewDescriptor, 363
bufferView, 364
initialCount, 364
resource, 365
ResourceViewDescriptor, 364
textureView, 365
LLGL::Sampler, 366
GetResourceType, 366
Sampler, 366
LLGL::SamplerDescriptor, 367
addressModeU, 367
addressModeV, 367
addressModeW, 368
borderColor, 368
compareEnabled, 368
compareOp, 368
magFilter, 368
maxAnisotropy, 368
maxLOD, 368
minFilter, 368
minLOD, 369
mipMapFilter, 369
mipMapLODBias, 369
mipMapping, 369
LLGL::Scissor, 370
height, 371
Scissor, 370
width, 371
x, 371
y, 371
LLGL::Shader, 371
FindUniformLocation, 372
GetReport, 372
GetStageFlags, 373
GetType, 373
IsPostTessellationVertex, 373
Reflect, 373
Shader, 372
LLGL::ShaderCompileFlags, 374
Debug, 375
NoOptimization, 375
OptimizationLevel1, 375
OptimizationLevel2, 375
OptimizationLevel3, 376
PatchClippingOrigin, 376
SeparateShader, 376
WarningsAreErrors, 376
LLGL::ShaderDescriptor, 377
compute, 378
defines, 378
entryPoint, 379
flags, 379
fragment, 379
operator=, 378
profile, 379
ShaderDescriptor, 378
source, 380
sourceSize, 380
sourceType, 380

type, 381
vertex, 381
LLGL::ShaderMacro, 381
definition, 382
name, 382
operator=, 382
ShaderMacro, 382
LLGL::ShaderReflection, 383
compute, 383
fragment, 383
resources, 383
uniforms, 383
vertex, 384
LLGL::ShaderResource, 384
binding, 385
constantBufferSize, 385
storageBufferType, 385
LLGL::ShaderUniform, 385
location, 386
name, 386
size, 386
type, 386
LLGL::SmallVector
~SmallVector, 390
at, 390
back, 390
begin, 390
capacity, 390
cbegin, 390
cend, 390
clear, 390
const_iterator, 388
const_pointer, 388
const_reference, 388
const_reverse_iterator, 388
crbegin, 390
crend, 391
data, 391
difference_type, 388
empty, 391
end, 391
erase, 391
front, 391
insert, 391
iterator, 389
operator ArrayView< T >, 392
operator=, 392
operator[], 392
pointer, 389
pop_back, 392
push_back, 392
rbegin, 392
reference, 389
rend, 392
reserve, 393
resize, 393
reverse_iterator, 389
shrink_to_fit, 393
size, 393
size_type, 389
SmallVector, 389
swap, 393
value_type, 389
LLGL::SmallVector< T, LocalCapacity, Allocator, GrowStrategy >, 386
LLGL::SrcImageDescriptor, 393
data, 394
dataSize, 394
dataType, 395
format, 395
SrcImageDescriptor, 394
LLGL::StageFlags, 395
AllGraphicsStages, 396
AllStages, 396
AllTessStages, 396
ComputeStage, 396
FragmentStage, 396
GeometryStage, 396
TessControlStage, 396
TessEvaluationStage, 396
VertexStage, 396
LLGL::StencilDescriptor, 396
back, 397
front, 397
referenceDynamic, 397
testEnabled, 397
LLGL::StencilFaceDescriptor, 397
compareOp, 398
depthFailOp, 398
depthPassOp, 398
readMask, 398
reference, 398
stencilFailOp, 399
writeMask, 399
LLGL::Surface, 400
AdaptForVideoMode, 401
FindResidentDisplay, 402
GetContentSize, 402
GetNativeHandle, 402
ProcessEvents, 403
ResetPixelFormat, 403
LLGL::SwapChain, 404
~SwapChain, 405
GetColorFormat, 406
GetDepthStencilFormat, 406
GetNumColorAttachments, 406
GetResolution, 406
GetSurface, 406
HasDepthAttachment, 407
HasStencilAttachment, 407
Present, 407
ResetDisplayFullscreenMode, 407
ResizeBuffers, 407
ResizeBuffersPrimary, 408
SetDisplayFullscreenMode, 408
SetOrCreateSurface, 408

SetVsyncInterval, 409
ShareSurfaceAndConfig, 409
SwapChain, 405
SwitchFullscreen, 409
LLGL::SwapChainDescriptor, 410
colorBits, 411
depthBits, 411
fullscreen, 411
resolution, 411
samples, 411
stencilBits, 412
swapBuffers, 412
LLGL::TessellationDescriptor, 412
indexFormat, 413
maxTessFactor, 413
outputWindingCCW, 413
partition, 414
LLGL::Texture, 414
GetBindFlags, 415
GetDesc, 415
GetFormat, 416
GetMemoryFootprint, 416
GetMipExtent, 416
GetResourceType, 417
GetType, 417
Texture, 415
LLGL::TextureDescriptor, 417
arrayLayers, 418
bindFlags, 419
clearValue, 419
extent, 419
format, 420
mipLevels, 420
miscFlags, 420
samples, 420
type, 421
LLGL::TextureLocation, 421
arrayLayer, 422
mipLevel, 422
offset, 422
TextureLocation, 422
LLGL::TextureRegion, 423
extent, 424
offset, 424
subresource, 424
TextureRegion, 424
LLGL::TextureSubresource, 425
baseArrayLayer, 426
baseMipLevel, 426
numArrayLayers, 426
numMipLevels, 426
TextureSubresource, 425
LLGL::TextureSwizzleRGBA, 427
a, 428
b, 428
g, 428
r, 428
LLGL::TextureViewDescriptor, 428
format, 429
subresource, 429
swizzle, 429
type, 430
LLGL::Timer, 116
Frequency, 116
Tick, 116
LLGL::UTF8String, 431
at, 434
back, 434
begin, 434
c_str, 434
capacity, 434
cbegin, 434
cend, 434
clear, 434
compare, 434, 435
const_iterator, 433
const_pointer, 433
const_reference, 433
const_reverse_iterator, 433
crbegin, 435
crend, 435
data, 435
difference_type, 433
empty, 435
end, 435
front, 435
iterator, 433
length, 435
npos, 436
operator const_pointer, 435
operator StringView, 435
operator+=, 435
operator=, 435, 436
operator[], 436
pointer, 433
rbegin, 436
reference, 433
rend, 436
reverse_iterator, 433
size, 436
size_type, 433
ToWCharArray, 436
UTF8String, 433, 434
value_type, 433
LLGL::UninitializeTag, 430
LLGL::Version, 117
GetID, 117
GetMajor, 117
GetMinor, 117
GetRevision, 118
GetStatus, 118
GetString, 118
LLGL::VertexAttribute, 436
format, 438
GetSize, 438
instanceDivisor, 438

location, 439
name, 439
offset, 439
operator=, 438
semanticIndex, 439
slot, 440
stride, 440
systemValue, 440
VertexAttribute, 437, 438
LLGL::VertexFormat, 441
AppendAttribute, 441
attributes, 443
GetStride, 442
SetSlot, 442
SetStride, 442, 443
LLGL::VertexShaderAttributes, 443
inputAttribs, 444
outputAttribs, 444
LLGL::VideoAdapterDescriptor, 444
name, 445
outputs, 445
vendor, 445
videoMemory, 445
LLGL::VideoOutputDescriptor, 446
displayModes, 446
LLGL::Viewport, 446
height, 448
maxDepth, 448
minDepth, 448
Viewport, 447, 448
width, 448
x, 449
y, 449
LLGL::Window, 449
~Window, 451
AdaptForVideoMode, 452
AddEventListener, 452
Create, 452
FindResidentDisplay, 452
GetBehavior, 452
GetDesc, 453
GetPosition, 453
GetSize, 453
GetTitle, 453
HasFocus, 453
HasQuit, 453
IsShown, 453
OnProcessEvents, 453
PostChar, 453
PostDoubleClick, 454
PostGetFocus, 454
PostGlobalMotion, 454
PostKeyDown, 454
PostKeyUp, 454
PostLocalMotion, 454
PostLostFocus, 454
PostQuit, 455
PostResize, 455
PostTimer, 455
PostWheelMotion, 455
ProcessEvents, 455
RemoveEventListener, 456
SetBehavior, 456
SetDesc, 456
SetPosition, 456
SetSize, 456
SetTitle, 456
Show, 456
Window, 451
LLGL::Window::EventListener, 229
OnChar, 230
OnDoubleClick, 230
OnGetFocus, 230
OnGlobalMotion, 231
OnKeyDown, 231
OnKeyUp, 231
OnLocalMotion, 231
OnLostFocus, 231
OnProcessEvents, 231
OnQuit, 231
OnResize, 232
OnTimer, 232
OnWheelMotion, 232
Window, 232
LLGL::WindowBehavior, 457
disableClearOnResize, 457
moveAndResizeTimerID, 457
LLGL::WindowDescriptor, 458
acceptDropFiles, 459
borderless, 459
centered, 459
position, 459
resizable, 459
size, 459
title, 460
visible, 460
windowContext, 460
LLGL_API_CDECL
Platform.h, 485
LLGL_API_STDCALL
Platform.h, 485
LLGL_API_THISCALL
Platform.h, 485
LLGL_API
Platform.h, 485
LLGL_DECLARE_INTERFACE
Interface.h, 477
LLGL_EXPORT
Export.h, 470
LLGL_IMPLEMENT_INTERFACE
Interface.h, 477
LLGL_MAX_NUM_ATTACHMENTS
StaticLimits.h, 500
LLGL_MAX_NUM_COLOR_ATTACHMENTS
StaticLimits.h, 500
LLGL_MAX_NUM_SAMPLES

StaticLimits.h, 500
LLGL_MAX_NUM_SO_BUFFERS
StaticLimits.h, 500
LLGL_MAX_NUM_VIEWPORTS_AND_SCISSORS
StaticLimits.h, 500
LLGL, 55
A, 86
A8UNorm, 81
ABGR, 84
ANDInverted, 89
ANDReverse, 89
AND, 89
ARGB, 84
Add, 75
Alpha, 84, 101
AlwaysPass, 77
Any, 88
AnySamplesPassed, 92
AnySamplesPassedConservative, 92
AppendStructuredBuffer, 99
Apps, 86
AtomicCounter, 103
AttachmentLoadOp, 74
AttachmentStoreOp, 74
AttachmentType, 74
Attn, 88
B, 86
BC1, 84
BC1UNorm, 83
BC1UNorm_sRGB, 83
BC2, 84
BC2UNorm, 83
BC2UNorm_sRGB, 83
BC3, 84
BC3UNorm, 84
BC3UNorm_sRGB, 84
BC4, 84
BC4SNorm, 84
BC4UNorm, 84
BC5, 84
BC5SNorm, 84
BC5UNorm, 84
BGRA8SInt, 83
BGRA8SNorm, 83
BGRA8UInt, 83
BGRA8UNorm, 83
BGRA8UNorm_sRGB, 83
BGRA, 84
BGR, 84
Back, 78, 85, 97
BinaryBuffer, 95
BinaryFile, 95
BlendArithmetic, 75
BlendFactor, 76
BlendOp, 75
Blue, 101
Bool1, 103
Bool2, 103
Bool3, 103
Bool4, 103
Border, 94
BrowserBack, 87
BrowserFavorits, 87
BrowserForward, 87
BrowserHome, 87
BrowserRefresh, 87
BrowserSearch, 87
BrowserStop, 87
Buffer, 93
ByRegionNoWait, 92
ByRegionNoWaitInverted, 92
ByRegionWait, 92
ByRegionWaitInverted, 92
ByteAddressBuffer, 98
ByteBuffer, 72
C, 86
CPUAccess, 77
Cancel, 85
Capital, 85
CastColorValue, 104
CastColorValue< bool, bool >, 104
CastColorValue< double, double >, 104
CastColorValue< float, float >, 104
CastColorValue< std::uint8_t, std::uint8_t >, 104
CastTo, 104, 105
Clamp, 94
Clear, 74, 85, 89
ClipDistance, 100
ClippingRange, 76
CodeFile, 95
CodeString, 95
Color, 74, 100
ColorRGBAb, 72
ColorRGBAd, 73
ColorRGBAf, 73
ColorRGBAT, 73
ColorRGBAub, 73
ColorRGBb, 73
ColorRGBd, 73
ColorRGBf, 73
ColorRGBT, 73
ColorRGBub, 73
Comma, 88
CompareOp, 76
CompatibilityProfile, 89
Compute, 89, 96
ConsumeStructuredBuffer, 99
Control, 85
Copy, 89
CopyInverted, 89
CoreProfile, 89
CrSel, 88
CullDistance, 100
CullMode, 77
D, 86
D0, 85

D1, 85
D16UNorm, 83
D2, 85
D24UNormS8UInt, 83
D3, 85
D32Float, 83
D32FloatS8X24UInt, 83
D4, 85
D5, 85
D6, 86
D7, 86
D8, 86
D9, 86
DataType, 78
DecClamp, 98
DecWrap, 98
Delete, 85
Depth, 75, 84, 100
DepthGreater, 100
DepthLess, 100
DepthStencil, 75, 84
Disabled, 78, 89
Double1, 103
Double2, 103
Double2x2, 103
Double2x3, 103
Double2x4, 103
Double3, 103
Double3x2, 103
Double3x3, 103
Double3x4, 103
Double4, 103
Double4x2, 103
Double4x3, 103
Double4x4, 103
Down, 85
DstAlpha, 76
DstColor, 76
E, 86
ESProfile, 89
ESSL_100, 96
ESSL_300, 97
ESSL_310, 97
ESSL_320, 97
ESSL, 96
End, 85
Equal, 77
Equiv, 89
ErEOF, 88
ErrorType, 78
Escape, 85
ExSel, 88
Exe, 85
Exponent, 88
F, 86
F1, 87
F10, 87
F11, 87
F12, 87
F13, 87
F14, 87
F15, 87
F16, 87
F17, 87
F18, 87
F19, 87
F2, 87
F20, 87
F21, 87
F22, 87
F23, 87
F24, 87
F3, 87
F4, 87
F5, 87
F6, 87
F7, 87
F8, 87
F9, 87
Fill, 90
Float1, 103
Float16, 78
Float2, 103
Float2x2, 103
Float2x3, 103
Float2x4, 103
Float3, 103
Float32, 78
Float3x2, 103
Float3x3, 103
Float3x4, 103
Float4, 103
Float4x2, 103
Float4x3, 103
Float4x4, 103
Float64, 78
Format, 79
FractionalEven, 101
FractionalOdd, 101
Fragment, 96
Front, 78, 97
FrontAndBack, 97
FrontFacing, 100
G, 86
GLSL_110, 96
GLSL_120, 96
GLSL_130, 96
GLSL_140, 96
GLSL_150, 96
GLSL_330, 96
GLSL_400, 96
GLSL_410, 96
GLSL_420, 96
GLSL_430, 96
GLSL_440, 96
GLSL_450, 96

GLSL_460, 96
GLSL, 96
Geometry, 96
GetExtentRatio, 105
GetPrimitiveTopologyPatchSize, 106
Graphics, 89
Greater, 77
GreaterEqual, 77
Green, 101
H, 86
HLSL_2_0, 97
HLSL_2_0a, 97
HLSL_2_0b, 97
HLSL_3_0, 97
HLSL_4_0, 97
HLSL_4_1, 97
HLSL_5_0, 97
HLSL_5_1, 97
HLSL_6_0, 97
HLSL_6_1, 97
HLSL_6_2, 97
HLSL_6_3, 97
HLSL_6_4, 97
HLSL, 97
Help, 85
Home, 85
I, 86
Image, 103
ImageFormat, 84
ImproperArgument, 104
ImproperState, 104
IncClamp, 98
IncWrap, 98
Insert, 85
InstanceID, 100
Int1, 103
Int16, 78
Int2, 103
Int3, 103
Int32, 78
Int4, 103
Int8, 78
Integer, 101
InvBlendFactor, 76
InvDstAlpha, 76
InvDstColor, 76
InvSrc1Alpha, 76
InvSrc1Color, 76
InvSrcAlpha, 76
InvSrcColor, 75
InvalidArgument, 79
InvalidState, 79
Invert, 89, 98
IsInstanceOf, 106
IsPrimitiveTopologyPatches, 106
IsPrimitiveTopologyStrip, 106
IsShaderSourceBinary, 107
IsShaderSourceCode, 107
J, 86
K, 86
Keep, 98
Key, 84
Keypad0, 86
Keypad1, 86
Keypad2, 86
Keypad3, 86
Keypad4, 86
Keypad5, 86
Keypad6, 86
Keypad7, 86
Keypad8, 86
Keypad9, 86
KeypadDecimal, 87
KeypadDivide, 87
KeypadMinus, 87
KeypadMultiply, 86
KeypadPlus, 87
KeypadSeparator, 87
L, 86
LButton, 85
LControl, 87
LMenu, 87
LShift, 87
LWin, 86
LaunchApp1, 88
LaunchApp2, 88
LaunchMail, 88
LaunchMediaSelect, 88
Left, 85
Less, 77
LessEqual, 77
LineList, 90
LineListAdjacency, 90
LineStrip, 90
LineStripAdjacency, 90
Linear, 95
Load, 74
LogicOp, 88
LowerLeft, 95
M, 86
MButton, 85
Max, 75
MaxColorValue, 107
MaxColorValue< bool >, 107
MaxColorValue< std::uint8_t >, 107
MediaNextTrack, 88
MediaPlayPause, 88
MediaPrevTrack, 88
MediaStop, 88
Menu, 85
Metal, 97
Metal_1_0, 97
Metal_1_1, 97
Metal_1_2, 97
Metal_2_0, 97
Metal_2_1, 97

Min, 75
Minus, 88
MinusOneToOne, 76
Mirror, 93
MirrorOnce, 94
N, 86
NAND, 89
NOR, 89
Nearest, 95
NeverPass, 77
NoName, 88
NoOp, 89
NoWait, 92
NoWaitInverted, 92
NotEqual, 77
NumLock, 87
O, 86
OEMClear, 88
ORInverted, 89
ORReverse, 89
One, 75, 101
OpenGLContextProfile, 89
operator!=, 107, 108
operator∗, 108, 109
operator+, 109
operator-, 109
operator/, 109
operator==, 109, 110
OR, 89
P, 86
PA1, 88
PageDown, 85
PageUp, 85
Patches1, 91
Patches10, 91
Patches11, 91
Patches12, 91
Patches13, 91
Patches14, 91
Patches15, 91
Patches16, 91
Patches17, 91
Patches18, 91
Patches19, 91
Patches2, 91
Patches20, 91
Patches21, 91
Patches22, 91
Patches23, 91
Patches24, 91
Patches25, 91
Patches26, 91
Patches27, 91
Patches28, 91
Patches29, 91
Patches3, 91
Patches30, 91
Patches31, 91
Patches32, 91
Patches4, 91
Patches5, 91
Patches6, 91
Patches7, 91
Patches8, 91
Patches9, 91
Pause, 85
Period, 88
PipelineBindPoint, 89
PipelineStatistics, 92
Play, 88
Plus, 88
PointList, 90
PointlessOperation, 104
Points, 90
PolygonMode, 89
Position, 100
Pow2, 101
PrimitiveID, 100
PrimitiveTopology, 90
Print, 85
Q, 86
QueryType, 91
R, 84, 86
R16Float, 81
R16SInt, 81
R16SNorm, 81
R16UInt, 81
R16UNorm, 81
R32Float, 81
R32SInt, 81
R32UInt, 81
R64Float, 81
R8SInt, 81
R8SNorm, 81
R8UInt, 81
R8UNorm, 81
RButton, 85
RControl, 87
RG11B10Float, 83
RG16Float, 81
RG16SInt, 81
RG16SNorm, 81
RG16UInt, 81
RG16UNorm, 81
RG32Float, 81
RG32SInt, 81
RG32UInt, 81
RG64Float, 81
RG8SInt, 81
RG8SNorm, 81
RG8UInt, 81
RG8UNorm, 81
RGB10A2UInt, 83
RGB10A2UNorm, 83
RGB16Float, 82
RGB16SInt, 82

RGB16SNorm, 82
RGB16UInt, 82
RGB16UNorm, 82
RGB32Float, 82
RGB32SInt, 82
RGB32UInt, 82
RGB64Float, 82
RGB8SInt, 82
RGB8SNorm, 82
RGB8UInt, 82
RGB8UNorm, 81
RGB8UNorm_sRGB, 81
RGB9E5Float, 83
RGBA16Float, 83
RGBA16SInt, 83
RGBA16SNorm, 83
RGBA16UInt, 83
RGBA16UNorm, 83
RGBA32Float, 83
RGBA32SInt, 83
RGBA32UInt, 83
RGBA64Float, 83
RGBA8SInt, 83
RGBA8SNorm, 82
RGBA8UInt, 82
RGBA8UNorm, 82
RGBA8UNorm_sRGB, 82
RGBA, 84
RGB, 84
RMenu, 87
RShift, 87
RWByteAddressBuffer, 99
RWStructuredBuffer, 99
RWTypedBuffer, 98
RWin, 86
ReadOnly, 77
ReadWrite, 77
Red, 101
RenderConditionMode, 92
RenderSystemPtr, 73
RenderTargetIndex, 100
Repeat, 93
Replace, 98
ResourceType, 92
Return, 85
RevSubtract, 75
RG, 84
Right, 85
S, 86
SPIRV_100, 97
SPIRV, 97
SampleID, 100
SampleMask, 100
Sampler, 93, 103
SamplerAddressMode, 93
SamplerFilter, 94
SamplesPassed, 92
ScreenOrigin, 95
ScrollLock, 87
Select, 85
Set, 89
ShaderSourceType, 95
ShaderType, 95
ShadingLanguage, 96
Shift, 85
Sleep, 86
Snapshot, 85
Space, 85
Src1Alpha, 76
Src1Color, 76
SrcAlpha, 75
SrcAlphaSaturate, 76
SrcColor, 75
Stencil, 75, 100
StencilFace, 97
StencilOp, 97
StorageBufferType, 98
Store, 74
StreamOutOverflow, 92
StreamOutPrimitivesWritten, 92
StringView, 73
StructuredBuffer, 98
Subtract, 75
SystemValue, 99
T, 86
Tab, 85
TessControl, 96
TessEvaluation, 96
TessellationPartition, 101
Texture, 93
Texture1DArray, 102
Texture1D, 102
Texture2DArray, 102
Texture2DMSArray, 102
Texture2DMS, 102
Texture2D, 102
Texture3D, 102
TextureCube, 102
TextureCubeArray, 102
TextureSwizzle, 101
TextureType, 101
TimeElapsed, 92
TriangleList, 91
TriangleListAdjacency, 91
TriangleStrip, 91
TriangleStripAdjacency, 91
TypedBuffer, 98
U, 86
UInt1, 103
UInt16, 78
UInt2, 103
UInt3, 103
UInt32, 78
UInt4, 103
UInt8, 78
Undefined, 74, 78, 81, 89, 93, 96, 98, 100, 101, 103

UndefinedBehavior, 79
UniformLocation, 73
UniformType, 102
UnsupportedFeature, 79
Up, 85
UpperLeft, 95
V, 86
ValidateRenderingCaps, 110
VaryingBehavior, 104
VersionBitmask, 97
Vertex, 96
VertexID, 100
ViewportIndex, 101
VolumeDown, 88
VolumeMute, 87
VolumeUp, 88
W, 86
WStringView, 73
Wait, 92
WaitInverted, 92
WarningType, 103
Wireframe, 90
WriteDiscard, 77
WriteOnly, 77
X, 86
XButton1, 85
XButton2, 85
XOR, 89
Y, 86
Z, 86
Zero, 75, 98, 101
ZeroToOne, 76
Zoom, 88
LMenu
LLGL, 87
LShift
LLGL, 87
LWin
LLGL, 86
LaunchApp1
LLGL, 88
LaunchApp2
LLGL, 88
LaunchMail
LLGL, 88
LaunchMediaSelect
LLGL, 88
Left
LLGL, 85
length
LLGL::BasicStringView, 135
LLGL::UTF8String, 435
Less
LLGL, 77
LessEqual
LLGL, 77
limits
LLGL::RenderingCapabilities, 306
LineList
LLGL, 90
LineListAdjacency
LLGL, 90
LineStrip
LLGL, 90
LineStripAdjacency
LLGL, 90
lineWidth
LLGL::RasterizerDescriptor, 297
lineWidthRange
LLGL::RenderingLimits, 319
Linear
LLGL, 95
LinuxNativeHandle.h, 479
Listen
LLGL::Input, 271
Load
LLGL::RenderSystem, 338
LLGL, 74
loadOp
LLGL::AttachmentFormatDescriptor, 129
location
LLGL::FragmentAttribute, 242
LLGL::ShaderUniform, 386
LLGL::VertexAttribute, 439
Log.h, 480
LogicOp
LLGL, 88
logicOp
LLGL::BlendDescriptor, 142
LowerLeft
LLGL, 95

M
LLGL, 86
MButton
LLGL, 85
MacOSNativeHandle.h, 481
magFilter
LLGL::SamplerDescriptor, 368
majorVersion
LLGL::RendererConfigurationOpenGLES3, 300
LLGL::RendererConfigurationOpenGL, 299
MapBuffer
LLGL::RenderSystem, 339, 340
Max
LLGL, 75
max1DTextureSize
LLGL::RenderingLimits, 319
max2DTextureSize
LLGL::RenderingLimits, 319
max3DTextureSize
LLGL::RenderingLimits, 319
maxAnisotropy
LLGL::RenderingLimits, 319
LLGL::SamplerDescriptor, 368
maxBufferSize
LLGL::RenderingLimits, 320

maxColorAttachments
LLGL::RenderingLimits, 320
MaxColorValue
LLGL, 107
MaxColorValue< bool >
LLGL, 107
MaxColorValue< std::uint8_t >
LLGL, 107
maxComputeShaderWorkGroupSize
LLGL::RenderingLimits, 320
maxComputeShaderWorkGroups
LLGL::RenderingLimits, 320
maxConstantBufferSize
LLGL::RenderingLimits, 320
maxCubeTextureSize
LLGL::RenderingLimits, 321
maxDepth
LLGL::Viewport, 448
maxLOD
LLGL::SamplerDescriptor, 368
maxPatchVertices
LLGL::RenderingLimits, 321
maxStreamOutputs
LLGL::RenderingLimits, 321
maxTessFactor
LLGL::RenderingLimits, 321
LLGL::TessellationDescriptor, 413
maxTextureArrayLayers
LLGL::RenderingLimits, 322
maxThreadCount
LLGL::Constants, 113
maxViewportSize
LLGL::RenderingLimits, 322
maxViewports
LLGL::RenderingLimits, 322
MediaNextTrack
LLGL, 88
MediaPlayPause
LLGL, 88
MediaPrevTrack
LLGL, 88
MediaStop
LLGL, 88
Menu
LLGL, 85
Message
LLGL::RenderingDebugger::Message, 277
Metal
LLGL::RendererID, 303
LLGL, 97
Metal_1_0
LLGL, 97
Metal_1_1
LLGL, 97
Metal_1_2
LLGL, 97
Metal_2_0
LLGL, 97
Metal_2_1
LLGL, 97
Min
LLGL, 75
minConstantBufferAlignment
LLGL::RenderingLimits, 322
minDepth
LLGL::Viewport, 448
minDeviceMemoryAllocationSize
LLGL::RendererConfigurationVulkan, 301
minFilter
LLGL::SamplerDescriptor, 368
minLOD
LLGL::SamplerDescriptor, 369
minSampledBufferAlignment
LLGL::RenderingLimits, 323
minStorageBufferAlignment
LLGL::RenderingLimits, 323
minorVersion
LLGL::RendererConfigurationOpenGLES3, 300
LLGL::RendererConfigurationOpenGL, 299
Minus
LLGL, 88
MinusOneToOne
LLGL, 76
mipLevel
LLGL::AttachmentDescriptor, 127
LLGL::TextureLocation, 422
mipLevels
LLGL::TextureDescriptor, 420
mipMapFilter
LLGL::SamplerDescriptor, 369
mipMapLODBias
LLGL::SamplerDescriptor, 369
mipMapping
LLGL::SamplerDescriptor, 369
mipMapsGenerations
LLGL::FrameProfile, 249
Mirror
LLGL, 93
MirrorOnce
LLGL, 94
MirrorXYPlane
LLGL::Image, 265
MirrorXZPlane
LLGL::Image, 265
MirrorYZPlane
LLGL::Image, 265
miscFlags
LLGL::BufferDescriptor, 152
LLGL::TextureDescriptor, 420
moduleName
LLGL::RenderSystemDescriptor, 350
moveAndResizeTimerID
LLGL::WindowBehavior, 457
multiSampleEnabled
LLGL::RasterizerDescriptor, 298
MultiSubmit

LLGL::CommandBufferFlags, 207

N
LLGL, 86
NAND
LLGL, 89
NOR
LLGL, 89
name
LLGL::BindingDescriptor, 139
LLGL::FragmentAttribute, 243
LLGL::ShaderMacro, 382
LLGL::ShaderUniform, 386
LLGL::VertexAttribute, 439
LLGL::VideoAdapterDescriptor, 445
NativeHandle.h, 481
Nearest
LLGL, 95
NeverPass
LLGL, 77
NextProfile
LLGL::RenderingProfiler, 324
NoInitialData
LLGL::MiscFlags, 280
NoName
LLGL, 88
NoOp
LLGL, 89
NoOptimization
LLGL::ShaderCompileFlags, 375
NoWait
LLGL, 92
NoWaitInverted
LLGL, 92
NonCopyable
LLGL::NonCopyable, 285
NonCopyable.h, 481
NotEqual
LLGL, 77
npos
LLGL::BasicStringView, 135
LLGL::UTF8String, 436
Null
LLGL::RendererID, 303
numArrayLayers
LLGL::TextureSubresource, 426
numIndices
LLGL::DrawIndexedIndirectArguments, 224
numInstances
LLGL::DrawIndexedIndirectArguments, 224
LLGL::DrawIndirectArguments, 226
LLGL::DrawPatchIndirectArguments, 227
NumLock
LLGL, 87
NumMipDimensions
Texture utility functions to determine texture dimension and buffer sizes., 51
NumMipLevels
Texture utility functions to determine texture dimension and buffer sizes., 51, 52
numMipLevels
LLGL::TextureSubresource, 426
NumMipTexels
Texture utility functions to determine texture dimension and buffer sizes., 52, 53
numNativeBuffers
LLGL::CommandBufferDescriptor, 206
numPatches
LLGL::DrawPatchIndirectArguments, 227
numQueries
LLGL::QueryHeapDescriptor, 292
numResourceViews
LLGL::ResourceHeapDescriptor, 362
NumTextureDimensions
Texture utility functions to determine texture dimension and buffer sizes., 53
numThreadGroups
LLGL::DispatchIndirectArguments, 216
numVertices
LLGL::DrawIndirectArguments, 226

O
LLGL, 86
OEMClear
LLGL, 88
ORInverted
LLGL, 89
ORReverse
LLGL, 89
offset
LLGL::BufferViewDescriptor, 155
LLGL::TextureLocation, 422
LLGL::TextureRegion, 424
LLGL::VertexAttribute, 439
Offset2D
LLGL::Offset2D, 286
Offset3D
LLGL::Offset3D, 287
OnChar
LLGL::Window::EventListener, 230
OnDoubleClick
LLGL::Window::EventListener, 230
OnError
LLGL::RenderingDebugger, 309
OnGetFocus
LLGL::Window::EventListener, 230
OnGlobalMotion
LLGL::Window::EventListener, 231
OnKeyDown
LLGL::Window::EventListener, 231
OnKeyUp
LLGL::Window::EventListener, 231
OnLocalMotion
LLGL::Window::EventListener, 231
OnLostFocus
LLGL::Window::EventListener, 231
OnProcessEvents

LLGL::Canvas, 159
LLGL::Canvas::EventListener, 233
LLGL::Window, 453
LLGL::Window::EventListener, 231
OnQuit
LLGL::Canvas::EventListener, 234
LLGL::Window::EventListener, 231
OnResize
LLGL::Window::EventListener, 232
OnTimer
LLGL::Window::EventListener, 232
OnWarning
LLGL::RenderingDebugger, 309
OnWheelMotion
LLGL::Window::EventListener, 232
One
LLGL, 75, 101
OpenGLContextProfile
LLGL, 89
OpenGLES1
LLGL::RendererID, 303
OpenGLES2
LLGL::RendererID, 304
OpenGLES3
LLGL::RendererID, 304
OpenGL
LLGL::RendererID, 303
operator ArrayView< T >
LLGL::SmallVector, 392
operator const_pointer
LLGL::UTF8String, 435
operator StringView
LLGL::UTF8String, 435
operator!=
Global operators for basic data structures., 22, 23
LLGL, 107, 108
operator*
LLGL, 108, 109
operator*=
LLGL::Color, 166
LLGL::Color< T, 3u >, 170
LLGL::Color< T, 4u >, 175
operator()
LLGL::RenderSystemDeleter, 348
operator+
Global operators for basic data structures., 23
LLGL, 109
operator+=
LLGL::Color, 166
LLGL::Color< T, 3u >, 170
LLGL::Color< T, 4u >, 175
LLGL::UTF8String, 435
operator-
Global operators for basic data structures., 23, 24
LLGL::Color, 166
LLGL::Color< T, 3u >, 170
LLGL::Color< T, 4u >, 175
LLGL, 109
operator-=
LLGL::Color, 166
LLGL::Color< T, 3u >, 170
LLGL::Color< T, 4u >, 175
operator/
LLGL, 109
operator/=
LLGL::Color, 166
LLGL::Color< T, 3u >, 170
LLGL::Color< T, 4u >, 175
operator=
LLGL::ArrayView, 123
LLGL::AttachmentClear, 125
LLGL::BasicStringView, 135
LLGL::ClearValue, 163
LLGL::CommandBufferDescriptor, 206
LLGL::FragmentAttribute, 242
LLGL::Image, 265
LLGL::NonCopyable, 285
LLGL::RenderSystemDeleter, 348
LLGL::RenderSystemDescriptor, 350
LLGL::RenderingDebugger::Message, 278
LLGL::ShaderDescriptor, 378
LLGL::ShaderMacro, 382
LLGL::SmallVector, 392
LLGL::UTF8String, 435, 436
LLGL::VertexAttribute, 438
operator==
Global operators for basic data structures., 24
LLGL, 109, 110
operator[]
LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::Color, 166, 167
LLGL::Color< T, 3u >, 170, 171
LLGL::Color< T, 4u >, 175, 176
LLGL::SmallVector, 392
LLGL::UTF8String, 436
OptimizationLevel1
LLGL::ShaderCompileFlags, 375
OptimizationLevel2
LLGL::ShaderCompileFlags, 375
OptimizationLevel3
LLGL::ShaderCompileFlags, 376
OR
LLGL, 89
outputAttribs
LLGL::FragmentShaderAttributes, 244
LLGL::VertexShaderAttributes, 444
outputWindingCCW
LLGL::TessellationDescriptor, 413
outputs
LLGL::VideoAdapterDescriptor, 445

P
LLGL, 86
PA1
LLGL, 88
PageDown

LLGL, 85
PageUp
LLGL, 85
parentView
LLGL::NativeContextHandle, 282
parentWindow
LLGL::NativeContextHandle, 282
partition
LLGL::TessellationDescriptor, 414
PatchClippingOrigin
LLGL::ShaderCompileFlags, 376
Patches1
LLGL, 91
Patches10
LLGL, 91
Patches11
LLGL, 91
Patches12
LLGL, 91
Patches13
LLGL, 91
Patches14
LLGL, 91
Patches15
LLGL, 91
Patches16
LLGL, 91
Patches17
LLGL, 91
Patches18
LLGL, 91
Patches19
LLGL, 91
Patches2
LLGL, 91
Patches20
LLGL, 91
Patches21
LLGL, 91
Patches22
LLGL, 91
Patches23
LLGL, 91
Patches24
LLGL, 91
Patches25
LLGL, 91
Patches26
LLGL, 91
Patches27
LLGL, 91
Patches28
LLGL, 91
Patches29
LLGL, 91
Patches3
LLGL, 91
Patches30
LLGL, 91
Patches31
LLGL, 91
Patches32
LLGL, 91
Patches4
LLGL, 91
Patches5
LLGL, 91
Patches6
LLGL, 91
Patches7
LLGL, 91
Patches8
LLGL, 91
Patches9
LLGL, 91
Pause
LLGL, 85
Performance
LLGL::Log, 115
Period
LLGL, 88
PipelineBindPoint
LLGL, 89
PipelineLayout
LLGL::InterfaceID, 275
pipelineLayout
LLGL::ComputePipelineDescriptor, 211
LLGL::GraphicsPipelineDescriptor, 254
LLGL::ResourceHeapDescriptor, 363
PipelineLayout.h, 481
PipelineLayoutDesc
Global utility functions, especially to fill descriptor structures., 42
PipelineLayoutFlags.h, 482
PipelineState
LLGL::InterfaceID, 275
PipelineState.h, 482
PipelineStateFlags.h, 483
PipelineStatistics
LLGL, 92
Platform.h, 485
LLGL_API_CDECL, 485
LLGL_API_STDCALL, 485
LLGL_API_THISCALL, 485
LLGL_API, 485
Play
LLGL, 88
Plus
LLGL, 88
PointList
LLGL, 90
pointer
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::UTF8String, 433

PointlessOperation
LLGL, 104
Points
LLGL, 90
PolygonMode
LLGL, 89
polygonMode
LLGL::RasterizerDescriptor, 298
pop_back
LLGL::SmallVector, 392
PopDebugGroup
LLGL::CommandBuffer, 195
Position
LLGL, 100
position
LLGL::WindowDescriptor, 459
PostChar
LLGL::Window, 453
PostDoubleClick
LLGL::Window, 454
PostError
LLGL::RenderingDebugger, 309
PostGetFocus
LLGL::Window, 454
PostGlobalMotion
LLGL::Window, 454
PostKeyDown
LLGL::Window, 454
PostKeyUp
LLGL::Window, 454
PostLocalMotion
LLGL::Window, 454
PostLostFocus
LLGL::Window, 454
PostQuit
LLGL::Canvas, 159
LLGL::Window, 455
PostReport
LLGL::Log, 115
PostResize
LLGL::Window, 455
PostTimer
LLGL::Window, 455
PostWarning
LLGL::RenderingDebugger, 309
PostWheelMotion
LLGL::Window, 455
Pow2
LLGL, 101
Present
LLGL::SwapChain, 407
PrimitiveID
LLGL, 100
PrimitiveTopology
LLGL, 90
primitiveTopology
LLGL::GraphicsPipelineDescriptor, 254
Print
LLGL, 85
ProcessEvents
LLGL::Canvas, 159
LLGL::Surface, 403
LLGL::Window, 455
profile
LLGL::ShaderDescriptor, 379
Ptr
LLGL::Color, 167
LLGL::Color< T, 3u >, 171
LLGL::Color< T, 4u >, 176
push_back
LLGL::SmallVector, 392
PushDebugGroup
LLGL::CommandBuffer, 195

Q
LLGL, 86
QueryHeap
LLGL::InterfaceID, 275
LLGL::QueryHeap, 292
QueryHeap.h, 485
QueryHeapFlags.h, 486
QueryResult
LLGL::CommandQueue, 208
querySections
LLGL::FrameProfile, 249
QueryType
LLGL, 91

R
LLGL, 84, 86
r
LLGL::Color< T, 3u >, 172
LLGL::Color< T, 4u >, 177
LLGL::TextureSwizzleRGBA, 428
R16Float
LLGL, 81
R16SInt
LLGL, 81
R16SNorm
LLGL, 81
R16UInt
LLGL, 81
R16UNorm
LLGL, 81
R32Float
LLGL, 81
R32SInt
LLGL, 81
R32UInt
LLGL, 81
R64Float
LLGL, 81
R8SInt
LLGL, 81
R8SNorm
LLGL, 81
R8UInt

LLGL, 81
R8UNorm
LLGL, 81
RButton
LLGL, 85
RControl
LLGL, 87
RG11B10Float
LLGL, 83
RG16Float
LLGL, 81
RG16SInt
LLGL, 81
RG16SNorm
LLGL, 81
RG16UInt
LLGL, 81
RG16UNorm
LLGL, 81
RG32Float
LLGL, 81
RG32SInt
LLGL, 81
RG32UInt
LLGL, 81
RG64Float
LLGL, 81
RG8SInt
LLGL, 81
RG8SNorm
LLGL, 81
RG8UInt
LLGL, 81
RG8UNorm
LLGL, 81
RGB10A2UInt
LLGL, 83
RGB10A2UNorm
LLGL, 83
RGB16Float
LLGL, 82
RGB16SInt
LLGL, 82
RGB16SNorm
LLGL, 82
RGB16UInt
LLGL, 82
RGB16UNorm
LLGL, 82
RGB32Float
LLGL, 82
RGB32SInt
LLGL, 82
RGB32UInt
LLGL, 82
RGB64Float
LLGL, 82
RGB8SInt
LLGL, 82
RGB8SNorm
LLGL, 82
RGB8UInt
LLGL, 82
RGB8UNorm
LLGL, 81
RGB8UNorm_sRGB
LLGL, 81
RGB9E5Float
LLGL, 83
RGBA16Float
LLGL, 83
RGBA16SInt
LLGL, 83
RGBA16SNorm
LLGL, 83
RGBA16UInt
LLGL, 83
RGBA16UNorm
LLGL, 83
RGBA32Float
LLGL, 83
RGBA32SInt
LLGL, 83
RGBA32UInt
LLGL, 83
RGBA64Float
LLGL, 83
RGBA8SInt
LLGL, 83
RGBA8SNorm
LLGL, 82
RGBA8UInt
LLGL, 82
RGBA8UNorm
LLGL, 82
RGBA8UNorm_sRGB
LLGL, 82
RGBA
LLGL, 84
RGB
LLGL, 84
RMenu
LLGL, 87
RShift
LLGL, 87
RWByteAddressBuffer
LLGL, 99
RWStructuredBuffer
LLGL, 99
RWTypedBuffer
LLGL, 98
RWin
LLGL, 86
rasterizer
LLGL::GraphicsPipelineDescriptor, 254
rbegin

LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::SmallVector, 392
LLGL::UTF8String, 436
Read
LLGL::CPUAccessFlags, 213
ReadBuffer
LLGL::RenderSystem, 340
readMask
LLGL::StencilFaceDescriptor, 398
ReadOnly
LLGL, 77
ReadPixels
LLGL::Image, 265
ReadTexture
LLGL::RenderSystem, 341
ReadWrite
LLGL::CPUAccessFlags, 213
LLGL, 77
Red
LLGL, 101
reduceDeviceMemoryFragmentation
LLGL::RendererConfigurationVulkan, 301
reference
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::StencilFaceDescriptor, 398
LLGL::UTF8String, 433
referenceDynamic
LLGL::StencilDescriptor, 397
Reflect
LLGL::Shader, 373
refreshRate
LLGL::DisplayModeDescriptor, 222
Release
LLGL::Image, 266
LLGL::RenderSystem, 342, 343
RemoveEventListener
LLGL::Canvas, 159
LLGL::Window, 456
rend
LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::SmallVector, 392
LLGL::UTF8String, 436
renderCondition
LLGL::QueryHeapDescriptor, 292
RenderConditionMode
LLGL, 92
renderConditionSections
LLGL::FrameProfile, 250
RenderPass
LLGL::InterfaceID, 275
renderPass
LLGL::GraphicsPipelineDescriptor, 255
LLGL::RenderTargetDescriptor, 356
RenderPass.h, 488
RenderPassDesc
Global utility functions, especially to fill descriptor structures., 44
RenderPassFlags.h, 488
renderPassSections
LLGL::FrameProfile, 250
RenderSystem
LLGL::InterfaceID, 274
LLGL::RenderSystem, 331
RenderSystem.h, 489
RenderSystemChild
LLGL::InterfaceID, 274
RenderSystemChild.h, 491
RenderSystemDeleter
LLGL::RenderSystemDeleter, 348
RenderSystemDeleterFuncPtr
LLGL::RenderSystemDeleter, 348
RenderSystemDescriptor
LLGL::RenderSystemDescriptor, 349
RenderSystemFlags.h, 491
RenderSystemPtr
LLGL, 73
RenderTarget
LLGL::InterfaceID, 275
RenderTarget.h, 493
RenderTargetFlags.h, 493
RenderTargetIndex
LLGL, 100
rendererConfig
LLGL::RenderSystemDescriptor, 350
rendererConfigSize
LLGL::RenderSystemDescriptor, 350
RendererConfiguration.h, 486
rendererName
LLGL::RendererInfo, 305
RenderingDebugger
LLGL::RenderingDebugger, 308
LLGL::RenderingDebugger::Message, 278
RenderingDebugger.h, 487
RenderingProfiler.h, 488
Repeat
LLGL, 93
Replace
LLGL, 98
Report
LLGL::InterfaceID, 274
Report.h, 493
ReportCallback
LLGL::Log, 114
ReportType
LLGL::Log, 114
reserve
LLGL::SmallVector, 393
Reserved
LLGL::InterfaceID, 276
LLGL::RendererID, 304
Reset
LLGL::Image, 266

ResetDisplayFullscreenMode
LLGL::SwapChain, 407
ResetDisplayMode
LLGL::Display, 220
ResetPixelFormat
LLGL::Surface, 403
ResetResourceSlots
LLGL::CommandBuffer, 195
resizable
LLGL::WindowDescriptor, 459
Resize
LLGL::Image, 266, 267
resize
LLGL::SmallVector, 393
ResizeBuffers
LLGL::SwapChain, 407
ResizeBuffersPrimary
LLGL::SwapChain, 408
resolution
LLGL::DisplayModeDescriptor, 222
LLGL::RenderTargetDescriptor, 356
LLGL::SwapChainDescriptor, 411
Resource
LLGL::InterfaceID, 275
resource
LLGL::ResourceViewDescriptor, 365
Resource.h, 494
ResourceFlags.h, 494
ResourceHeap
LLGL::InterfaceID, 275
ResourceHeap.h, 495
resourceHeapBindings
LLGL::FrameProfile, 250
ResourceHeapDescriptor
LLGL::ResourceHeapDescriptor, 362
ResourceHeapFlags.h, 495
ResourceType
LLGL, 92
ResourceViewDescriptor
LLGL::ResourceViewDescriptor, 364
resources
LLGL::ShaderReflection, 383
Return
LLGL, 85
RevSubtract
LLGL, 75
reverse_iterator
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::UTF8String, 433
RG
LLGL, 84
Right
LLGL, 85
Round
LLGL::GrowStrategyRoundUpPow2, 258, 259

S

LLGL, 86
SPIRV_100
LLGL, 97
SPIRV
LLGL, 97
SampleID
LLGL, 100
SampleMask
LLGL, 100
sampleMask
LLGL::BlendDescriptor, 143
Sampled
LLGL::BindFlags, 137
sampledBufferBindings
LLGL::FrameProfile, 250
sampledTextureBindings
LLGL::FrameProfile, 250
Sampler
LLGL::InterfaceID, 275
LLGL::Sampler, 366
LLGL, 93, 103
Sampler.h, 495
SamplerAddressMode
LLGL, 93
samplerBindings
LLGL::FrameProfile, 251
SamplerFilter
LLGL, 94
SamplerFlags.h, 496
samples
LLGL::RenderPassDescriptor, 327
LLGL::RenderTargetDescriptor, 356
LLGL::SwapChainDescriptor, 411
LLGL::TextureDescriptor, 420
SamplesPassed
LLGL, 92
Scissor
LLGL::Scissor, 370
scissorTestEnabled
LLGL::RasterizerDescriptor, 298
scissors
LLGL::GraphicsPipelineDescriptor, 255
screen
LLGL::NativeContextHandle, 282
ScreenOrigin
LLGL, 95
screenOrigin
LLGL::RenderingCapabilities, 306
ScrollLock
LLGL, 87
Secondary
LLGL::CommandBufferFlags, 207
Select
LLGL, 85
semanticIndex
LLGL::VertexAttribute, 439
SeparateShader
LLGL::ShaderCompileFlags, 376

Set
LLGL, 89
SetBehavior
LLGL::Window, 456
SetBlendFactor
LLGL::CommandBuffer, 196
SetCursorPosition
LLGL::Display, 221
SetDebugGroup
LLGL::RenderingDebugger, 310
SetDesc
LLGL::Window, 456
SetDisplayFullscreenMode
LLGL::SwapChain, 408
SetDisplayMode
LLGL::Display, 221
SetGraphicsAPIDependentState
LLGL::CommandBuffer, 197
SetIndexBuffer
LLGL::CommandBuffer, 197, 198
SetName
LLGL::RenderSystemChild, 346
SetOrCreateSurface
LLGL::SwapChain, 408
SetPipelineState
LLGL::CommandBuffer, 198
SetPosition
LLGL::Window, 456
SetRendererInfo
LLGL::RenderSystem, 343
SetRenderingCaps
LLGL::RenderSystem, 343
SetReportCallback
LLGL::Log, 115
SetReportCallbackStd
LLGL::Log, 115
SetReportLimit
LLGL::Log, 116
SetResource
LLGL::CommandBuffer, 199
SetResourceHeap
LLGL::CommandBuffer, 199
SetScissor
LLGL::CommandBuffer, 200
SetScissors
LLGL::CommandBuffer, 200
SetSize
LLGL::Window, 456
SetSlot
LLGL::VertexFormat, 442
SetSource
LLGL::RenderingDebugger, 310
SetStencilReference
LLGL::CommandBuffer, 201
SetStride
LLGL::VertexFormat, 442, 443
SetTitle
LLGL::Canvas, 160
LLGL::Window, 456
SetUniform
LLGL::CommandBuffer, 201
SetUniforms
LLGL::CommandBuffer, 202
SetVertexBuffer
LLGL::CommandBuffer, 202
SetVertexBufferArray
LLGL::CommandBuffer, 203
SetViewport
LLGL::CommandBuffer, 203
SetViewports
LLGL::CommandBuffer, 203
SetVsyncInterval
LLGL::SwapChain, 409
Shader
LLGL::InterfaceID, 275
LLGL::Shader, 372
Shader.h, 496
ShaderDescFromFile
Global utility functions, especially to fill descriptor structures., 44
ShaderDescriptor
LLGL::ShaderDescriptor, 378
ShaderFlags.h, 497
ShaderMacro
LLGL::ShaderMacro, 382
ShaderReflection.h, 498
ShaderSourceType
LLGL, 95
ShaderType
LLGL, 95
ShadingLanguage
LLGL, 96
shadingLanguageName
LLGL::RendererInfo, 305
shadingLanguages
LLGL::RenderingCapabilities, 307
ShareSurfaceAndConfig
LLGL::SwapChain, 409
Shift
LLGL, 85
Show
LLGL::Window, 456
ShowCursor
LLGL::Display, 221
shrink_to_fit
LLGL::SmallVector, 393
size
LLGL::ArrayView, 123
LLGL::BasicStringView, 135
LLGL::BufferDescriptor, 153
LLGL::BufferViewDescriptor, 155
LLGL::ShaderUniform, 386
LLGL::SmallVector, 393
LLGL::UTF8String, 436
LLGL::WindowDescriptor, 459
size_type

LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::UTF8String, 433
Sleep
LLGL, 86
slopeFactor
LLGL::DepthBiasDescriptor, 214
slot
LLGL::BindingDescriptor, 140
LLGL::VertexAttribute, 440
SmallVector
LLGL::SmallVector, 389
SmallVector.h, 499
Snapshot
LLGL, 85
source
LLGL::ShaderDescriptor, 380
sourceSize
LLGL::ShaderDescriptor, 380
sourceType
LLGL::ShaderDescriptor, 380
Space
LLGL, 85
Src1Alpha
LLGL, 76
Src1Color
LLGL, 76
SrcAlpha
LLGL, 75
srcAlpha
LLGL::BlendTargetDescriptor, 145
SrcAlphaSaturate
LLGL, 76
SrcColor
LLGL, 75
srcColor
LLGL::BlendTargetDescriptor, 145
SrcImageDescriptor
LLGL::SrcImageDescriptor, 394
stageFlags
LLGL::BindingDescriptor, 140
StaticLimits.h, 500
LLGL_MAX_NUM_ATTACHMENTS, 500
LLGL_MAX_NUM_COLOR_ATTACHMENTS, 500
LLGL_MAX_NUM_SAMPLES, 500
LLGL_MAX_NUM_SO_BUFFERS, 500
LLGL_MAX_NUM_VIEWPORTS_AND_SCISS↩ ORS, 500
Stencil
LLGL::ClearFlags, 161
LLGL, 75, 100
stencil
LLGL::ClearValue, 163
LLGL::GraphicsPipelineDescriptor, 255
stencilAttachment
LLGL::RenderPassDescriptor, 327
stencilBits
LLGL::SwapChainDescriptor, 412
StencilFace
LLGL, 97
stencilFailOp
LLGL::StencilFaceDescriptor, 399
StencilOp
LLGL, 97
Storage
LLGL::BarrierFlags, 131
LLGL::BindFlags, 137
StorageBuffer
LLGL::BarrierFlags, 130
storageBufferBindings
LLGL::FrameProfile, 251
StorageBufferDesc
Global utility functions, especially to fill descriptor structures., 44
StorageBufferType
LLGL, 98
storageBufferType
LLGL::ShaderResource, 385
StorageTexture
LLGL::BarrierFlags, 130
storageTextureBindings
LLGL::FrameProfile, 251
Store
LLGL, 74
storeOp
LLGL::AttachmentFormatDescriptor, 129
StreamOutOverflow
LLGL, 92
StreamOutPrimitivesWritten
LLGL, 92
StreamOutputBuffer
LLGL::BindFlags, 137
streamOutputSections
LLGL::FrameProfile, 251
stride
LLGL::BufferDescriptor, 153
LLGL::VertexAttribute, 440
StringView
LLGL, 73
StringView.h, 502
Strings.h, 501
StructuredBuffer
LLGL, 98
Submit
LLGL::CommandQueue, 209, 210
subresource
LLGL::TextureRegion, 424
LLGL::TextureViewDescriptor, 429
substr
LLGL::BasicStringView, 135
Subtract
LLGL, 75
SupportsGenerateMips
LLGL::FormatFlags, 240
SupportsMips

LLGL::FormatFlags, 240
SupportsRenderTarget
LLGL::FormatFlags, 240
SupportsTexture1D
LLGL::FormatFlags, 240
SupportsTexture2D
LLGL::FormatFlags, 241
SupportsTexture3D
LLGL::FormatFlags, 241
SupportsTextureCube
LLGL::FormatFlags, 241
SupportsVertex
LLGL::FormatFlags, 241
Surface
LLGL::InterfaceID, 274
Surface.h, 502
Swap
LLGL::Image, 267
swap
LLGL::SmallVector, 393
swapBuffers
LLGL::SwapChainDescriptor, 412
SwapChain
LLGL::InterfaceID, 275
LLGL::SwapChain, 405
SwapChain.h, 503
SwapChainFlags.h, 503
SwitchFullscreen
LLGL::SwapChain, 409
swizzle
LLGL::TextureViewDescriptor, 429
SystemValue
LLGL, 99
systemValue
LLGL::FragmentAttribute, 243
LLGL::VertexAttribute, 440
SystemValue.h, 503

T
LLGL, 86
Tab
LLGL, 85
Tags.h, 504
targets
LLGL::BlendDescriptor, 143
TessControl
LLGL, 96
tessControlShader
LLGL::GraphicsPipelineDescriptor, 255
tessControlShaderInvocations
LLGL::QueryPipelineStatistics, 295
TessControlStage
LLGL::StageFlags, 396
TessEvaluation
LLGL, 96
tessEvaluationShader
LLGL::GraphicsPipelineDescriptor, 256
tessEvaluationShaderInvocations
LLGL::QueryPipelineStatistics, 295
TessEvaluationStage
LLGL::StageFlags, 396
tessFactorBufferSlot
LLGL::MetalDependentStateDescriptor, 279
tessellation
LLGL::GraphicsPipelineDescriptor, 255
TessellationPartition
LLGL, 101
testEnabled
LLGL::DepthDescriptor, 215
LLGL::StencilDescriptor, 397
Texture
LLGL::InterfaceID, 275
LLGL::Texture, 415
LLGL, 93
texture
LLGL::AttachmentDescriptor, 127
Texture utility functions to determine texture dimension and buffer sizes., 47
GetMemoryFootprint, 48
GetMipExtent, 49
IsArrayTexture, 50
IsCubeTexture, 50
IsMipMappedTexture, 50
IsMultiSampleTexture, 50
IsTextureSwizzleIdentity, 50
NumMipDimensions, 51
NumMipLevels, 51, 52
NumMipTexels, 52, 53
NumTextureDimensions, 53
Texture.h, 504
Texture1DArray
LLGL, 102
Texture1DArrayDesc
Global utility functions, especially to fill descriptor structures., 45
Texture1DDesc
Global utility functions, especially to fill descriptor structures., 45
Texture1D
LLGL, 102
Texture2DArray
LLGL, 102
Texture2DArrayDesc
Global utility functions, especially to fill descriptor structures., 45
Texture2DDesc
Global utility functions, especially to fill descriptor structures., 45
Texture2DMSArray
LLGL, 102
Texture2DMSArrayDesc
Global utility functions, especially to fill descriptor structures., 45
Texture2DMSDesc
Global utility functions, especially to fill descriptor structures., 46
Texture2DMS

LLGL, 102
Texture2D
LLGL, 102
Texture3DDesc
Global utility functions, especially to fill descriptor structures., 46
Texture3D
LLGL, 102
textureCopies
LLGL::FrameProfile, 251
TextureCube
LLGL, 102
TextureCubeArray
LLGL, 102
TextureCubeArrayDesc
Global utility functions, especially to fill descriptor structures., 46
TextureCubeDesc
Global utility functions, especially to fill descriptor structures., 46
TextureFlags.h, 505
textureFormats
LLGL::RenderingCapabilities, 307
TextureLocation
LLGL::TextureLocation, 422
textureReads
LLGL::FrameProfile, 252
TextureRegion
LLGL::TextureRegion, 424
TextureSubresource
LLGL::TextureSubresource, 425
TextureSwizzle
LLGL, 101
TextureType
LLGL, 101
textureView
LLGL::ResourceViewDescriptor, 365
textureWrites
LLGL::FrameProfile, 252
Tick
LLGL::Timer, 116
TimeElapsed
LLGL, 92
timeRecordingEnabled
LLGL::RenderingProfiler, 324
timeRecords
LLGL::FrameProfile, 252
Timer.h, 506
title
LLGL::CanvasDescriptor, 160
LLGL::WindowDescriptor, 460
ToRGBA
LLGL::Color< T, 3u >, 171
ToRGB
LLGL::Color< T, 4u >, 176
ToReportString
LLGL::RenderingDebugger::Message, 278
ToString
Global type-to-string conversion functions., 39, 40
ToWCharArray
LLGL::UTF8String, 436
TriangleList
LLGL, 91
TriangleListAdjacency
LLGL, 91
TriangleStrip
LLGL, 91
TriangleStripAdjacency
LLGL, 91
type
LLGL::AttachmentDescriptor, 127
LLGL::BindingDescriptor, 140
LLGL::QueryHeapDescriptor, 293
LLGL::ShaderDescriptor, 381
LLGL::ShaderUniform, 386
LLGL::TextureDescriptor, 421
LLGL::TextureViewDescriptor, 430
TypeInfo.h, 507
TypeNames.h, 507
TypedBuffer
LLGL, 98
Types.h, 508

U
LLGL, 86
UInt1
LLGL, 103
UInt16
LLGL, 78
UInt2
LLGL, 103
UInt3
LLGL, 103
UInt32
LLGL, 78
UInt4
LLGL, 103
UInt8
LLGL, 78
UTF8String
LLGL::UTF8String, 433, 434
UTF8String.h, 509
Undefined
LLGL::RendererID, 304
LLGL, 74, 78, 81, 89, 93, 96, 98, 100, 101, 103
UndefinedBehavior
LLGL, 79
UniformLocation
LLGL, 73
UniformType
LLGL, 102
uniforms
LLGL::ShaderReflection, 383
Unload
LLGL::RenderSystem, 343
UnmapBuffer
LLGL::RenderSystem, 344

UnsupportedFeature
LLGL, 79
Up
LLGL, 85
UpdateBuffer
LLGL::CommandBuffer, 204
UpperLeft
LLGL, 95
Utility.h, 510

V
LLGL, 86
ValidateMipResolution
LLGL::RenderTarget, 354
ValidateRenderingCaps
LLGL, 110
ValidateRenderingCapsFunc
Global type aliases to callback interfaces., 25
ValidateResolution
LLGL::RenderTarget, 354
value_type
LLGL::ArrayView, 122
LLGL::BasicStringView, 133
LLGL::SmallVector, 389
LLGL::UTF8String, 433
values
LLGL::FrameProfile, 252
VaryingBehavior
LLGL, 104
vendor
LLGL::VideoAdapterDescriptor, 445
vendorName
LLGL::RendererInfo, 305
Version.h, 511
VersionBitmask
LLGL, 97
Vertex
LLGL, 96
vertex
LLGL::ShaderDescriptor, 381
LLGL::ShaderReflection, 384
vertexAttribs
LLGL::BufferDescriptor, 153
VertexAttribute
LLGL::VertexAttribute, 437, 438
VertexAttribute.h, 512
VertexBuffer
LLGL::BindFlags, 136
vertexBufferBindings
LLGL::FrameProfile, 252
VertexBufferDesc
Global utility functions, especially to fill descriptor structures., 46
VertexFormat.h, 512
VertexID
LLGL, 100
vertexOffset
LLGL::DrawIndexedIndirectArguments, 224
vertexShader
LLGL::GraphicsPipelineDescriptor, 256
vertexShaderInvocations
LLGL::QueryPipelineStatistics, 295
VertexStage
LLGL::StageFlags, 396
VideoAdapter.h, 513
videoMemory
LLGL::VideoAdapterDescriptor, 445
view
LLGL::NativeHandle, 283
Viewport
LLGL::Viewport, 447, 448
ViewportIndex
LLGL, 101
viewports
LLGL::GraphicsPipelineDescriptor, 256
visible
LLGL::WindowDescriptor, 460
visual
LLGL::NativeContextHandle, 282
LLGL::NativeHandle, 283
VolumeDown
LLGL, 88
VolumeMute
LLGL, 87
VolumeUp
LLGL, 88
Vulkan
LLGL::RendererID, 304

W
LLGL, 86
WStringView
LLGL, 73
Wait
LLGL, 92
WaitFence
LLGL::CommandQueue, 210
WaitIdle
LLGL::CommandQueue, 210
WaitInverted
LLGL, 92
Warning
LLGL::Log, 114
WarningType
LLGL, 103
WarningsAreErrors
LLGL::ShaderCompileFlags, 376
wholeSize
LLGL::Constants, 113
width
LLGL::Extent2D, 235
LLGL::Extent3D, 236
LLGL::Scissor, 371
LLGL::Viewport, 448
Win32NativeHandle.h, 513
Window
LLGL::InterfaceID, 274
LLGL::Window, 451

LLGL::Window::EventListener, 232
window
LLGL::NativeHandle, 283
Window.h, 513
Window_EventListener
LLGL::InterfaceID, 274
windowContext
LLGL::WindowDescriptor, 460
WindowFlags.h, 514
WindowedMode
LLGL::ResizeBuffersFlags, 359
Wireframe
LLGL, 90
workGroupSize
LLGL::ComputeShaderAttributes, 212
Write
LLGL::CPUAccessFlags, 213
WriteBuffer
LLGL::RenderSystem, 344
WriteDiscard
LLGL, 77
writeEnabled
LLGL::DepthDescriptor, 215
writeMask
LLGL::StencilFaceDescriptor, 399
WriteOnly
LLGL, 77
WritePixels
LLGL::Image, 268
WriteResourceHeap
LLGL::RenderSystem, 344
WriteTexture
LLGL::RenderSystem, 345

X
LLGL, 86
x
LLGL::Offset2D, 286
LLGL::Offset3D, 287
LLGL::Scissor, 371
LLGL::Viewport, 449
XButton1
LLGL, 85
XButton2
LLGL, 85
XOR
LLGL, 89

Y
LLGL, 86
y
LLGL::Offset2D, 286
LLGL::Offset3D, 287
LLGL::Scissor, 371
LLGL::Viewport, 449

Z
LLGL, 86
z
LLGL::Offset3D, 287
Zero
LLGL, 75, 98, 101
ZeroToOne
LLGL, 76
Zoom
LLGL, 88