

Easy OpenGL

Generated by Doxygen 1.16.1

1 Introduction	1
1.1 Recommended Setup	1
2 Directory Hierarchy	3
2.1 Directories	3
3 Namespace Index	5
3.1 Namespace List	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Data Structure Index	9
5.1 Data Structures	9
6 File Index	11
6.1 File List	11
7 Directory Documentation	13
7.1 inc/EGL Directory Reference	13
7.2 inc Directory Reference	13
8 Namespace Documentation	15
8.1 egl Namespace Reference	15
8.1.1 Typedef Documentation	17
8.1.1.1 UniformData	17
8.1.2 Enumeration Type Documentation	17
8.1.2.1 BufferFlag	17
8.1.2.2 BufferType	17
8.1.2.3 BufferUsage	18
8.1.2.4 MapUsage	19
8.1.2.5 ShaderType	19
8.1.3 Function Documentation	20
8.1.3.1 glCheckError()	20
8.1.3.2 glErrorString()	20
8.1.3.3 initGLFW()	21
8.1.3.4 operator&() [1/2]	21
8.1.3.5 operator&() [2/2]	21
8.1.3.6 operator" () [1/2]	21
8.1.3.7 operator" () [2/2]	22
8.1.3.8 operator" =() [1/2]	22
8.1.3.9 operator" =() [2/2]	22
8.1.3.10 pollEvents()	22
8.1.3.11 shaderTypeToString()	22
8.1.3.12 terminateGLFW()	23

8.1.3.13 toGLenum() [1/5]	23
8.1.3.14 toGLenum() [2/5]	23
8.1.3.15 toGLenum() [3/5]	23
8.1.3.16 toGLenum() [4/5]	23
8.1.3.17 toGLenum() [5/5]	24
8.1.3.18 validateBufferFlag()	24
8.1.3.19 validateMapUsage()	24
9 Data Structure Documentation	27
9.1 egl::_uniformData Struct Reference	27
9.1.1 Field Documentation	27
9.1.1.1 arraySize	27
9.1.1.2 glType	27
9.1.1.3 location	27
9.2 egl::Buffer Class Reference	28
9.2.1 Constructor & Destructor Documentation	29
9.2.1.1 Buffer() [1/4]	29
9.2.1.2 Buffer() [2/4]	29
9.2.1.3 Buffer() [3/4]	29
9.2.1.4 Buffer() [4/4]	29
9.2.1.5 ~Buffer()	29
9.2.2 Member Function Documentation	29
9.2.2.1 _check()	29
9.2.2.2 _delete()	29
9.2.2.3 bind()	29
9.2.2.4 getSubData()	30
9.2.2.5 getType()	30
9.2.2.6 map()	30
9.2.2.7 operator=(()) [1/2]	31
9.2.2.8 operator=(()) [2/2]	31
9.2.2.9 setData() [1/2]	31
9.2.2.10 setData() [2/2]	32
9.2.2.11 setStorage() [1/2]	32
9.2.2.12 setStorage() [2/2]	32
9.2.2.13 setSubData()	33
9.2.2.14 size()	33
9.2.2.15 unmap()	33
9.2.3 Field Documentation	34
9.2.3.1 _flags	34
9.2.3.2 _id	34
9.2.3.3 _mapped	34
9.2.3.4 _mapUsage	34

9.2.3.5 <code>_type</code>	34
9.3 <code>egl::Program</code> Class Reference	34
9.3.1 Detailed Description	40
9.3.2 Constructor & Destructor Documentation	41
9.3.2.1 <code>Program()</code> [1/3]	41
9.3.2.2 <code>Program()</code> [2/3]	41
9.3.2.3 <code>Program()</code> [3/3]	41
9.3.2.4 <code>~Program()</code>	41
9.3.3 Member Function Documentation	41
9.3.3.1 <code>_check()</code>	41
9.3.3.2 <code>_delete()</code>	41
9.3.3.3 <code>_ensure()</code>	41
9.3.3.4 <code>_getError()</code>	41
9.3.3.5 <code>_queryUniformData()</code>	41
9.3.3.6 <code>_setupUniform()</code>	42
9.3.3.7 <code>attach()</code>	42
9.3.3.8 <code>attached()</code>	42
9.3.3.9 <code>bind()</code>	43
9.3.3.10 <code>detach()</code>	43
9.3.3.11 <code>getUniform()</code> [1/42]	44
9.3.3.12 <code>getUniform()</code> [2/42]	44
9.3.3.13 <code>getUniform()</code> [3/42]	44
9.3.3.14 <code>getUniform()</code> [4/42]	44
9.3.3.15 <code>getUniform()</code> [5/42]	45
9.3.3.16 <code>getUniform()</code> [6/42]	45
9.3.3.17 <code>getUniform()</code> [7/42]	45
9.3.3.18 <code>getUniform()</code> [8/42]	45
9.3.3.19 <code>getUniform()</code> [9/42]	45
9.3.3.20 <code>getUniform()</code> [10/42]	45
9.3.3.21 <code>getUniform()</code> [11/42]	46
9.3.3.22 <code>getUniform()</code> [12/42]	46
9.3.3.23 <code>getUniform()</code> [13/42]	46
9.3.3.24 <code>getUniform()</code> [14/42]	46
9.3.3.25 <code>getUniform()</code> [15/42]	46
9.3.3.26 <code>getUniform()</code> [16/42]	46
9.3.3.27 <code>getUniform()</code> [17/42]	47
9.3.3.28 <code>getUniform()</code> [18/42]	47
9.3.3.29 <code>getUniform()</code> [19/42]	47
9.3.3.30 <code>getUniform()</code> [20/42]	47
9.3.3.31 <code>getUniform()</code> [21/42]	47
9.3.3.32 <code>getUniform()</code> [22/42]	48
9.3.3.33 <code>getUniform()</code> [23/42]	48

9.3.3.34 getUniform() [24/42]	48
9.3.3.35 getUniform() [25/42]	48
9.3.3.36 getUniform() [26/42]	49
9.3.3.37 getUniform() [27/42]	49
9.3.3.38 getUniform() [28/42]	49
9.3.3.39 getUniform() [29/42]	49
9.3.3.40 getUniform() [30/42]	49
9.3.3.41 getUniform() [31/42]	49
9.3.3.42 getUniform() [32/42]	50
9.3.3.43 getUniform() [33/42]	50
9.3.3.44 getUniform() [34/42]	50
9.3.3.45 getUniform() [35/42]	50
9.3.3.46 getUniform() [36/42]	50
9.3.3.47 getUniform() [37/42]	50
9.3.3.48 getUniform() [38/42]	51
9.3.3.49 getUniform() [39/42]	51
9.3.3.50 getUniform() [40/42]	51
9.3.3.51 getUniform() [41/42]	51
9.3.3.52 getUniform() [42/42]	51
9.3.3.53 getUniformLocation()	52
9.3.3.54 link()	52
9.3.3.55 linked()	52
9.3.3.56 operator=() [1/2]	53
9.3.3.57 operator=() [2/2]	53
9.3.3.58 operator[]() [1/4]	53
9.3.3.59 operator[]() [2/4]	53
9.3.3.60 operator[]() [3/4]	54
9.3.3.61 operator[]() [4/4]	54
9.3.3.62 reset()	55
9.3.3.63 setUniform() [1/42]	55
9.3.3.64 setUniform() [2/42]	55
9.3.3.65 setUniform() [3/42]	55
9.3.3.66 setUniform() [4/42]	56
9.3.3.67 setUniform() [5/42]	56
9.3.3.68 setUniform() [6/42]	56
9.3.3.69 setUniform() [7/42]	56
9.3.3.70 setUniform() [8/42]	56
9.3.3.71 setUniform() [9/42]	56
9.3.3.72 setUniform() [10/42]	57
9.3.3.73 setUniform() [11/42]	57
9.3.3.74 setUniform() [12/42]	57
9.3.3.75 setUniform() [13/42]	57

9.3.3.76 setUniform() [14/42]	57
9.3.3.77 setUniform() [15/42]	57
9.3.3.78 setUniform() [16/42]	58
9.3.3.79 setUniform() [17/42]	58
9.3.3.80 setUniform() [18/42]	58
9.3.3.81 setUniform() [19/42]	58
9.3.3.82 setUniform() [20/42]	59
9.3.3.83 setUniform() [21/42]	59
9.3.3.84 setUniform() [22/42]	59
9.3.3.85 setUniform() [23/42]	59
9.3.3.86 setUniform() [24/42]	60
9.3.3.87 setUniform() [25/42]	60
9.3.3.88 setUniform() [26/42]	60
9.3.3.89 setUniform() [27/42]	60
9.3.3.90 setUniform() [28/42]	60
9.3.3.91 setUniform() [29/42]	60
9.3.3.92 setUniform() [30/42]	61
9.3.3.93 setUniform() [31/42]	61
9.3.3.94 setUniform() [32/42]	61
9.3.3.95 setUniform() [33/42]	61
9.3.3.96 setUniform() [34/42]	61
9.3.3.97 setUniform() [35/42]	61
9.3.3.98 setUniform() [36/42]	62
9.3.3.99 setUniform() [37/42]	62
9.3.3.100 setUniform() [38/42]	62
9.3.3.101 setUniform() [39/42]	62
9.3.3.102 setUniform() [40/42]	62
9.3.3.103 setUniform() [41/42]	63
9.3.3.104 setUniform() [42/42]	63
9.3.3.105 unbind()	63
9.3.4 Field Documentation	63
9.3.4.1 _id	63
9.3.4.2 _linked	64
9.3.4.3 _uniformData	64
9.3.4.4 _uniformIndexMap	64
9.3.4.5 _uniformLocationIndexMap	64
9.4 egl::ProgramLinkError Class Reference	64
9.4.1 Detailed Description	64
9.4.2 Constructor & Destructor Documentation	65
9.4.2.1 ProgramLinkError()	65
9.5 egl::ProgramValidateError Class Reference	65
9.5.1 Detailed Description	65

9.5.2 Constructor & Destructor Documentation	66
9.5.2.1 ProgramValidateError()	66
9.6 egl::Shader Class Reference	66
9.6.1 Detailed Description	67
9.6.2 Constructor & Destructor Documentation	67
9.6.2.1 Shader() [1/8]	67
9.6.2.2 Shader() [2/8]	68
9.6.2.3 Shader() [3/8]	68
9.6.2.4 Shader() [4/8]	68
9.6.2.5 Shader() [5/8]	69
9.6.2.6 Shader() [6/8]	69
9.6.2.7 Shader() [7/8]	69
9.6.2.8 Shader() [8/8]	69
9.6.2.9 ~Shader()	70
9.6.3 Member Function Documentation	70
9.6.3.1 _check()	70
9.6.3.2 _delete()	70
9.6.3.3 _ensure()	70
9.6.3.4 _getError()	70
9.6.3.5 compile() [1/4]	70
9.6.3.6 compile() [2/4]	71
9.6.3.7 compile() [3/4]	71
9.6.3.8 compile() [4/4]	72
9.6.3.9 compiled()	72
9.6.3.10 getType()	72
9.6.3.11 operator=() [1/2]	72
9.6.3.12 operator=() [2/2]	72
9.6.3.13 reset()	73
9.6.4 Field Documentation	73
9.6.4.1 _compiled	73
9.6.4.2 _id	73
9.6.4.3 _type	73
9.6.4.4 Program	73
9.7 egl::ShaderCompileError Class Reference	73
9.7.1 Detailed Description	74
9.7.2 Constructor & Destructor Documentation	74
9.7.2.1 ShaderCompileError()	74
9.8 egl::Program::UniformProxy Class Reference	74
9.8.1 Constructor & Destructor Documentation	75
9.8.1.1 UniformProxy() [1/3]	75
9.8.1.2 UniformProxy() [2/3]	75
9.8.1.3 UniformProxy() [3/3]	76

9.8.2 Member Function Documentation	76
9.8.2.1 operator float()	76
9.8.2.2 operator glm::ivec2()	76
9.8.2.3 operator glm::ivec3()	76
9.8.2.4 operator glm::ivec4()	76
9.8.2.5 operator glm::mat2()	76
9.8.2.6 operator glm::mat2x3()	76
9.8.2.7 operator glm::mat2x4()	76
9.8.2.8 operator glm::mat3()	76
9.8.2.9 operator glm::mat3x2()	76
9.8.2.10 operator glm::mat3x4()	77
9.8.2.11 operator glm::mat4()	77
9.8.2.12 operator glm::mat4x2()	77
9.8.2.13 operator glm::mat4x3()	77
9.8.2.14 operator glm::uvec2()	77
9.8.2.15 operator glm::uvec3()	77
9.8.2.16 operator glm::uvec4()	77
9.8.2.17 operator glm::vec2()	77
9.8.2.18 operator glm::vec3()	77
9.8.2.19 operator glm::vec4()	77
9.8.2.20 operator int()	78
9.8.2.21 operator unsigned int()	78
9.8.2.22 operator=() [1/23]	78
9.8.2.23 operator=() [2/23]	78
9.8.2.24 operator=() [3/23]	78
9.8.2.25 operator=() [4/23]	78
9.8.2.26 operator=() [5/23]	78
9.8.2.27 operator=() [6/23]	78
9.8.2.28 operator=() [7/23]	78
9.8.2.29 operator=() [8/23]	79
9.8.2.30 operator=() [9/23]	79
9.8.2.31 operator=() [10/23]	79
9.8.2.32 operator=() [11/23]	79
9.8.2.33 operator=() [12/23]	79
9.8.2.34 operator=() [13/23]	79
9.8.2.35 operator=() [14/23]	79
9.8.2.36 operator=() [15/23]	79
9.8.2.37 operator=() [16/23]	79
9.8.2.38 operator=() [17/23]	80
9.8.2.39 operator=() [18/23]	80
9.8.2.40 operator=() [19/23]	80
9.8.2.41 operator=() [20/23]	80

9.8.2.42 operator=() [21/23]	80
9.8.2.43 operator=() [22/23]	80
9.8.2.44 operator=() [23/23]	80
9.9 egl::Program::UniformProxyConst Class Reference	80
9.9.1 Constructor & Destructor Documentation	81
9.9.1.1 UniformProxyConst() [1/3]	81
9.9.1.2 UniformProxyConst() [2/3]	81
9.9.1.3 UniformProxyConst() [3/3]	81
9.9.2 Member Function Documentation	82
9.9.2.1 operator float()	82
9.9.2.2 operator glm::ivec2()	82
9.9.2.3 operator glm::ivec3()	82
9.9.2.4 operator glm::ivec4()	82
9.9.2.5 operator glm::mat2()	82
9.9.2.6 operator glm::mat2x3()	82
9.9.2.7 operator glm::mat2x4()	82
9.9.2.8 operator glm::mat3()	82
9.9.2.9 operator glm::mat3x2()	82
9.9.2.10 operator glm::mat3x4()	82
9.9.2.11 operator glm::mat4()	83
9.9.2.12 operator glm::mat4x2()	83
9.9.2.13 operator glm::mat4x3()	83
9.9.2.14 operator glm::uvec2()	83
9.9.2.15 operator glm::uvec3()	83
9.9.2.16 operator glm::uvec4()	83
9.9.2.17 operator glm::vec2()	83
9.9.2.18 operator glm::vec3()	83
9.9.2.19 operator glm::vec4()	83
9.9.2.20 operator int()	83
9.9.2.21 operator unsigned int()	84
9.9.2.22 operator=() [1/2]	84
9.9.2.23 operator=() [2/2]	84
9.10 egl::VertexBuffer Class Reference	84
9.11 egl::WindowContext Class Reference	84
9.11.1 Detailed Description	85
9.11.2 Constructor & Destructor Documentation	85
9.11.2.1 WindowContext() [1/3]	85
9.11.2.2 WindowContext() [2/3]	86
9.11.2.3 WindowContext() [3/3]	86
9.11.2.4 ~WindowContext()	86
9.11.3 Member Function Documentation	86
9.11.3.1 operator=() [1/2]	86

9.11.3.2 operator=() [2/2]	86
9.11.3.3 run()	86
9.11.3.4 shouldClose()	87
9.11.3.5 swapBuffers()	87
9.11.3.6 useContext()	87
9.11.4 Field Documentation	87
9.11.4.1 window	87
10 File Documentation	89
10.1 inc/EGL/buffer.h File Reference	89
10.2 buffer.h	90
10.3 inc/EGL/debug.h File Reference	92
10.3.1 Macro Definition Documentation	92
10.3.1.1 DEBUG_ONLY	92
10.3.1.2 GL_CALL	93
10.4 debug.h	93
10.5 inc/EGL/mainpage.md File Reference	93
10.6 inc/EGL/program.h File Reference	93
10.7 program.h	94
10.8 inc/EGL/shader.h File Reference	98
10.9 shader.h	99
10.10 inc/EGL/vertexBuffer.h File Reference	100
10.11 vertexBuffer.h	100
10.12 inc/EGL/windowContext.h File Reference	101
10.13 windowContext.h	101
Index	103

Chapter 1

Introduction

This project provides a low-level OpenGL abstraction implemented in modern C++, using try-catch based error handling and classes representing various OpenGL objects.

1.1 Recommended Setup

The recommended setup is to have your main code class inherit from `egl::WindowContext` and implement your logic in the provided virtual run function.

Calling `egl::WindowContext::useContext` is recommended to avoid potential errors.

Inside of `egl::WindowContext::run` any Easy OpenGL functionality can be used safely.

Chapter 2

Directory Hierarchy

2.1 Directories

EGL	13
buffer.h	89
debug.h	92
program.h	93
shader.h	98
vertexBuffer.h	100
windowContext.h	101
inc	13
EGL	13
buffer.h	89
debug.h	92
program.h	93
shader.h	98
vertexBuffer.h	100
windowContext.h	101

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

egl	15
-------------------------------	----

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

egl::_uniformData	27
egl::Buffer	28
egl::Program	34
std::runtime_error	
egl::ProgramLinkError	64
egl::ProgramValidateError	65
egl::ShaderCompileError	73
egl::Shader	66
egl::Program::UniformProxy	74
egl::Program::UniformProxyConst	80
egl::VertexBuffer	84
egl::WindowContext	84

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

egl::_uniformData	27
egl::Buffer	28
egl::Program	
Program class to abstract OpenGL Shader Programs	34
egl::ProgramLinkError	
Exception thrown when Program linking fails	64
egl::ProgramValidateError	
Exception thrown when Program validation fails, mainly used in debug builds	65
egl::Shader	
Shader class to abstract OpenGL Shaders	66
egl::ShaderCompileError	
Exception thrown when Shader compilation fails	73
egl::Program::UniformProxy	74
egl::Program::UniformProxyConst	80
egl::VertexBuffer	84
egl::WindowContext	
An abstract class to contain an window based application	84

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

inc/EGL/ buffer.h	89
inc/EGL/ debug.h	92
inc/EGL/ program.h	93
inc/EGL/ shader.h	98
inc/EGL/ vertexBuffer.h	100
inc/EGL/ windowContext.h	101

Chapter 7

Directory Documentation

7.1 inc/EGL Directory Reference

Files

- file [buffer.h](#)
- file [debug.h](#)
- file [program.h](#)
- file [shader.h](#)
- file [vertexBuffer.h](#)
- file [windowContext.h](#)

7.2 inc Directory Reference

Directories

- directory [EGL](#)

Chapter 8

Namespace Documentation

8.1 egl Namespace Reference

Data Structures

- class [Buffer](#)
- class [ProgramLinkError](#)

Exception thrown when [Program](#) linking fails.
- class [ProgramValidateError](#)

Exception thrown when [Program](#) validation fails, mainly used in debug builds.
- struct [_uniformData](#)
- class [Program](#)

Program class to abstract OpenGL [Shader](#) Programs.
- class [ShaderCompileError](#)

Exception thrown when [Shader](#) compilation fails.
- class [Shader](#)

Shader class to abstract OpenGL Shaders.
- class [VertexBuffer](#)
- class [WindowContext](#)

An abstract class to contain an window based application.

Typedefs

- typedef struct [egl::_uniformData](#) UniformData

Enumerations

- enum class [BufferType](#) {
 [Array](#) , [AtomicCounter](#) , [CopyRead](#) , [CopyWrite](#) ,
 [DispatchIndirect](#) , [DrawIndirect](#) , [ElementArray](#) , [PixelPack](#) ,
 [PixelUnpack](#) , [Query](#) , [ShaderStorage](#) , [Texture](#) ,
 [TransformFeedback](#) , [Uniform](#) }

Enum to indicate the type of [Buffer](#).

- enum class `BufferUsage` {
 StreamDraw , StreamRead , StreamCopy , StaticDraw ,
 StaticRead , StaticCopy , DynamicDraw , DynamicRead ,
 DynamicCopy }

Enum to indicate Buffer Usage.
- enum class `MapUsage` : `uint32_t` {
 `None` = 0 , `Read` = 1 << 0 , `Write` = 1 << 1 , `Persistent` = 1 << 2 ,
 `Coherent` = 1 << 3 , `InvalidRange` = 1 << 4 , `InvalidateBuffer` = 1 << 5 , `FlushExplicit` = 1 << 6 ,
 `Unsynchronized` = 1 << 7 }

Enum flags to indicate Buffer map usage.
- enum class `BufferFlag` : `uint32_t` {
 `None` = 0 , `DynamicStorage` = 1 << 0 , `MapRead` = 1 << 1 , `MapWrite` = 1 << 2 ,
 `MapPersistent` = 1 << 3 , `MapCoherent` = 1 << 4 , `ClientStorage` = 1 << 5 }

Enum flags for explicit Buffer usage in setStorage.
- enum class `ShaderType` {
 Fragment , Vertex , Geometry , TessEvaluation ,
 TessControl , Compute }

ShaderType enum to indicate usage.

Functions

- `MapUsage operator| (MapUsage a, MapUsage b)`
- `MapUsage operator& (MapUsage a, MapUsage b)`
- `MapUsage & operator|= (MapUsage &a, MapUsage b)`
- `BufferFlag operator| (BufferFlag a, BufferFlag b)`
- `BufferFlag operator& (BufferFlag a, BufferFlag b)`
- `BufferFlag & operator|= (BufferFlag &a, BufferFlag b)`
- `unsigned int toGLenum (BufferType type)`

Converts a BufferType enum into a GLenum.
- `unsigned int toGLenum (BufferUsage usage)`

Converts a BufferUsage enum into a GLenum.
- `unsigned int toGLenum (MapUsage usage)`

Converts a MapUsage enum into a GLenum.
- `unsigned int toGLenum (BufferFlag flag)`

Converts a BufferFlag enum into a GLenum.
- `bool validateMapUsage (MapUsage usage, std::string &error)`

Checks if the given MapUsage flag combination is valid.
- `bool validateBufferFlag (BufferFlag flag, std::string &error)`

Checks if the given BufferFlag flag combination is valid.
- `const char * glErrorString (unsigned int err)`

Gets a c-style string from from a OpenGL enum.
- `void glCheckError (const char *func, const char *file, int line)`

Checks and prints if any OpenGL error has occurred.
- `constexpr std::string shaderTypeToString (ShaderType type)`

Converts the ShaderType enum into a std::string.
- `unsigned int toGLenum (ShaderType type)`

Converts the ShaderType enum into an OpenGL enum.
- `bool initGLFW ()`

Initialize GLFW.
- `void terminateGLFW ()`

Terminates GLFW.
- `void pollEvents ()`

Polls GLFW events.

8.1.1 Typedef Documentation

8.1.1.1 UniformData

```
typedef struct egl::_uniformData egl::UniformData
```

8.1.2 Enumeration Type Documentation

8.1.2.1 BufferFlag

```
enum class egl::BufferFlag : uint32_t [strong]
```

Enum flags for explicit [Buffer](#) usage in setStorage.

Enumerator

None	
DynamicStorage	Indicates that the contents of the data store may be updated after creation through calls to glBufferSubData.
MapRead	Indicates that the data store may be mapped by the client for read access and a pointer in the client's address space obtained that may be read from.
MapWrite	Indicates that the data store may be mapped by the client for write access and a pointer in the client's address space obtained that may be written through.
MapPersistent	Indicates that the client may request that the server read from or write to the buffer while it is mapped. The client's pointer to the data store remains valid so long as the data store is mapped, even during execution of drawing or dispatch commands.
MapCoherent	Indicates that shared access to buffers that are simultaneously mapped for client access and are used by the server will be coherent, so long as that mapping is performed using glMapBufferRange.
ClientStorage	When all other criteria for the buffer storage allocation are met, this bit may be used by an implementation to determine whether to use storage that is local to the server or to the client to serve as the backing store for the buffer.

8.1.2.2 BufferType

```
enum class egl::BufferType [strong]
```

Enum to indicate the type of [Buffer](#).

Enumerator

Array	Vertex attributes.
AtomicCounter	Atomic counter storage.
CopyRead	Buffer copy source.
CopyWrite	Buffer copy destination.
DispatchIndirect	Indirect compute dispatch commands.

DrawIndirect	Indirect command arguments.
ElementArray	Vertex array indices.
PixelPack	Pixel read target.
PixelUnpack	Texture data source.
Query	Query result buffer.
ShaderStorage	Read-write storage for shaders.
Texture	Texture data buffer.
TransformFeedback	Transform feedback buffer.
Uniform	Uniform block storage.

8.1.2.3 BufferUsage

```
enum class egl::BufferUsage [strong]
```

Enum to indicate [Buffer](#) Usage.

The Usage of a [Buffer](#) can be split into two parts as follows:

The frequency of usage may be one of these:

Stream

- The data store contents will be modified once and used at most a few times.

Static

- The data store contents will be modified once and used many times.

Dynamic

- The data store contents will be modified repeatedly and used many times.

The nature of usage may be one of these:

Draw

- The data store contents are modified by the application, and used as the source for GL drawing and image specification commands.

Read

- The data store contents are modified by reading data from the GL, and used to return that data when queried by the application.

Copy

- The data store contents are modified by reading data from the GL, and used as the source for GL drawing and image specification commands.

Enumerator

StreamDraw	
StreamRead	
StreamCopy	
StaticDraw	
StaticRead	
StaticCopy	
DynamicDraw	
DynamicRead	
DynamicCopy	

8.1.2.4 MapUsage

```
enum class egl::MapUsage : uint32_t [strong]
```

Enum flags to indicate [Buffer](#) map usage.

Enumerator

None	
Read	Indicates that the returned pointer may be used to read buffer object data.
Write	Indicates that the returned pointer may be used to modify buffer object data.
Persistent	Indicates that the mapping is to be made in a persistent fashion and that the client intends to hold and use the returned pointer during subsequent GL operation.
Coherent	Indicates that a persistent mapping is also to be coherent. Coherent maps guarantee that the effect of writes to a buffer's data store by either the client or server will eventually become visible to the other without further intervention from the application.
InvalidRange	Indicates that the previous contents of the specified range may be discarded. Data within this range are undefined with the exception of subsequently written data.
InvalidateBuffer	Indicates that the previous contents of the entire buffer may be discarded. Data within the entire buffer are undefined with the exception of subsequently written data.
FlushExplicit	Indicates that one or more discrete subranges of the mapping may be modified. When this flag is set, modifications to each subrange must be explicitly flushed by calling <code>glFlushMappedBufferRange</code> .
Unsynchronized	Indicates that the GL should not attempt to synchronize pending operations on the buffer prior to returning from <code>glMapBufferRange</code> or <code>glMapNamedBufferRange</code> .

8.1.2.5 ShaderType

```
enum class egl::ShaderType [strong]
```

[ShaderType](#) enum to indicate usage.

Enumerator

Fragment	A Shader that is intended to run on the programmable fragment processor.
Vertex	A Shader that is intended to run on the programmable vertex processor.
Geometry	A Shader that is intended to run on the programmable geometry processor.
TessEvaluation	A Shader that is intended to run on the programmable tessellation processor in the evaluation stage.
TessControl	A Shader that is intended to run on the programmable tessellation processor in the control stage.
Compute	A Shader intended to run on the programmable compute processor.

8.1.3 Function Documentation**8.1.3.1 glCheckError()**

```
void egl::glCheckError (
    const char * func,
    const char * file,
    int line)
```

Checks and prints if any OpenGL error has occurred.

Iterates over the OpenGL error flags with glGetError and prints them until no error are found anymore.

Note

Mainly used in the GL_CALL macro in the backend of the Easy OpenGL abstraction.

Parameters

<i>func</i>	The line of code that is being checked as a c-style string for debug output
<i>file</i>	The file in which this function was called as a c-style string for debug output
<i>line</i>	The line on which this function was called for debug output

8.1.3.2 glErrorString()

```
const char * egl::glErrorString (
    unsigned int err)
```

Gets a c-style string from from a OpenGL enum.

Note

Returns "UNKNOWN_ERROR" if the error is not known.

Mainly used in the GL_CALL macro in the backend of the Easy OpenGL abstraction.

Parameters

<code>err</code>	OpenGL error enum to get the string of
------------------	--

Returns

c-style NULL terminated string

8.1.3.3 initGLFW()

```
bool egl::initGLFW ()
```

Initialize GLFW.

Sets up error callbacks and calls glfwInit.

Note

`egl::initGLFW` is not thread safe and may only be called from the main thread.

Returns

true if GLFW has been successfully initialized.

8.1.3.4 operator&() [1/2]

```
BufferFlag egl::operator& (
    BufferFlag a,
    BufferFlag b) [inline]
```

8.1.3.5 operator&() [2/2]

```
MapUsage egl::operator& (
    MapUsage a,
    MapUsage b) [inline]
```

8.1.3.6 operator" |() [1/2]

```
BufferFlag egl::operator| (
    BufferFlag a,
    BufferFlag b) [inline]
```

8.1.3.7 operator" | () [2/2]

```
MapUsage egl::operator| (
    MapUsage a,
    MapUsage b) [inline]
```

8.1.3.8 operator" |=() [1/2]

```
BufferFlag & egl::operator|= (
    BufferFlag & a,
    BufferFlag b) [inline]
```

8.1.3.9 operator" |=() [2/2]

```
MapUsage & egl::operator|= (
    MapUsage & a,
    MapUsage b) [inline]
```

8.1.3.10 pollEvents()

```
void egl::pollEvents ()
```

Polls GLFW events.

Note

`egl::pollEvents` is not thread safe and may only be called from the main thread.

8.1.3.11 shaderTypeToString()

```
std::string egl::shaderTypeToString (
    ShaderType type) [constexpr]
```

Converts the `ShaderType` enum into a `std::string`.

Parameters

<code>type</code>	The <code>ShaderType</code> to convert to a <code>std::string</code> .
-------------------	--

Returns

The name of the given `ShaderType` (if unknown "INVALID" is returned).

8.1.3.12 terminateGLFW()

```
void egl::terminateGLFW ()
```

Terminates GLFW.

Note

`egl::terminateGLFW` is not thread safe and may only be called from the main thread.

8.1.3.13 toGLenum() [1/5]

```
unsigned int egl::toGLenum (
    BufferFlag flag)
```

Converts a `BufferFlag` enum into a GLenum.

8.1.3.14 toGLenum() [2/5]

```
unsigned int egl::toGLenum (
    BufferType type)
```

Converts a `BufferType` enum into a GLenum.

Exceptions

<code>std::invalid_argument</code>	If the <code>BufferType</code> is invalid.
------------------------------------	--

8.1.3.15 toGLenum() [3/5]

```
unsigned int egl::toGLenum (
    BufferUsage usage)
```

Converts a `BufferUsage` enum into a GLenum.

Exceptions

<code>std::invalid_argument</code>	If the <code>BufferUsage</code> is invalid.
------------------------------------	---

8.1.3.16 toGLenum() [4/5]

```
unsigned int egl::toGLenum (
    MapUsage usage)
```

Converts a `MapUsage` enum into a GLenum.

8.1.3.17 toGLenum() [5/5]

```
unsigned int egl::toGLenum (
    ShaderType type)
```

Converts the `ShaderType` enum into an OpenGL enum.

Exceptions

<code>std::logic_error</code>	If the <code>ShaderType</code> is invalid.
-------------------------------	--

Parameters

<code>type</code>	The <code>ShaderType</code> to convert to a <code>std::string</code> .
-------------------	--

Returns

The resulting OpenGL enum.

8.1.3.18 validateBufferFlag()

```
bool egl::validateBufferFlag (
    BufferFlag flag,
    std::string & error)
```

Checks if the given `BufferFlag` flag combination is valid.

Parameters

<code>flag</code>	<code>BufferFlag</code> flag combination to check
<code>error</code>	Error string output if the combination is invalid

Returns

true if the `BufferFlag` combination is valid

8.1.3.19 validateMapUsage()

```
bool egl::validateMapUsage (
    MapUsage usage,
    std::string & error)
```

Checks if the given `MapUsage` flag combination is valid.

Parameters

<i>usage</i>	MapUsage flag combination to check
<i>error</i>	Error string output if the combination is invalid

Returns

true if the [MapUsage](#) combination is valid

Chapter 9

Data Structure Documentation

9.1 egl::_uniformData Struct Reference

```
#include <program.h>
```

Data Fields

- int `location`
- unsigned int `glType`
- int `arraySize`

9.1.1 Field Documentation

9.1.1.1 `arraySize`

```
int egl::_uniformData::arraySize
```

9.1.1.2 `glType`

```
unsigned int egl::_uniformData::glType
```

9.1.1.3 `location`

```
int egl::_uniformData::location
```

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

- inc/EGL/program.h

9.2 egl::Buffer Class Reference

```
#include <buffer.h>
```

Public Member Functions

- `Buffer ()=delete`
- `Buffer (BufferType type)`
Construct a new `Buffer` object of given type.
- `Buffer (Buffer &&other)`
- `Buffer (const Buffer &other)=delete`
- `~Buffer () noexcept`
- `void bind () const`
Binds the `Buffer` to the appropriate binding point.
- `int64_t size () const`
Returns the size in bytes of the `Buffer`.
- `BufferType getType () const`
Get the Type of the `Buffer`.
- `void setData (int64_t size, const void *data, BufferUsage usage)`
Set the data of the `Buffer` with a given usage.
- template<typename T>
`void setData (std::vector< T > data, BufferUsage usage)`
Set the data of the `Buffer` with a given usage.
- `void setStorage (int64_t size, const void *data, BufferFlag flags)`
Set the data of the `Buffer` with given `Buffer` flags.
- template<typename T>
`void setStorage (std::vector< T > data, BufferFlag flags)`
Set the data of the `Buffer` with given `Buffer` flags.
- `void setSubData (int64_t offset, int64_t size, const void *data)`
Set a subset of the data in the `Buffer`.
- `void getSubData (int64_t offset, int64_t size, void *data)`
Get a subset of the data in the `Buffer`.
- `void * map (int64_t offset, int64_t length, MapUsage access)`
Map a part of the `Buffer` data to the client's address space.
- `void unmap ()`
Unmaps the `Buffer`.
- `Buffer & operator= (Buffer &&other)`
- `Buffer & operator= (const Buffer &other)=delete`

Protected Member Functions

- `void _delete ()`
- `void _check ()`

Protected Attributes

- `unsigned int _id = 0`
- `bool _mapped = false`
- `MapUsage _mapUsage = MapUsage::None`
- `BufferFlag _flags = BufferFlag::None`
- `BufferType _type`

9.2.1 Constructor & Destructor Documentation

9.2.1.1 Buffer() [1/4]

```
egl::Buffer::Buffer () [delete]
```

9.2.1.2 Buffer() [2/4]

```
egl::Buffer::Buffer (
    BufferType type)
```

Construct a new [Buffer](#) object of given type.

9.2.1.3 Buffer() [3/4]

```
egl::Buffer::Buffer (
    Buffer && other)
```

9.2.1.4 Buffer() [4/4]

```
egl::Buffer::Buffer (
    const Buffer & other) [delete]
```

9.2.1.5 ~Buffer()

```
egl::Buffer::~Buffer () [noexcept]
```

9.2.2 Member Function Documentation

9.2.2.1 _check()

```
void egl::Buffer::_check () [protected]
```

9.2.2.2 _delete()

```
void egl::Buffer::_delete () [protected]
```

9.2.2.3 bind()

```
void egl::Buffer::bind () const
```

Binds the [Buffer](#) to the appropriate binding point.

9.2.2.4 getSubData()

```
void egl::Buffer::getSubData (
    int64_t offset,
    int64_t size,
    void * data)
```

Get a subset of the data in the [Buffer](#).

Exceptions

<code>std::runtime_error</code>	If offset is negative
<code>std::runtime_error</code>	If size is negative
<code>std::runtime_error</code>	If offset + size is greater than the size of the Buffer
<code>std::runtime_error</code>	If the Buffer is mapped and MapUsage::Persistent is not set

Parameters

<code>offset</code>	The offset of the start of the subset to get in bytes
<code>size</code>	The size of the subset to get in bytes
<code>data</code>	The destination of the data of the subset (must be at least size bytes big)

9.2.2.5 getType()

```
BufferType egl::Buffer::getType () const
```

Get the Type of the [Buffer](#).

9.2.2.6 map()

```
void * egl::Buffer::map (
    int64_t offset,
    int64_t length,
    MapUsage access)
```

Map a part of the [Buffer](#) data to the client's address space.

Exceptions

<code>std::runtime_error</code>	If the Buffer was already mapped
<code>std::runtime_error</code>	If length is less than or equal to 0
<code>std::runtime_error</code>	If offset is less than 0
<code>std::runtime_error</code>	If length + offset is greater than the size of the Buffer
<code>std::runtime_error</code>	If the MapUsage is invalid
<code>std::runtime_error</code>	If MapUsage::Persistent was requested without BufferFlag::MapPersistent being set through setStorage
<code>std::runtime_error</code>	If mapping failed and a nullptr was returned

Parameters

<i>offset</i>	The offset of the map range into the <code>Buffer</code> in bytes
<i>length</i>	The length of the map range in the <code>Buffer</code> in bytes
<i>access</i>	The usage of the mapped range

Returns

A pointer to the beginning of the mapped range

9.2.2.7 operator=() [1/2]

```
Buffer & egl::Buffer::operator= (
    Buffer && other)
```

9.2.2.8 operator=() [2/2]

```
Buffer & egl::Buffer::operator= (
    const Buffer & other) [delete]
```

9.2.2.9 setData() [1/2]

```
void egl::Buffer::setData (
    int64_t size,
    const void * data,
    BufferUsage usage)
```

Set the data of the `Buffer` with a given usage.

Exceptions

<code>std::runtime_error</code>	If size is negative
---------------------------------	---------------------

Parameters

<i>size</i>	The size of the data in bytes
<i>data</i>	The data to store in the <code>Buffer</code> (must have at least size bytes of data)
<i>usage</i>	The usage hint of the <code>Buffer</code>

9.2.2.10 setData() [2/2]

```
template<typename T>
void egl::Buffer::setData (
    std::vector< T > data,
    BufferUsage usage) [inline]
```

Set the data of the [Buffer](#) with a given usage.

Parameters

<i>data</i>	The data to store in the Buffer
<i>usage</i>	The usage hint of the Buffer

9.2.2.11 setStorage() [1/2]

```
void egl::Buffer::setStorage (
    int64_t size,
    const void * data,
    BufferFlag flags)
```

Set the data of the [Buffer](#) with given [Buffer](#) flags.

Exceptions

<i>std::runtime_error</i>	If size is not greater than 0
<i>std::runtime_error</i>	If the BufferFlag combination is invalid

Parameters

<i>size</i>	The size of the data in bytes
<i>data</i>	The data to store in the Buffer (must have at least size bytes of data)
<i>flags</i>	The Buffer usage flags

9.2.2.12 setStorage() [2/2]

```
template<typename T>
void egl::Buffer::setStorage (
    std::vector< T > data,
    BufferFlag flags) [inline]
```

Set the data of the [Buffer](#) with given [Buffer](#) flags.

Exceptions

<i>std::runtime_error</i>	If size is not greater than 0
<i>std::runtime_error</i>	If the BufferFlag combination is invalid

Parameters

<i>data</i>	The data to store in the Buffer
<i>flags</i>	The Buffer usage flags

9.2.2.13 setSubData()

```
void egl::Buffer::setSubData (
    int64_t offset,
    int64_t size,
    const void * data)
```

Set a subset of the data in the [Buffer](#).

Exceptions

<i>std::runtime_error</i>	If offset is negative
<i>std::runtime_error</i>	If size is negative
<i>std::runtime_error</i>	If offset + size is greater than the size of the Buffer
<i>std::runtime_error</i>	If the Buffer is mapped and MapUsage::Persistent is not set

Parameters

<i>offset</i>	The offset of the start of the subset to set in bytes
<i>size</i>	The size of the subset to set in bytes
<i>data</i>	The data of the subset (must have at least size bytes of data)

9.2.2.14 size()

```
int64_t egl::Buffer::size () const
```

Returns the size in bytes of the [Buffer](#).

9.2.2.15 unmap()

```
void egl::Buffer::unmap ()
```

Unmaps the [Buffer](#).

Exceptions

<i>std::runtime_error</i>	If OpenGL signalled data corruption
---------------------------	-------------------------------------

9.2.3 Field Documentation

9.2.3.1 `_flags`

```
BufferFlag egl::Buffer::_flags = BufferFlag::None [protected]
```

9.2.3.2 `_id`

```
unsigned int egl::Buffer::_id = 0 [protected]
```

9.2.3.3 `_mapped`

```
bool egl::Buffer::_mapped = false [protected]
```

9.2.3.4 `_mapUsage`

```
MapUsage egl::Buffer::_mapUsage = MapUsage::None [protected]
```

9.2.3.5 `_type`

```
BufferType egl::Buffer::_type [protected]
```

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

- inc/EGL/buffer.h

9.3 egl::Program Class Reference

Program class to abstract OpenGL Shader Programs.

```
#include <program.h>
```

Data Structures

- class UniformProxy
- class UniformProxyConst

Public Member Functions

- **Program ()**
Construct an empty Program object.
- **Program (Program &&other)**
- **Program (const Program &)=delete**
- **~Program ()**
- **void reset ()**
Resets the Program to an empty State.
- **bool attached (const Shader &shader)**
Checks if the given Shader is attached to the Program.
- **void attach (const Shader &shader)**
Attaches Shader to Program for linking.
- **void detach (const Shader &shader)**
Detaches Shader from Program for linking.
- **bool linked () const**
Gets if the Program has been successfully linked.
- **void link ()**
Links all attached Shaders and creates a valid Program.
- **void bind () const**
Binds this Program.
- **int getUniformLocation (const std::string &name) const**
Gets the Location of the given Uniform.
- **void setUniform (int location, float data)**
Sets a uniform at the given location.
- **void setUniform (int location, const glm::vec2 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::vec3 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::vec4 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, int data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::ivec2 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::ivec3 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::ivec4 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, unsigned int data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::uvec2 &data)**
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- **void setUniform (int location, const glm::uvec3 &data)**

- This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- void **setUniform** (int location, const glm::uvec4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat2x3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat3x2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat2x4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat4x2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat3x4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (int location, const glm::mat4x3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, float data)
Sets a uniform at the given name.
 - void **setUniform** (const std::string &name, const glm::vec2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::vec3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::vec4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, int data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::ivec2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::ivec3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::ivec4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- void **setUniform** (const std::string &name, unsigned int data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::uvec2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::uvec3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::uvec4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat2x3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat3x2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat2x4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat4x2 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat3x4 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **setUniform** (const std::string &name, const glm::mat4x3 &data)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **getUniform** (int location, float &data) const
Gets a uniform at the given location.
 - void **getUniform** (int location, glm::vec2 &data) const
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **getUniform** (int location, glm::vec3 &data) const
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **getUniform** (int location, glm::vec4 &data) const
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void **getUniform** (int location, int &data) const

- This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- void `getUniform` (int location, glm::ivec2 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::ivec3 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::ivec4 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, unsigned int &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::uvec2 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::uvec3 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::uvec4 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat2 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat3 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat4 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat2x3 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat3x2 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat2x4 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat4x2 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat3x4 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (int location, glm::mat4x3 &data) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
 - void `getUniform` (const std::string &name, float &data) const

Gets a uniform at the given name.
 - void `getUniform` (const std::string &name, glm::vec2 &data) const

- This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void [getUniform](#) (const std::string &name, glm::mat4x3 &data) const
 - This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- [Program & operator=](#) ([Program](#) &&other)
- [Program & operator=](#) (const [Program](#) &) = delete
- [UniformProxy operator\[\]](#) (int location)
 - Returns a Proxy to get / set the uniform at the given location with assignment and type conversion operators.
- [UniformProxy operator\[\]](#) (const std::string &name)
 - Returns a Proxy to get / set the uniform with the given name with assignment and type conversion operators.
- [UniformProxyConst operator\[\]](#) (int location) const
 - Returns a constant only Proxy to get the uniform at the given location with type conversion operators.
- [UniformProxyConst operator\[\]](#) (const std::string &name) const
 - Returns a Proxy to get the uniform with the given name with type conversion operators.

Static Public Member Functions

- static void [unbind](#) ()
 - Unbinds a [Program](#) by binding id 0.*

Protected Member Functions

- void [_delete](#) ()
- void [_check](#) () const
- void [_ensure](#) () const
- void [_queryUniformData](#) ()
- void [_setupUniform](#) (int loc, int sizeCheck, int typeCheck) const
- std::string [_getError](#) ()

Protected Attributes

- unsigned int [_id](#) = 0
- bool [_linked](#) = false
- std::unordered_map< std::string, int > [_uniformIndexMap](#) = {}
- std::unordered_map< int, int > [_uniformLocationIndexMap](#) = {}
- std::vector< [UniformData](#) > [_uniformData](#) = {}

9.3.1 Detailed Description

[Program](#) class to abstract OpenGL [Shader](#) Programs.

Warning

[Program](#) must be deconstructed before the OpenGL context is destroyed.

This class is not guaranteed to be thread-safe.

Note

This class owns the underlying OpenGL [Program](#) object and releases it upon destruction or [reset\(\)](#).

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Program() [1/3]

```
egl::Program::Program ()
```

Construct an empty [Program](#) object.

9.3.2.2 Program() [2/3]

```
egl::Program::Program (
    Program && other)
```

9.3.2.3 Program() [3/3]

```
egl::Program::Program (
    const Program & ) [delete]
```

9.3.2.4 ~Program()

```
egl::Program::~Program ()
```

9.3.3 Member Function Documentation

9.3.3.1 _check()

```
void egl::Program::_check () const [protected]
```

9.3.3.2 _delete()

```
void egl::Program::_delete () [protected]
```

9.3.3.3 _ensure()

```
void egl::Program::_ensure () const [protected]
```

9.3.3.4 _getError()

```
std::string egl::Program::_getError () [protected]
```

9.3.3.5 _queryUniformData()

```
void egl::Program::_queryUniformData () [protected]
```

9.3.3.6 `_setupUniform()`

```
void egl::Program::_setupUniform (
    int loc,
    int sizeCheck,
    int typeCheck) const [protected]
```

9.3.3.7 `attach()`

```
void egl::Program::attach (
    const Shader & shader)
```

Attaches [Shader](#) to [Program](#) for linking.

Note

Attaching a [Shader](#) invalidates the current link state.

Warning

Attaching a destroyed [Shader](#) is undefined behavior.

Attached Shaders must outlive the [Program](#) (at least until linking).

Exceptions

<code>std::runtime_error</code>	If the Shader has already been attached.
<code>std::logic_error</code>	If the current Program object does not exist (<code>reset()</code> is recommended to return to a valid state)

Parameters

<code>shader</code>	The Shader that should be attached to the Program .
---------------------	---

9.3.3.8 `attached()`

```
bool egl::Program::attached (
    const Shader & shader)
```

Checks if the given [Shader](#) is attached to the [Program](#).

Parameters

<code>shader</code>	The Shader to check.
---------------------	--------------------------------------

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
-------------------------	---

Returns

true if the [Shader](#) is attached, false otherwise.

9.3.3.9 bind()

```
void egl::Program::bind () const
```

Binds this [Program](#).

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<i>std::runtime_error</i>	If the Program was not linked to avoid invalid usage.

9.3.3.10 detach()

```
void egl::Program::detach (
    const Shader & shader)
```

Detaches [Shader](#) from [Program](#) for linking.

Note

Detaching a [Shader](#) invalidates the current link state.

Exceptions

<i>std::runtime_error</i>	If the Shader was not attached.
<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)

Parameters

<i>shader</i>	The Shader that should be detached from the Program .
---------------	---

9.3.3.11 getUniform() [1/42]

```
void egl::Program::getUniform (
    const std::string & name,
    float & data) const [inline]
```

Gets a uniform at the given name.

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<i>std::invalid_argument</i>	If the location does not correspond to a uniform
<i>std::runtime_error</i>	If the type does not correspond to the GLSL type
<i>std::invalid_argument</i>	If it attempts to get a GLSL array
<i>std::runtime_error</i>	If the current Program is unlinked

Parameters

<i>name</i>	The uniform name
<i>data</i>	The variable to store the data in

This overload works for `float`, `int`, `glm::vec*`, `glm::mat*`, etc.

9.3.3.12 getUniform() [2/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::ivec2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.13 getUniform() [3/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::ivec3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.14 getUniform() [4/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::ivec4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.15 getUniform() [5/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.16 getUniform() [6/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat2x3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.17 getUniform() [7/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat2x4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.18 getUniform() [8/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.19 getUniform() [9/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat3x2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.20 getUniform() [10/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat3x4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.21 getUniform() [11/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.22 getUniform() [12/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat4x2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.23 getUniform() [13/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::mat4x3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.24 getUniform() [14/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::uvec2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.25 getUniform() [15/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::uvec3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.26 getUniform() [16/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::uvec4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.27 getUniform() [17/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::vec2 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.28 getUniform() [18/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::vec3 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.29 getUniform() [19/42]

```
void egl::Program::getUniform (
    const std::string & name,
    glm::vec4 & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.30 getUniform() [20/42]

```
void egl::Program::getUniform (
    const std::string & name,
    int & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.31 getUniform() [21/42]

```
void egl::Program::getUniform (
    const std::string & name,
    unsigned int & data) const [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.32 getUniform() [22/42]

```
void egl::Program::getUniform (
    int location,
    float & data) const
```

Gets a uniform at the given location.

Exceptions

<code>std::logic_error</code>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<code>std::invalid_argument</code>	If the location does not correspond to a uniform
<code>std::runtime_error</code>	If the type does not correspond to the GLSL type
<code>std::invalid_argument</code>	If it attempts to get a GLSL array
<code>std::runtime_error</code>	If the current Program is unlinked

Parameters

<code>location</code>	The uniform location returned by getUniformLocation()
<code>data</code>	The variable to store the data in

This overload works for `float`, `int`, `glm::vec*`, `glm::mat*`, etc.

9.3.3.33 getUniform() [23/42]

```
void egl::Program::getUniform (
    int location,
    glm::ivec2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.34 getUniform() [24/42]

```
void egl::Program::getUniform (
    int location,
    glm::ivec3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.35 getUniform() [25/42]

```
void egl::Program::getUniform (
    int location,
    glm::ivec4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.36 getUniform() [26/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.37 getUniform() [27/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat2x3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.38 getUniform() [28/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat2x4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.39 getUniform() [29/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.40 getUniform() [30/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat3x2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.41 getUniform() [31/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat3x4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.42 getUniform() [32/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.43 getUniform() [33/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat4x2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.44 getUniform() [34/42]

```
void egl::Program::getUniform (
    int location,
    glm::mat4x3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.45 getUniform() [35/42]

```
void egl::Program::getUniform (
    int location,
    glm::uvec2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.46 getUniform() [36/42]

```
void egl::Program::getUniform (
    int location,
    glm::uvec3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.47 getUniform() [37/42]

```
void egl::Program::getUniform (
    int location,
    glm::uvec4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.48 getUniform() [38/42]

```
void egl::Program::getUniform (
    int location,
    glm::vec2 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.49 getUniform() [39/42]

```
void egl::Program::getUniform (
    int location,
    glm::vec3 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.50 getUniform() [40/42]

```
void egl::Program::getUniform (
    int location,
    glm::vec4 & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.51 getUniform() [41/42]

```
void egl::Program::getUniform (
    int location,
    int & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.52 getUniform() [42/42]

```
void egl::Program::getUniform (
    int location,
    unsigned int & data) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.53 getUniformLocation()

```
int egl::Program::getUniformLocation (
    const std::string & name) const
```

Gets the Location of the given Uniform.

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<i>std::runtime_error</i>	If the current Program was not linked before use
<i>std::invalid_argument</i>	If the uniform does not exist

Parameters

<i>name</i>	The name of the uniform to find
-------------	---------------------------------

Returns

The location of the uniform

9.3.3.54 link()

```
void egl::Program::link ()
```

Links all attached Shaders and creates a valid [Program](#).

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<i>egl::ProgramLinkError</i>	If the Program fails to link. The Program remains in a valid state afterwards.
<i>egl::ProgramValidateError</i>	If the Program fails to validate. This only occurs when DEBUG_BUILD is defined else the validity is not checked.

9.3.3.55 linked()

```
bool egl::Program::linked () const [inline]
```

Gets if the [Program](#) has been successfully linked.

Returns

true if the [Program](#) is linked, false otherwise.

9.3.3.56 operator=() [1/2]

```
Program & egl::Program::operator= (
    const Program & )  [delete]
```

9.3.3.57 operator=() [2/2]

```
Program & egl::Program::operator= (
    Program && other)
```

9.3.3.58 operator[]() [1/4]

```
UniformProxy egl::Program::operator[ ] (
    const std::string & name)  [inline]
```

Returns a Proxy to get / set the uniform with the given name with assignment and type conversion operators.

Note

Possible errors when assigning can be found under [egl::Program::setUniform](#).

Possible errors on explicit type conversion can be found under [egl::Program::getUniform](#).

Warning

Setting a uniform binds the program.

Exceptions

<code>std::logic_error</code>	If the current <code>Program</code> object does not exist (<code>reset()</code> is recommended to return to a valid state)
<code>std::runtime_error</code>	If the current <code>Program</code> was not linked before use
<code>std::invalid_argument</code>	If the uniform does not exist

Parameters

<code>name</code>	The uniform name
-------------------	------------------

9.3.3.59 operator[]() [2/4]

```
UniformProxyConst egl::Program::operator[ ] (
    const std::string & name) const  [inline]
```

Returns a Proxy to get the uniform with the given name with type conversion operators.

Note

Possible errors on explicit type conversion can be found under [egl::Program::getUniform](#).

Exceptions

<i>std::logic_error</i>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<i>std::runtime_error</i>	If the current Program was not linked before use
<i>std::invalid_argument</i>	If the uniform does not exist

Parameters

<i>name</i>	The uniform name
-------------	------------------

9.3.3.60 operator[]() [3/4]

```
UniformProxy egl::Program::operator[] (
    int location) [inline]
```

Returns a Proxy to get / set the uniform at the given location with assignment and type conversion operators.

Note

Possible errors when assigning can be found under [egl::Program::setUniform](#).

Possible errors on explicit type conversion can be found under [egl::Program::getUniform](#).

Warning

Setting a uniform binds the program.

Parameters

<i>location</i>	The uniform location returned by getUniformLocation()
-----------------	---

9.3.3.61 operator[]() [4/4]

```
UniformProxyConst egl::Program::operator[] (
    int location) const [inline]
```

Returns a constant only Proxy to get the uniform at the given location with type conversion operators.

Note

Possible errors on explicit type conversion can be found under [egl::Program::getUniform](#).

Parameters

<i>location</i>	The uniform location returned by getUniformLocation()
-----------------	---

9.3.3.62 reset()

```
void egl::Program::reset ()
```

Resets the [Program](#) to an empty State.

Exceptions

<i>std::runtime_error</i>	If OpenGL failed to create a new Program .
---------------------------	--

9.3.3.63 setUniform() [1/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::ivec2 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.64 setUniform() [2/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::ivec3 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.65 setUniform() [3/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::ivec4 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.66 setUniform() [4/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat2 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.67 setUniform() [5/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat2x3 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.68 setUniform() [6/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat2x4 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.69 setUniform() [7/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat3 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.70 setUniform() [8/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat3x2 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.71 setUniform() [9/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat3x4 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.72 setUniform() [10/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat4 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.73 setUniform() [11/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat4x2 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.74 setUniform() [12/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::mat4x3 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.75 setUniform() [13/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::uvec2 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.76 setUniform() [14/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::uvec3 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.77 setUniform() [15/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::uvec4 & data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.78 `setUniform()` [16/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::vec2 & data)  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.79 `setUniform()` [17/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::vec3 & data)  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.80 `setUniform()` [18/42]

```
void egl::Program::setUniform (
    const std::string & name,
    const glm::vec4 & data)  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.81 `setUniform()` [19/42]

```
void egl::Program::setUniform (
    const std::string & name,
    float data)  [inline]
```

Sets a uniform at the given name.

Exceptions

<code>std::logic_error</code>	If the current Program object does not exist (reset() is recommended to return to a valid state)
<code>std::invalid_argument</code>	If the location does not correspond to a uniform
<code>std::runtime_error</code>	If the type does not correspond to the GLSL type
<code>std::invalid_argument</code>	If it attempts to set a GLSL array
<code>std::runtime_error</code>	If the current Program is unlinked

Warning

Setting a uniform binds the program.

Parameters

<i>name</i>	The uniform name
<i>data</i>	The value to assign to the uniform

This overload works for float, int, glm::vec*, glm::mat*, etc.

9.3.3.82 setUniform() [20/42]

```
void egl::Program::setUniform (
    const std::string & name,
    int data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.83 setUniform() [21/42]

```
void egl::Program::setUniform (
    const std::string & name,
    unsigned int data) [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.84 setUniform() [22/42]

```
void egl::Program::setUniform (
    int location,
    const glm::ivec2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.85 setUniform() [23/42]

```
void egl::Program::setUniform (
    int location,
    const glm::ivec3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.86 setUniform() [24/42]

```
void egl::Program::setUniform (
    int location,
    const glm::ivec4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.87 setUniform() [25/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.88 setUniform() [26/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat2x3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.89 setUniform() [27/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat2x4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.90 setUniform() [28/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.91 setUniform() [29/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat3x2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.92 setUniform() [30/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat3x4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.93 setUniform() [31/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.94 setUniform() [32/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat4x2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.95 setUniform() [33/42]

```
void egl::Program::setUniform (
    int location,
    const glm::mat4x3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.96 setUniform() [34/42]

```
void egl::Program::setUniform (
    int location,
    const glm::uvec2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.97 setUniform() [35/42]

```
void egl::Program::setUniform (
    int location,
    const glm::uvec3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.98 `setUniform()` [36/42]

```
void egl::Program::setUniform (
    int location,
    const glm::uvec4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.99 `setUniform()` [37/42]

```
void egl::Program::setUniform (
    int location,
    const glm::vec2 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.100 `setUniform()` [38/42]

```
void egl::Program::setUniform (
    int location,
    const glm::vec3 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.101 `setUniform()` [39/42]

```
void egl::Program::setUniform (
    int location,
    const glm::vec4 & data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.102 `setUniform()` [40/42]

```
void egl::Program::setUniform (
    int location,
    float data)
```

Sets a uniform at the given location.

Exceptions

<code>std::logic_error</code>	If the current Program object does not exist (reset() is recommended to return to a valid state)
-------------------------------	---

<code>std::invalid_argument</code>	If the location does not correspond to a uniform
<code>std::runtime_error</code>	If the type does not correspond to the GLSL type
<code>std::invalid_argument</code>	If it attempts to set a GLSL array
<code>std::runtime_error</code>	If the current Program is unlinked

Warning

Setting a uniform binds the program.

Parameters

<code>location</code>	The uniform location returned by getUniformLocation()
<code>data</code>	The value to assign to the uniform

This overload works for `float`, `int`, `glm::vec*`, `glm::mat*`, etc.

9.3.3.103 `setUniform()` [41/42]

```
void egl::Program::setUniform (
    int location,
    int data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.104 `setUniform()` [42/42]

```
void egl::Program::setUniform (
    int location,
    unsigned int data)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.3.3.105 `unbind()`

```
void egl::Program::unbind () [static]
```

Unbinds a [Program](#) by binding id 0.

9.3.4 Field Documentation**9.3.4.1 `_id`**

```
unsigned int egl::Program::_id = 0 [protected]
```

9.3.4.2 `_linked`

```
bool egl::Program::_linked = false [protected]
```

9.3.4.3 `_uniformData`

```
std::vector<UniformData> egl::Program::_uniformData = {} [protected]
```

9.3.4.4 `_uniformIndexMap`

```
std::unordered_map<std::string, int> egl::Program::_uniformIndexMap = {} [protected]
```

9.3.4.5 `_uniformLocationIndexMap`

```
std::unordered_map<int, int> egl::Program::_uniformLocationIndexMap = {} [protected]
```

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

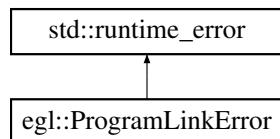
- inc/EGL/program.h

9.4 egl::ProgramLinkError Class Reference

Exception thrown when [Program](#) linking fails.

```
#include <program.h>
```

Inheritance diagram for egl::ProgramLinkError:



Public Member Functions

- [ProgramLinkError](#) (const std::string &infoLog)
Construct a new Program Link Error object.

9.4.1 Detailed Description

Exception thrown when [Program](#) linking fails.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 ProgramLinkError()

```
egl::ProgramLinkError::ProgramLinkError (
    const std::string & infoLog) [inline]
```

Construct a new [Program](#) Link Error object.

Parameters

<i>infoLog</i>	InfoLog buffer from glGetProgramInfoLog.
----------------	--

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

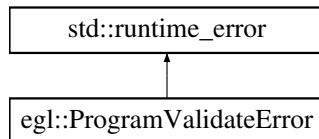
- inc/EGL/program.h

9.5 egl::ProgramValidateError Class Reference

Exception thrown when [Program](#) validation fails, mainly used in debug builds.

```
#include <program.h>
```

Inheritance diagram for egl::ProgramValidateError:



Public Member Functions

- [ProgramValidateError](#) (const std::string &infoLog)

Construct a new Program Validate Error object.

9.5.1 Detailed Description

Exception thrown when [Program](#) validation fails, mainly used in debug builds.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 ProgramValidateError()

```
egl::ProgramValidateError::ProgramValidateError (
    const std::string & infoLog) [inline]
```

Construct a new [Program](#) Validate Error object.

Parameters

<i>infoLog</i>	InfoLog buffer from glGetProgramInfoLog.
----------------	--

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

- inc/EGL/program.h

9.6 egl::Shader Class Reference

[Shader](#) class to abstract OpenGL Shaders.

```
#include <shader.h>
```

Public Member Functions

- [Shader \(\)=delete](#)
 - Constructs a new [Shader](#) object of given type.
- [Shader \(ShaderType type\)](#)
 - Constructs and compiles a new [Shader](#) object of given type.
- [Shader \(ShaderType type, const char *src\)](#)
 - Constructs and compiles a new [Shader](#) object of given type.
- [Shader \(ShaderType type, const std::string &src\)](#)
 - Constructs and compiles a new [Shader](#) object of given type.
- [Shader \(ShaderType type, std::istream &in\)](#)
 - Constructs and compiles a new [Shader](#) object of given type.
- [Shader \(ShaderType type, std::istream &&in\)](#)
 - Constructs and compiles a new [Shader](#) object of given type.
- [Shader \(Shader &&other\)](#)
 - Shaders are moveable.
- [Shader \(const Shader &other\)=delete](#)
 - Shaders are non-copyable.
- [~Shader \(\) noexcept](#)
 - Destructor.
- [void reset \(\)](#)
 - Resets the [Shader](#) to an empty state.
- [bool compiled \(\) const](#)
 - Gets if the [Shader](#) has been successfully compiled.
- [ShaderType getType \(\) const](#)
 - Get the type of the [Shader](#).
- [void compile \(const char *src\)](#)
 - Compiles the [Shader](#) with the given source.
- [void compile \(std::istream &in\)](#)
 - Compiles the [Shader](#) with the given source input stream.
- [void compile \(std::istream &&in\)](#)
 - Compiles the [Shader](#) with the given source input stream.
- [void compile \(const std::string &str\)](#)
 - Compiles the [Shader](#) with the given source.
- [Shader & operator= \(Shader &&other\)](#)
 - Assignment operator.
- [Shader & operator= \(const Shader &other\)=delete](#)
 - Assignment operator.

Data Fields

- friend [Program](#)

Protected Member Functions

- void [_delete \(\)](#)
- void [_check \(\)](#)
- void [_ensure \(\)](#)
- std::string [_getError \(\)](#)

Protected Attributes

- unsigned int [_id](#) = 0
- [ShaderType](#) [_type](#)
- bool [_compiled](#) = false

9.6.1 Detailed Description

[Shader](#) class to abstract OpenGL Shaders.

Warning

[Shader](#) must be deconstructed before the OpenGL context is destroyed.

This class is not guaranteed to be thread-safe.

Note

This class owns the underlying OpenGL [Shader](#) object and releases it upon destruction or [reset\(\)](#).

9.6.2 Constructor & Destructor Documentation

9.6.2.1 [Shader\(\)](#) [1/8]

```
egl::Shader::Shader () [delete]
```

9.6.2.2 `Shader()` [2/8]

```
egl::Shader::Shader (
    ShaderType type)
```

Construct a new `Shader` object of given type.

Exceptions

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.

9.6.2.3 `Shader()` [3/8]

```
egl::Shader::Shader (
    ShaderType type,
    const char * src)
```

Constructs and compiles a new `Shader` object of given type.

Exceptions

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.
<code>std::invalid_argument</code>	If the <code>Shader</code> source is NULL.
<code>egl::ShaderCompileError</code>	If the <code>Shader</code> fails to compile.

9.6.2.4 `Shader()` [4/8]

```
egl::Shader::Shader (
    ShaderType type,
    const std::string & src)
```

Constructs and compiles a new `Shader` object of given type.

Exceptions

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.
<code>std::invalid_argument</code>	If the <code>Shader</code> source is NULL.
<code>egl::ShaderCompileError</code>	If the <code>Shader</code> fails to compile.

9.6.2.5 `Shader()` [5/8]

```
egl::Shader::Shader (
    ShaderType type,
    std::istream & in)
```

Constructs and compiles a new `Shader` object of given type.

Exceptions

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.
<code>std::invalid_argument</code>	If the istream is not good().
<code>std::invalid_argument</code>	If in fails to read the <code>Shader</code> source.
<code>std::invalid_argument</code>	If the given input stream is invalid.
<code>std::invalid_argument</code>	If the <code>Shader</code> source is NULL.
<code>egl::ShaderCompileError</code>	If the <code>Shader</code> fails to compile.

9.6.2.6 `Shader()` [6/8]

```
egl::Shader::Shader (
    ShaderType type,
    std::istream && in)
```

Constructs and compiles a new `Shader` object of given type.

Exceptions

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.
<code>std::invalid_argument</code>	If the istream is not good().
<code>std::invalid_argument</code>	If in fails to read the <code>Shader</code> source.
<code>std::invalid_argument</code>	If the given input stream is invalid.
<code>std::invalid_argument</code>	If the <code>Shader</code> source is NULL.
<code>egl::ShaderCompileError</code>	If the <code>Shader</code> fails to compile.

9.6.2.7 `Shader()` [7/8]

```
egl::Shader::Shader (
    Shader && other)
```

9.6.2.8 `Shader()` [8/8]

```
egl::Shader::Shader (
    const Shader & other) [delete]
```

9.6.2.9 ~Shader()

```
egl::Shader::~Shader () [noexcept]
```

9.6.3 Member Function Documentation

9.6.3.1 _check()

```
void egl::Shader::_check () [protected]
```

9.6.3.2 _delete()

```
void egl::Shader::_delete () [protected]
```

9.6.3.3 _ensure()

```
void egl::Shader::_ensure () [protected]
```

9.6.3.4 _getError()

```
std::string egl::Shader::_getError () [protected]
```

9.6.3.5 compile() [1/4]

```
void egl::Shader::compile (
    const char * src)
```

Compiles the [Shader](#) with the given source.

Exceptions

<i>std::invalid_argument</i>	If the Shader source is NULL.
<i>std::logic_error</i>	If the current Shader object does not exist (reset() is recommended to return to a valid state).
<i>egl::ShaderCompileError</i>	If the Shader fails to compile.

Parameters

<i>src</i>	Null terminated c-style string to compile.
------------	--

9.6.3.6 compile() [2/4]

```
void egl::Shader::compile (
    const std::string & str)
```

Compiles the [Shader](#) with the given source.

Exceptions

<i>std::invalid_argument</i>	If the given input stream is invalid.
<i>std::invalid_argument</i>	If the Shader source is NULL.
<i>std::logic_error</i>	If the current Shader object does not exist (reset() is recommended to return to a valid state).
<i>egl::ShaderCompileError</i>	If the Shader fails to compile.

Parameters

<i>str</i>	Source to compile.
------------	--------------------

9.6.3.7 compile() [3/4]

```
void egl::Shader::compile (
    std::istream && in)
```

Compiles the [Shader](#) with the given source input stream.

Note

Reads the entire stream and compiles it.

Exceptions

<i>std::invalid_argument</i>	If the istream is not good().
<i>std::invalid_argument</i>	If in fails to read the Shader source.
<i>std::invalid_argument</i>	If the Shader source is NULL.
<i>std::logic_error</i>	If the current Shader object does not exist (reset() is recommended to return to a valid state).
<i>egl::ShaderCompileError</i>	If the Shader fails to compile.

Parameters

<i>in</i>	Input stream to compile.
-----------	--------------------------

9.6.3.8 compile() [4/4]

```
void egl::Shader::compile (
    std::istream & in)
```

Compiles the [Shader](#) with the given source input stream.

Note

Reads the entire stream and compiles it.

Exceptions

<i>std::invalid_argument</i>	If the istream is not good().
<i>std::invalid_argument</i>	If in fails to read the Shader source.
<i>std::invalid_argument</i>	If the Shader source is NULL.
<i>std::logic_error</i>	If the current Shader object does not exist (reset() is recommended to return to a valid state).
egl::ShaderCompileError	If the Shader fails to compile.

Parameters

<i>in</i>	Input stream to compile.
-----------	--------------------------

9.6.3.9 compiled()

```
bool egl::Shader::compiled () const [inline]
```

Gets if the [Shader](#) has been successfully compiled.

9.6.3.10 getType()

```
ShaderType egl::Shader::getType () const [inline]
```

Get the type of the [Shader](#).

9.6.3.11 operator=() [1/2]

```
Shader & egl::Shader::operator= (
    const Shader & other) [delete]
```

9.6.3.12 operator=() [2/2]

```
Shader & egl::Shader::operator= (
    Shader && other)
```

9.6.3.13 reset()

```
void egl::Shader::reset ()
```

Resets the [Shader](#) to an empty state.

Note

Reset creates an entirely new [Shader](#) object of the current type and therefore can be used to get a valid State as long as OpenGL doesn't fail to create a new [Shader](#) object.

Exceptions

<code>std::runtime_error</code>	If OpenGL failed to create a new Shader object.
---------------------------------	---

9.6.4 Field Documentation

9.6.4.1 _compiled

```
bool egl::Shader::_compiled = false [protected]
```

9.6.4.2 _id

```
unsigned int egl::Shader::_id = 0 [protected]
```

9.6.4.3 _type

```
ShaderType egl::Shader::_type [protected]
```

9.6.4.4 Program

```
friend egl::Shader::Program
```

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

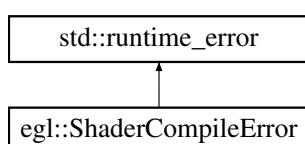
- inc/EGL/shader.h

9.7 egl::ShaderCompileError Class Reference

Exception thrown when [Shader](#) compilation fails.

```
#include <shader.h>
```

Inheritance diagram for egl::ShaderCompileError:



Public Member Functions

- [ShaderCompileError \(ShaderType ShaderType, const std::string &infoLog\)](#)
Construct a new [Shader](#) Compile Error object.

9.7.1 Detailed Description

Exception thrown when [Shader](#) compilation fails.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 [ShaderCompileError\(\)](#)

```
egl::ShaderCompileError::ShaderCompileError (
    ShaderType ShaderType,
    const std::string & infoLog) [inline]
```

Construct a new [Shader](#) Compile Error object.

Parameters

ShaderType	The type of Shader that failed to compile.
infoLog	InfoLog buffer from glGetShaderInfoLog.

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

- inc/EGL/shader.h

9.8 egl::Program::UniformProxy Class Reference

```
#include <program.h>
```

Public Member Functions

- [UniformProxy \(Program &parent, int location\)](#)
- [UniformProxy \(UniformProxy &&other\)=delete](#)
- [UniformProxy \(const UniformProxy &other\)=delete](#)
- [operator float \(\) const](#)
- [operator glm::vec2 \(\) const](#)
- [operator glm::vec3 \(\) const](#)
- [operator glm::vec4 \(\) const](#)
- [operator int \(\) const](#)
- [operator glm::ivec2 \(\) const](#)
- [operator glm::ivec3 \(\) const](#)
- [operator glm::ivec4 \(\) const](#)
- [operator unsigned int \(\) const](#)
- [operator glm::uvec2 \(\) const](#)

- `operator glm::uvec3 () const`
- `operator glm::uvec4 () const`
- `operator glm::mat2 () const`
- `operator glm::mat3 () const`
- `operator glm::mat4 () const`
- `operator glm::mat2x3 () const`
- `operator glm::mat3x2 () const`
- `operator glm::mat2x4 () const`
- `operator glm::mat4x2 () const`
- `operator glm::mat3x4 () const`
- `operator glm::mat4x3 () const`
- `UniformProxy & operator= (float other)`
- `UniformProxy & operator= (const glm::vec2 &other)`
- `UniformProxy & operator= (const glm::vec3 &other)`
- `UniformProxy & operator= (const glm::vec4 &other)`
- `UniformProxy & operator= (int other)`
- `UniformProxy & operator= (const glm::ivec2 &other)`
- `UniformProxy & operator= (const glm::ivec3 &other)`
- `UniformProxy & operator= (const glm::ivec4 &other)`
- `UniformProxy & operator= (unsigned int other)`
- `UniformProxy & operator= (const glm::uvec2 &other)`
- `UniformProxy & operator= (const glm::uvec3 &other)`
- `UniformProxy & operator= (const glm::uvec4 &other)`
- `UniformProxy & operator= (const glm::mat2 &other)`
- `UniformProxy & operator= (const glm::mat3 &other)`
- `UniformProxy & operator= (const glm::mat4 &other)`
- `UniformProxy & operator= (const glm::mat2x3 &other)`
- `UniformProxy & operator= (const glm::mat3x2 &other)`
- `UniformProxy & operator= (const glm::mat2x4 &other)`
- `UniformProxy & operator= (const glm::mat4x2 &other)`
- `UniformProxy & operator= (const glm::mat3x4 &other)`
- `UniformProxy & operator= (const glm::mat4x3 &other)`
- `UniformProxy & operator= (const UniformProxy &other)=delete`
- `UniformProxy & operator= (UniformProxy &&other)=delete`

9.8.1 Constructor & Destructor Documentation

9.8.1.1 UniformProxy() [1/3]

```
egl::Program::UniformProxy::UniformProxy (
    Program & parent,
    int location) [inline]
```

9.8.1.2 UniformProxy() [2/3]

```
egl::Program::UniformProxy::UniformProxy (
    UniformProxy && other) [delete]
```

9.8.1.3 UniformProxy() [3/3]

```
egl::Program::UniformProxy::UniformProxy (
    const UniformProxy & other) [delete]
```

9.8.2 Member Function Documentation

9.8.2.1 operator float()

```
egl::Program::UniformProxy::operator float () const [inline], [explicit]
```

9.8.2.2 operator glm::ivec2()

```
egl::Program::UniformProxy::operator glm::ivec2 () const [inline], [explicit]
```

9.8.2.3 operator glm::ivec3()

```
egl::Program::UniformProxy::operator glm::ivec3 () const [inline], [explicit]
```

9.8.2.4 operator glm::ivec4()

```
egl::Program::UniformProxy::operator glm::ivec4 () const [inline], [explicit]
```

9.8.2.5 operator glm::mat2()

```
egl::Program::UniformProxy::operator glm::mat2 () const [inline], [explicit]
```

9.8.2.6 operator glm::mat2x3()

```
egl::Program::UniformProxy::operator glm::mat2x3 () const [inline], [explicit]
```

9.8.2.7 operator glm::mat2x4()

```
egl::Program::UniformProxy::operator glm::mat2x4 () const [inline], [explicit]
```

9.8.2.8 operator glm::mat3()

```
egl::Program::UniformProxy::operator glm::mat3 () const [inline], [explicit]
```

9.8.2.9 operator glm::mat3x2()

```
egl::Program::UniformProxy::operator glm::mat3x2 () const [inline], [explicit]
```

9.8.2.10 operator glm::mat3x4()

```
egl::Program::UniformProxy::operator glm::mat3x4 () const [inline], [explicit]
```

9.8.2.11 operator glm::mat4()

```
egl::Program::UniformProxy::operator glm::mat4 () const [inline], [explicit]
```

9.8.2.12 operator glm::mat4x2()

```
egl::Program::UniformProxy::operator glm::mat4x2 () const [inline], [explicit]
```

9.8.2.13 operator glm::mat4x3()

```
egl::Program::UniformProxy::operator glm::mat4x3 () const [inline], [explicit]
```

9.8.2.14 operator glm::uvec2()

```
egl::Program::UniformProxy::operator glm::uvec2 () const [inline], [explicit]
```

9.8.2.15 operator glm::uvec3()

```
egl::Program::UniformProxy::operator glm::uvec3 () const [inline], [explicit]
```

9.8.2.16 operator glm::uvec4()

```
egl::Program::UniformProxy::operator glm::uvec4 () const [inline], [explicit]
```

9.8.2.17 operator glm::vec2()

```
egl::Program::UniformProxy::operator glm::vec2 () const [inline], [explicit]
```

9.8.2.18 operator glm::vec3()

```
egl::Program::UniformProxy::operator glm::vec3 () const [inline], [explicit]
```

9.8.2.19 operator glm::vec4()

```
egl::Program::UniformProxy::operator glm::vec4 () const [inline], [explicit]
```

9.8.2.20 operator int()

```
egl::Program::UniformProxy::operator int () const [inline], [explicit]
```

9.8.2.21 operator unsigned int()

```
egl::Program::UniformProxy::operator unsigned int () const [inline], [explicit]
```

9.8.2.22 operator=() [1/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::ivec2 & other) [inline]
```

9.8.2.23 operator=() [2/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::ivec3 & other) [inline]
```

9.8.2.24 operator=() [3/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::ivec4 & other) [inline]
```

9.8.2.25 operator=() [4/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat2 & other) [inline]
```

9.8.2.26 operator=() [5/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat2x3 & other) [inline]
```

9.8.2.27 operator=() [6/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat2x4 & other) [inline]
```

9.8.2.28 operator=() [7/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat3 & other) [inline]
```

9.8.2.29 operator=() [8/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat3x2 & other) [inline]
```

9.8.2.30 operator=() [9/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat3x4 & other) [inline]
```

9.8.2.31 operator=() [10/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat4 & other) [inline]
```

9.8.2.32 operator=() [11/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat4x2 & other) [inline]
```

9.8.2.33 operator=() [12/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::mat4x3 & other) [inline]
```

9.8.2.34 operator=() [13/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::uvec2 & other) [inline]
```

9.8.2.35 operator=() [14/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::uvec3 & other) [inline]
```

9.8.2.36 operator=() [15/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::uvec4 & other) [inline]
```

9.8.2.37 operator=() [16/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::vec2 & other) [inline]
```

9.8.2.38 operator=() [17/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::vec3 & other) [inline]
```

9.8.2.39 operator=() [18/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const glm::vec4 & other) [inline]
```

9.8.2.40 operator=() [19/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    const UniformProxy & other) [delete]
```

9.8.2.41 operator=() [20/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    float other) [inline]
```

9.8.2.42 operator=() [21/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    int other) [inline]
```

9.8.2.43 operator=() [22/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    UniformProxy && other) [delete]
```

9.8.2.44 operator=() [23/23]

```
UniformProxy & egl::Program::UniformProxy::operator= (
    unsigned int other) [inline]
```

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

- inc/EGL/program.h

9.9 egl::Program::UniformProxyConst Class Reference

```
#include <program.h>
```

Public Member Functions

- `UniformProxyConst (const Program &parent, int location)`
- `UniformProxyConst (UniformProxyConst &&other)=delete`
- `UniformProxyConst (const UniformProxyConst &other)=delete`
- `operator float () const`
- `operator glm::vec2 () const`
- `operator glm::vec3 () const`
- `operator glm::vec4 () const`
- `operator int () const`
- `operator glm::ivec2 () const`
- `operator glm::ivec3 () const`
- `operator glm::ivec4 () const`
- `operator unsigned int () const`
- `operator glm::uvec2 () const`
- `operator glm::uvec3 () const`
- `operator glm::uvec4 () const`
- `operator glm::mat2 () const`
- `operator glm::mat3 () const`
- `operator glm::mat4 () const`
- `operator glm::mat2x3 () const`
- `operator glm::mat3x2 () const`
- `operator glm::mat2x4 () const`
- `operator glm::mat4x2 () const`
- `operator glm::mat3x4 () const`
- `operator glm::mat4x3 () const`
- `UniformProxyConst & operator=(const UniformProxyConst &other)=delete`
- `UniformProxyConst & operator=(UniformProxyConst &&other)=delete`

9.9.1 Constructor & Destructor Documentation

9.9.1.1 UniformProxyConst() [1/3]

```
egl::Program::UniformProxyConst::UniformProxyConst (
    const Program & parent,
    int location) [inline]
```

9.9.1.2 UniformProxyConst() [2/3]

```
egl::Program::UniformProxyConst::UniformProxyConst (
    UniformProxyConst && other) [delete]
```

9.9.1.3 UniformProxyConst() [3/3]

```
egl::Program::UniformProxyConst::UniformProxyConst (
    const UniformProxyConst & other) [delete]
```

9.9.2 Member Function Documentation

9.9.2.1 operator float()

```
egl::Program::UniformProxyConst::operator float () const [inline], [explicit]
```

9.9.2.2 operator glm::ivec2()

```
egl::Program::UniformProxyConst::operator glm::ivec2 () const [inline], [explicit]
```

9.9.2.3 operator glm::ivec3()

```
egl::Program::UniformProxyConst::operator glm::ivec3 () const [inline], [explicit]
```

9.9.2.4 operator glm::ivec4()

```
egl::Program::UniformProxyConst::operator glm::ivec4 () const [inline], [explicit]
```

9.9.2.5 operator glm::mat2()

```
egl::Program::UniformProxyConst::operator glm::mat2 () const [inline], [explicit]
```

9.9.2.6 operator glm::mat2x3()

```
egl::Program::UniformProxyConst::operator glm::mat2x3 () const [inline], [explicit]
```

9.9.2.7 operator glm::mat2x4()

```
egl::Program::UniformProxyConst::operator glm::mat2x4 () const [inline], [explicit]
```

9.9.2.8 operator glm::mat3()

```
egl::Program::UniformProxyConst::operator glm::mat3 () const [inline], [explicit]
```

9.9.2.9 operator glm::mat3x2()

```
egl::Program::UniformProxyConst::operator glm::mat3x2 () const [inline], [explicit]
```

9.9.2.10 operator glm::mat3x4()

```
egl::Program::UniformProxyConst::operator glm::mat3x4 () const [inline], [explicit]
```

9.9.2.11 operator glm::mat4()

```
egl::Program::UniformProxyConst::operator glm::mat4 () const [inline], [explicit]
```

9.9.2.12 operator glm::mat4x2()

```
egl::Program::UniformProxyConst::operator glm::mat4x2 () const [inline], [explicit]
```

9.9.2.13 operator glm::mat4x3()

```
egl::Program::UniformProxyConst::operator glm::mat4x3 () const [inline], [explicit]
```

9.9.2.14 operator glm::uvec2()

```
egl::Program::UniformProxyConst::operator glm::uvec2 () const [inline], [explicit]
```

9.9.2.15 operator glm::uvec3()

```
egl::Program::UniformProxyConst::operator glm::uvec3 () const [inline], [explicit]
```

9.9.2.16 operator glm::uvec4()

```
egl::Program::UniformProxyConst::operator glm::uvec4 () const [inline], [explicit]
```

9.9.2.17 operator glm::vec2()

```
egl::Program::UniformProxyConst::operator glm::vec2 () const [inline], [explicit]
```

9.9.2.18 operator glm::vec3()

```
egl::Program::UniformProxyConst::operator glm::vec3 () const [inline], [explicit]
```

9.9.2.19 operator glm::vec4()

```
egl::Program::UniformProxyConst::operator glm::vec4 () const [inline], [explicit]
```

9.9.2.20 operator int()

```
egl::Program::UniformProxyConst::operator int () const [inline], [explicit]
```

9.9.2.21 operator unsigned int()

```
egl::Program::UniformProxyConst::operator unsigned int () const [inline], [explicit]
```

9.9.2.22 operator=() [1/2]

```
UniformProxyConst & egl::Program::UniformProxyConst::operator= (
    const UniformProxyConst & other) [delete]
```

9.9.2.23 operator=() [2/2]

```
UniformProxyConst & egl::Program::UniformProxyConst::operator= (
    UniformProxyConst && other) [delete]
```

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

- inc/EGL/program.h

9.10 egl::VertexBuffer Class Reference

```
#include <vertexBuffer.h>
```

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

- inc/EGL/vertexBuffer.h

9.11 egl::WindowContext Class Reference

An abstract class to contain an window based application.

```
#include <windowContext.h>
```

Public Member Functions

- [WindowContext](#) (int width, int height, const char *windowName)
Construct a new Window Context object with given height, width and name.
- [WindowContext](#) (WindowContext &&other)
- [WindowContext](#) (const WindowContext &other)=delete
- [~WindowContext](#) ()
- virtual void [run](#) ()=0
Virtual run method to overload when implementing a child class.
- [WindowContext & operator=](#) ([WindowContext](#) &&other)
- [WindowContext & operator=](#) (const [WindowContext](#) &other)=delete

Protected Member Functions

- void [useContext \(\)](#)
Makes the owned window the current context.
- bool [shouldClose \(\)](#)
Checks if the Window should close.
- void [swapBuffers \(\)](#)
Swaps the display buffers.

Protected Attributes

- GLFWwindow * [window](#) = NULL
GLFW window handle for low level GLFW access.

9.11.1 Detailed Description

An abstract class to contain an window based application.

Instantiates a window in the constructor using GLFW and destroys it once it goes out of scope. That window can be bound and used in the child class in the [WindowContext::run](#) method implementation containing the setup and rendering loop.

Note

Calling [useContext](#) is recommended in [run](#) to ensure the local context is used.

[egl::WindowContext](#) is not thread safe and may only be used on the main thread.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 [WindowContext\(\)](#) [1/3]

```
egl::WindowContext::WindowContext (
    int width,
    int height,
    const char * windowName)
```

Construct a new Window Context object with given height, width and name.

Exceptions

<code>std::runtime_error</code>	If GLFW could not create a Window.
<code>std::runtime_error</code>	If the GLFW Window is invalid.
<code>std::runtime_error</code>	If GLEW could not be initialized.

Parameters

<i>width</i>	Width of the Window in pixels.
<i>height</i>	Height of the Window in pixels.
<i>windowName</i>	C-style NULL terminated string window name.

9.11.2.2 WindowContext() [2/3]

```
egl::WindowContext::WindowContext (
    WindowContext && other)
```

9.11.2.3 WindowContext() [3/3]

```
egl::WindowContext::WindowContext (
    const WindowContext & other) [delete]
```

9.11.2.4 ~WindowContext()

```
egl::WindowContext::~WindowContext ()
```

9.11.3 Member Function Documentation**9.11.3.1 operator=() [1/2]**

```
WindowContext & egl::WindowContext::operator= (
    const WindowContext & other) [delete]
```

9.11.3.2 operator=() [2/2]

```
WindowContext & egl::WindowContext::operator= (
    WindowContext && other)
```

9.11.3.3 run()

```
virtual void egl::WindowContext::run () [pure virtual]
```

Virtual run method to overload when implementing a child class.

Virtual run method to contain the rendering and window handling for your Application.

Note

Calling useContext is recommended in run to ensure the local context is used.

9.11.3.4 shouldClose()

```
bool egl::WindowContext::shouldClose () [protected]
```

Checks if the Window should close.

Exceptions

<code>std::runtime_error</code>	If the GLFW Window is invalid.
---------------------------------	--------------------------------

9.11.3.5 swapBuffers()

```
void egl::WindowContext::swapBuffers () [protected]
```

Swaps the display buffers.

Exceptions

<code>std::runtime_error</code>	If the GLFW Window is invalid.
---------------------------------	--------------------------------

9.11.3.6 useContext()

```
void egl::WindowContext::useContext () [protected]
```

Makes the owned window the current context.

Exceptions

<code>std::runtime_error</code>	If the GLFW window is invalid.
---------------------------------	--------------------------------

9.11.4 Field Documentation

9.11.4.1 window

```
GLFWwindow* egl::WindowContext::window = NULL [protected]
```

GLFW window handle for low level GLFW access.

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

- inc/EGL/windowContext.h

Chapter 10

File Documentation

10.1 inc/EGL/buffer.h File Reference

```
#include <string>
#include <stdexcept>
#include <cstdint>
#include <vector>
```

Data Structures

- class [egl::Buffer](#)

Namespaces

- namespace [egl](#)

Enumerations

- enum class [egl::BufferType](#) {
 [egl::Array](#) , [egl::AtomicCounter](#) , [egl::CopyRead](#) , [egl::CopyWrite](#) ,
 [egl::DispatchIndirect](#) , [egl::DrawIndirect](#) , [egl::ElementArray](#) , [egl::PixelPack](#) ,
 [egl::PixelUnpack](#) , [egl::Query](#) , [egl::ShaderStorage](#) , [egl::Texture](#) ,
 [egl::TransformFeedback](#) , [egl::Uniform](#) }
- Enum to indicate the type of Buffer.*
- enum class [egl::BufferUsage](#) {
 [egl::StreamDraw](#) , [egl::StreamRead](#) , [egl::StreamCopy](#) , [egl::StaticDraw](#) ,
 [egl::StaticRead](#) , [egl::StaticCopy](#) , [egl::DynamicDraw](#) , [egl::DynamicRead](#) ,
 [egl::DynamicCopy](#) }
- Enum to indicate Buffer Usage.*
- enum class [egl::MapUsage](#) : uint32_t {
 [egl::None](#) = 0 , [egl::Read](#) = 1 << 0 , [egl::Write](#) = 1 << 1 , [egl::Persistent](#) = 1 << 2 ,
 [egl::Coherent](#) = 1 << 3 , [egl::InvalidRange](#) = 1 << 4 , [egl::InvalidateBuffer](#) = 1 << 5 , [egl::FlushExplicit](#) = 1
 << 6 ,
 [egl::Unsynchronized](#) = 1 << 7 }
- Enum flags to indicate Buffer map usage.*
- enum class [egl::BufferFlag](#) : uint32_t {
 [egl::None](#) = 0 , [egl::DynamicStorage](#) = 1 << 0 , [egl::MapRead](#) = 1 << 1 , [egl::MapWrite](#) = 1 << 2 ,
 [egl::MapPersistent](#) = 1 << 3 , [egl::MapCoherent](#) = 1 << 4 , [egl::ClientStorage](#) = 1 << 5 }
- Enum flags for explicit Buffer usage in setStorage.*

Functions

- `MapUsage egl::operator| (MapUsage a, MapUsage b)`
- `MapUsage egl::operator& (MapUsage a, MapUsage b)`
- `MapUsage & egl::operator|= (MapUsage &a, MapUsage b)`
- `BufferFlag egl::operator| (BufferFlag a, BufferFlag b)`
- `BufferFlag egl::operator& (BufferFlag a, BufferFlag b)`
- `BufferFlag & egl::operator|= (BufferFlag &a, BufferFlag b)`
- `unsigned int egl::toGLenum (BufferType type)`
`Converts a BufferType enum into a Glenum.`
- `unsigned int egl::toGLenum (BufferUsage usage)`
`Converts a BufferUsage enum into a Glenum.`
- `unsigned int egl::toGLenum (MapUsage usage)`
`Converts a MapUsage enum into a Glenum.`
- `unsigned int egl::toGLenum (BufferFlag flag)`
`Converts a BufferFlag enum into a Glenum.`
- `bool egl::validateMapUsage (MapUsage usage, std::string &error)`
`Checks if the given MapUsage flag combination is valid.`
- `bool egl::validateBufferFlag (BufferFlag flag, std::string &error)`
`Checks if the given BufferFlag flag combination is valid.`

10.2 buffer.h

[Go to the documentation of this file.](#)

```

00001 #ifndef EGL_BUFFER_H
00002 #define EGL_BUFFER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <cstdint>
00007 #include <vector>
00008
00009 namespace egl {
00010
00014 enum class BufferType {
00015     Array,
00016     AtomicCounter,
00017     CopyRead,
00018     CopyWrite,
00019     DispatchIndirect,
00020     DrawIndirect,
00021     ElementArray,
00022     PixelPack,
00023     PixelUnpack,
00024     Query,
00025     ShaderStorage,
00026     Texture,
00027     TransformFeedback,
00028     Uniform
00029 };
00030
00059 enum class BufferUsage {
00060     StreamDraw,
00061     StreamRead,
00062     StreamCopy,
00063     StaticDraw,
00064     StaticRead,
00065     StaticCopy,
00066     DynamicDraw,
00067     DynamicRead,
00068     DynamicCopy
00069 };
00070
00074 enum class MapUsage : uint32_t {
00075     None          = 0,
00076     Read          = 1 << 0,
00077     Write         = 1 << 1,
00078     Persistent    = 1 << 2,

```

```
00079     Coherent          = 1 << 3,
00080     InvalidRange       = 1 << 4,
00081     InvalidateBuffer  = 1 << 5,
00082     FlushExplicit      = 1 << 6,
00083     Unsynchronized     = 1 << 7
00084 };
00085
00086 inline MapUsage operator|(MapUsage a, MapUsage b) {
00087     return static_cast<MapUsage>(
00088         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00089     );
00090 }
00091
00092 inline MapUsage operator&(MapUsage a, MapUsage b) {
00093     return static_cast<MapUsage>(
00094         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00095     );
00096 }
00097
00098 inline MapUsage& operator|=(MapUsage& a, MapUsage b) {
00099     a = a | b;
00100     return a;
00101 }
00102
00103 enum class BufferFlag : uint32_t {
00104     None          = 0,
00105     DynamicStorage = 1 << 0,
00106     MapRead        = 1 << 1,
00107     MapWrite        = 1 << 2,
00108     MapPersistent   = 1 << 3,
00109     MapCoherent     = 1 << 4,
00110     ClientStorage   = 1 << 5
00111 };
00112
00113 inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
00114     return static_cast<BufferFlag>(
00115         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00116     );
00117 }
00118
00119 inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
00120     return static_cast<BufferFlag>(
00121         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00122     );
00123 }
00124
00125 inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
00126     a = a | b;
00127     return a;
00128 }
00129
00130 unsigned int toGLenum(BufferType type);
00131
00132
00133 unsigned int toGLenum(BufferUsage usage);
00134
00135 unsigned int toGLenum(MapUsage usage);
00136
00137 unsigned int toGLenum(BufferFlag flag);
00138
00139
00140 bool validateMapUsage(MapUsage usage, std::string& error);
00141
00142 bool validateBufferFlag(BufferFlag flag, std::string& error);
00143
00144
00145 class Buffer {
00146 protected:
00147     unsigned int _id = 0;
00148     bool _mapped = false;
00149     MapUsage _mapUsage = MapUsage::None;
00150     BufferFlag _flags = BufferFlag::None;
00151     BufferType _type;
00152
00153     void _delete();
00154     void _check();
00155
00156 public:
00157     Buffer() = delete;
00158     Buffer(BufferType type);
00159     Buffer(Buffer&& other);
00160     Buffer(const Buffer& other) = delete;
00161     ~Buffer() noexcept;
00162
00163     void bind() const;
00164
00165     int64_t size() const;
00166
00167     BufferType getType() const;
00168 }
```

```

00222     void setData(int64_t size, const void* data, BufferUsage usage);
00223
00230     template <typename T>
00231     void setData(std::vector<T> data, BufferUsage usage) { setData(data.size() * sizeof(T),
00232     (void*)data.data(), usage); }
00232
00243     void setStorage(int64_t size, const void* data, BufferFlag flags);
00244
00254     template <typename T>
00255     void setStorage(std::vector<T> data, BufferFlag flags) { setStorage(data.size() * sizeof(T),
00256     (void*)data.data(), flags); }
00256
00269     void setSubData(int64_t offset, int64_t size, const void* data);
00270
00283     void getSubData(int64_t offset, int64_t size, void* data);
00284
00302     void* map(int64_t offset, int64_t length, MapUsage access);
00303
00309     void unmap();
00310
00311     Buffer& operator=(Buffer&& other);
00312     Buffer& operator=(const Buffer& other) = delete;
00313 };
00314
00315 }
00316
00317 #endif

```

10.3 inc/EGL/debug.h File Reference

Namespaces

- namespace `egl`

Macros

- `#define GL_CALL(x)`
Runs the given code and checks for errors.
- `#define DEBUG_ONLY(x)`
Runs given code only if DEBUG_BUILD is defined.

Functions

- `const char * egl::glErrorString (unsigned int err)`
Gets a c-style string from from a OpenGL enum.
- `void egl::glCheckError (const char *func, const char *file, int line)`
Checks and prints if any OpenGL error has occurred.

10.3.1 Macro Definition Documentation

10.3.1.1 DEBUG_ONLY

```
#define DEBUG_ONLY(
    x)
```

Value:

```
((void)0)
```

Runs given code only if DEBUG_BUILD is defined.

10.3.1.2 GL_CALL

```
#define GL_CALL(
    x)
```

Value:

x

Runs the given code and checks for errors.

Note

For performance reasons `egl::glCheckError` is only called when `DEBUG_BUILD` is defined.

Warning

The code is run in a scope below the current.

Parameters

x	The code to run before checking for errors
---	--

10.4 debug.h

[Go to the documentation of this file.](#)

```
00001 #ifndef EGL_DEBUG_H
00002 #define EGL_DEBUG_H
00003
00004 namespace egl {
00005
00015 const char* glErrorString(unsigned int err);
00016
00028 void glCheckError(const char* func, const char* file, int line);
00029
00030 };
00031
00040
00044
00045 #ifdef DEBUG_BUILD
00046
00047     #define GL_CALL(x) do { x; egl::glCheckError(#x, __FILE__, __LINE__); } while(0)
00048
00049     #define DEBUG_ONLY(x) x
00050
00051 #else
00052
00053     #define GL_CALL(x) x
00054
00055     #define DEBUG_ONLY(x) ((void)0)
00056
00057 #endif
00058
00059 #endif
```

10.5 inc/EGL/mainpage.md File Reference

10.6 inc/EGL/program.h File Reference

```
#include <string>
#include <stdexcept>
#include <unordered_map>
#include <glm/vec2.hpp>
#include <glm/vec3.hpp>
#include <glm/vec4.hpp>
#include <glm/matrix.hpp>
```

Data Structures

- class `egl::ProgramLinkError`
Exception thrown when Program linking fails.
- class `egl::ProgramValidateError`
Exception thrown when Program validation fails, mainly used in debug builds.
- struct `egl::_uniformData`
- class `egl::Program`
Program class to abstract OpenGL Shader Programs.
- class `egl::Program::UniformProxy`
- class `egl::Program::UniformProxyConst`

Namespaces

- namespace `egl`

Typedefs

- typedef struct `egl::_uniformData egl::UniformData`

10.7 program.h

[Go to the documentation of this file.](#)

```

00001 #ifndef EGL_PROGRAM_H
00002 #define EGL_PROGRAM_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <unordered_map>
00007 #include <glm/vec2.hpp>
00008 #include <glm/vec3.hpp>
00009 #include <glm/vec4.hpp>
00010 #include <glm/matrix.hpp>
00011
00012 namespace egl {
00013
00014 class Shader;
00015
00016 class ProgramLinkError : public std::runtime_error {
00017 public:
00018     ProgramLinkError(const std::string& infoLog)
00019         : std::runtime_error(
00020             "Program link failed:\n" + infoLog) {}
00021 };
00022
00023 class ProgramValidateError : public std::runtime_error {
00024 public:
00025     ProgramValidateError(const std::string& infoLog)
00026         : std::runtime_error(
00027             "Program validation failed:\n" + infoLog) {}
00028 };
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068

```

```

00069
00070     void _delete();
00071     void _check() const;
00072     void _ensure() const;
00073     void _queryUniformData();
00074     void _setupUniform(int loc, int sizeCheck, int typeCheck) const;
00075     std::string _getError();
00076
00077     class UniformProxy {
00078     private:
00079         Program& _parent;
00080         int _location;
00081
00082     public:
00083         UniformProxy(Program& parent, int location) : _parent(parent), _location(location) {}
00084         UniformProxy(UniformProxy&& other) = delete;
00085         UniformProxy(const UniformProxy& other) = delete;
00086
00087         explicit operator float() const { float val; _parent.getUniform(_location, val); return val; }
00088         explicit operator glm::vec2() const { glm::vec2 val; _parent.getUniform(_location, val);
00089             return val; }
00090         explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
00091             return val; }
00092         explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
00093             return val; }
00094         explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
00095         explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
00096             return val; }
00097         explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
00098             return val; }
00099         explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
00100             return val; }
00101         explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val);
00102             return val; }
00103         explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val);
00104             return val; }
00105         explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val);
00106             return val; }
00107         explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val);
00108             return val; }
00109         explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
00110             val; }
00111         explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
00112             val; }
00113         explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
00114             val; }
00115         explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
00116             return val; }
00117         explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
00118             return val; }
00119         explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
00120             return val; }
00121         explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
00122             return val; }
00123         explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
00124             return val; }
00125         explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
00126             return val; }
00127
00128         UniformProxy& operator=(float other) { _parent.setUniform(_location, other); return *this; }
00129         UniformProxy& operator=(const glm::vec2& other) { _parent.setUniform(_location, other); return
00130             *this; }
00131         UniformProxy& operator=(const glm::vec3& other) { _parent.setUniform(_location, other); return
00132             *this; }
00133         UniformProxy& operator=(const glm::vec4& other) { _parent.setUniform(_location, other); return
00134             *this; }
00135         UniformProxy& operator=(int other) { _parent.setUniform(_location, other); return *this; }
00136         UniformProxy& operator=(const glm::ivec2& other) { _parent.setUniform(_location, other);
00137             return *this; }
00138         UniformProxy& operator=(const glm::ivec3& other) { _parent.setUniform(_location, other);
00139             return *this; }
00140         UniformProxy& operator=(const glm::ivec4& other) { _parent.setUniform(_location, other);
00141             return *this; }
00142         UniformProxy& operator=(unsigned int other) { _parent.setUniform(_location, other); return
00143             *this; }
00144         UniformProxy& operator=(const glm::uvec2& other) { _parent.setUniform(_location, other);
00145             return *this; }
00146         UniformProxy& operator=(const glm::uvec3& other) { _parent.setUniform(_location, other);
00147             return *this; }
00148         UniformProxy& operator=(const glm::uvec4& other) { _parent.setUniform(_location, other);
00149             return *this; }
00150         UniformProxy& operator=(const glm::mat2& other) { _parent.setUniform(_location, other); return
00151             *this; }
00152         UniformProxy& operator=(const glm::mat3& other) { _parent.setUniform(_location, other); return
00153             *this; }
00154         UniformProxy& operator=(const glm::mat4& other) { _parent.setUniform(_location, other); return
00155             *this; }

```

```

00124     UniformProxy& operator=(const glm::mat2x3& other) { _parent.setUniform(_location, other);
00125     return *this; }
00126     UniformProxy& operator=(const glm::mat3x2& other) { _parent.setUniform(_location, other);
00127     return *this; }
00128     UniformProxy& operator=(const glm::mat2x4& other) { _parent.setUniform(_location, other);
00129     return *this; }
00130     UniformProxy& operator=(const glm::mat4x2& other) { _parent.setUniform(_location, other);
00131     return *this; }
00132     UniformProxy& operator=(const glm::mat3x4& other) { _parent.setUniform(_location, other);
00133     return *this; }
00134     UniformProxy& operator=(const glm::mat4x3& other) { _parent.setUniform(_location, other);
00135     return *this; }
00136
00137     UniformProxy& operator=(const UniformProxy& other) = delete;
00138     UniformProxy& operator=(UniformProxy&& other) = delete;
00139 };
00140
00141     class UniformProxyConst {
00142     private:
00143         const Program& _parent;
00144         int _location;
00145
00146     public:
00147         UniformProxyConst(const Program& parent, int location) : _parent(parent), _location(location)
00148     {}
00149         UniformProxyConst(UniformProxyConst&& other) = delete;
00150         UniformProxyConst(const UniformProxyConst& other) = delete;
00151
00152         explicit operator float() const { float val; _parent.getUniform(_location, val); return val; }
00153         explicit operator glm::vec2() const { glm::vec2 val; _parent.getUniform(_location, val);
00154             return val; }
00155         explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
00156             return val; }
00157         explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
00158             return val; }
00159         explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
00160         explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
00161             return val; }
00162         explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
00163             return val; }
00164         explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
00165             return val; }
00166         explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val);
00167             return val; }
00168         explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val);
00169             return val; }
00170         explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val);
00171             return val; }
00172         explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val);
00173             return val; }
00174
00175         explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
00176             val; }
00177         explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
00178             val; }
00179         explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
00180             val; }
00181         explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
00182             return val; }
00183         explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
00184             return val; }
00185         explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
00186             return val; }
00187         explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
00188             return val; }
00189         explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
00190             return val; }
00191         explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
00192             return val; }
00193
00194         UniformProxyConst& operator=(const UniformProxyConst& other) = delete;
00195         UniformProxyConst& operator=(UniformProxyConst&& other) = delete;
00196     };
00197
00198     public:
00199         Program();
00200         Program(Program&& other);
00201         Program(const Program&) = delete; // OpenGL Programs are not copy safe
00202         ~Program();
00203
00204         void reset();
00205
00206         bool attached(const Shader& shader);
00207
00208         void attach(const Shader& shader);
00209
00210         void detach(const Shader& shader);
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224

```

```
00230     bool linked() const { return _linked; }
00231
00239     void link();
00240
00247     void bind() const;
00248
00252     static void unbind();
00253
00264     int getUniformLocation(const std::string& name) const;
00265
00282     void setUniform(int location, float data);
00283     void setUniform(int location, const glm::vec2& data);
00284     void setUniform(int location, const glm::vec3& data);
00285     void setUniform(int location, const glm::vec4& data);
00286     void setUniform(int location, int data);
00287     void setUniform(int location, const glm::ivec2& data);
00288     void setUniform(int location, const glm::ivec3& data);
00289     void setUniform(int location, const glm::ivec4& data);
00290     void setUniform(int location, unsigned int data);
00291     void setUniform(int location, const glm::uvec2& data);
00292     void setUniform(int location, const glm::uvec3& data);
00293     void setUniform(int location, const glm::uvec4& data);
00294     void setUniform(int location, const glm::mat2& data);
00295     void setUniform(int location, const glm::mat3& data);
00296     void setUniform(int location, const glm::mat4& data);
00297     void setUniform(int location, const glm::mat2x3& data);
00298     void setUniform(int location, const glm::mat3x2& data);
00299     void setUniform(int location, const glm::mat2x4& data);
00300     void setUniform(int location, const glm::mat4x2& data);
00301     void setUniform(int location, const glm::mat3x4& data);
00302     void setUniform(int location, const glm::mat4x3& data);
00319     void setUniform(const std::string& name, float data) { setUniform(getUniformLocation(name), data);
00320     }
00320     void setUniform(const std::string& name, const glm::vec2& data) {
00321         setUniform(getUniformLocation(name), data); }
00321     void setUniform(const std::string& name, const glm::vec3& data) {
00322         setUniform(getUniformLocation(name), data); }
00322     void setUniform(const std::string& name, const glm::vec4& data) {
00323         setUniform(getUniformLocation(name), data); }
00323     void setUniform(const std::string& name, int data) { setUniform(getUniformLocation(name), data); }
00324     void setUniform(const std::string& name, const glm::ivec2& data) {
00325         setUniform(getUniformLocation(name), data); }
00325     void setUniform(const std::string& name, const glm::ivec3& data) {
00326         setUniform(getUniformLocation(name), data); }
00326     void setUniform(const std::string& name, const glm::ivec4& data) {
00327         void setUniform(const std::string& name, unsigned int data) { setUniform(getUniformLocation(name),
00328             data); }
00328         void setUniform(const std::string& name, const glm::uvec2& data) {
00329             setUniform(getUniformLocation(name), data); }
00329         void setUniform(const std::string& name, const glm::uvec3& data) {
00330             setUniform(getUniformLocation(name), data); }
00330         void setUniform(const std::string& name, const glm::uvec4& data) {
00331             setUniform(getUniformLocation(name), data); }
00331         void setUniform(const std::string& name, const glm::mat2& data) {
00332             setUniform(getUniformLocation(name), data); }
00332         void setUniform(const std::string& name, const glm::mat3& data) {
00333             void setUniform(const std::string& name, const glm::mat4& data) {
00334                 setUniform(getUniformLocation(name), data); }
00334         void setUniform(const std::string& name, const glm::mat2x3& data) {
00335             setUniform(getUniformLocation(name), data); }
00335         void setUniform(const std::string& name, const glm::mat3x2& data) {
00336             void setUniform(const std::string& name, const glm::mat2x4& data) {
00337                 setUniform(getUniformLocation(name), data); }
00337         void setUniform(const std::string& name, const glm::mat4x2& data) {
00338             void setUniform(const std::string& name, const glm::mat3x4& data) {
00339                 void setUniform(const std::string& name, const glm::mat4x3& data) {
00340                     setUniform(getUniformLocation(name), data); }
00340
00355         void getUniform(int location, float& data) const;
00356         void getUniform(int location, glm::vec2& data) const;
00357         void getUniform(int location, glm::vec3& data) const;
00358         void getUniform(int location, glm::vec4& data) const;
00359         void getUniform(int location, int& data) const;
00360         void getUniform(int location, glm::ivec2& data) const;
00361         void getUniform(int location, glm::ivec3& data) const;
00362         void getUniform(int location, glm::ivec4& data) const;
00363         void getUniform(int location, unsigned int& data) const;
00364         void getUniform(int location, glm::uvec2& data) const;
00365         void getUniform(int location, glm::uvec3& data) const;
00366         void getUniform(int location, glm::uvec4& data) const;
00367         void getUniform(int location, glm::mat2& data) const;
00368         void getUniform(int location, glm::mat3& data) const;
```

```

00369     void getUniform(int location, glm::mat4& data) const;
00370     void getUniform(int location, glm::mat2x3& data) const;
00371     void getUniform(int location, glm::mat3x2& data) const;
00372     void getUniform(int location, glm::mat2x4& data) const;
00373     void getUniform(int location, glm::mat4x2& data) const;
00374     void getUniform(int location, glm::mat3x4& data) const;
00375     void getUniform(int location, glm::mat4x3& data) const;
00376     void getUniform(const std::string& name, float& data) const { getUniform(getUniformLocation(name),
00377         data); }
00378     void getUniform(const std::string& name, glm::vec2& data) const {
00379         getUniform(getUniformLocation(name), data); }
00380     void getUniform(const std::string& name, glm::vec3& data) const {
00381         getUniform(getUniformLocation(name), data); }
00382     void getUniform(const std::string& name, glm::vec4& data) const {
00383         getUniform(getUniformLocation(name), data); }
00384     void getUniform(const std::string& name, int& data) const { getUniform(getUniformLocation(name),
00385         data); }
00386     void getUniform(const std::string& name, glm::ivec2& data) const {
00387         getUniform(getUniformLocation(name), data); }
00388     void getUniform(const std::string& name, glm::ivec3& data) const {
00389         getUniform(getUniformLocation(name), data); }
00390     void getUniform(const std::string& name, glm::ivec4& data) const {
00391         getUniform(getUniformLocation(name), data); }
00392     void getUniform(const std::string& name, unsigned int& data) const {
00393         getUniform(getUniformLocation(name), data); }
00394     void getUniform(const std::string& name, glm::uvec2& data) const {
00395         getUniform(getUniformLocation(name), data); }
00396     void getUniform(const std::string& name, glm::uvec3& data) const {
00397         getUniform(getUniformLocation(name), data); }
00398     void getUniform(const std::string& name, glm::uvec4& data) const {
00399         getUniform(getUniformLocation(name), data); }
00400     void getUniform(const std::string& name, glm::mat2& data) const {
00401         getUniform(getUniformLocation(name), data); }
00402     void getUniform(const std::string& name, glm::mat3& data) const {
00403         getUniform(getUniformLocation(name), data); }
00404     void getUniform(const std::string& name, glm::mat4& data) const {
00405         getUniform(getUniformLocation(name), data); }
00406     void getUniform(const std::string& name, glm::mat2x3& data) const {
00407         getUniform(getUniformLocation(name), data); }
00408     void getUniform(const std::string& name, glm::mat3x2& data) const {
00409         getUniform(getUniformLocation(name), data); }
00410     void getUniform(const std::string& name, glm::mat2x4& data) const {
00411         getUniform(getUniformLocation(name), data); }
00412     void getUniform(const std::string& name, glm::mat4x2& data) const {
00413         getUniform(getUniformLocation(name), data); }
00414     void getUniform(const std::string& name, glm::mat3x4& data) const {
00415         getUniform(getUniformLocation(name), data); }
00416     void getUniform(const std::string& name, glm::mat4x3& data) const {
00417         getUniform(getUniformLocation(name), data); }
00418
00419     Program& operator=(Program&& other);
00420     Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe
00421
00422     UniformProxy operator[](int location) { return UniformProxy(*this, location); }
00423     UniformProxy operator[](const std::string& name) { return UniformProxy(*this,
00424         getUniformLocation(name)); }
00425     UniformProxyConst operator[](int location) const { return UniformProxyConst(*this, location); }
00426     UniformProxyConst operator[](const std::string& name) const { return UniformProxyConst(*this,
00427         getUniformLocation(name)); }
00428
00429 };
00430
00431 }
00432
00433 }
00434
00435 #endif

```

10.8 inc/EGL/shader.h File Reference

```

#include <string>
#include <stdexcept>
#include <iostream>

```

Data Structures

- class [egl::ShaderCompileError](#)

Exception thrown when `Shader` compilation fails.

- class `egl::Shader`
Shader class to abstract OpenGL Shaders.

Namespaces

- namespace `egl`

Enumerations

- enum class `egl::ShaderType` {
 `egl::Fragment` , `egl::Vertex` , `egl::Geometry` , `egl::TessEvaluation` ,
 `egl::TessControl` , `egl::Compute` }
- ShaderType enum to indicate usage.*

Functions

- `constexpr std::string egl::shaderTypeToString (ShaderType type)`
Converts the `ShaderType` enum into a `std::string`.
- `unsigned int egl::toGLenum (ShaderType type)`
Converts the `ShaderType` enum into an OpenGL enum.

10.9 shader.h

Go to the documentation of this file.

```
00001 #ifndef EGL_SHADER_H
00002 #define EGL_SHADER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <iostream> // std::istream forward-declared
00007
00008 namespace egl {
00009
00010 class Program;
00011
00015 enum class ShaderType {
00016     Fragment,
00017     Vertex,
00018     Geometry,
00019     TessEvaluation,
00020     TessControl,
00021     Compute
00022 };
00023
00030 constexpr std::string shaderTypeToString(ShaderType type);
00031
00040 unsigned int toGLenum(ShaderType type);
00041
00045 class ShaderCompileError : public std::runtime_error {
00046 public:
00053     ShaderCompileError(ShaderType ShaderType,
00054             const std::string& infoLog)
00055         : std::runtime_error(
00056             "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
00057 };
00058
00068 class Shader {
00069 protected:
00070     unsigned int _id = 0;
00071     ShaderType _type;
00072     bool _compiled = false;
00073
00074     void _delete();
```

```

00075     void _check();
00076     void _ensure();
00077     std::string _getError();
00078
00079 public:
00080     Shader() = delete;
00081     Shader(ShaderType type);
00096     Shader(ShaderType type, const char* src);
00105     Shader(ShaderType type, const std::string& src);
00117     Shader(ShaderType type, std::istream& in);
00129     Shader(ShaderType type, std::istream&& in);
00130
00131     Shader(Shader&& other);
00132     Shader(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00133     ~Shader() noexcept;
00134
00143     void reset();
00144
00148     bool compiled() const { return _compiled; }
00149
00153     ShaderType getType() const { return _type; }
00154
00164     void compile(const char* src);
00165
00179     void compile(std::istream& in);
00180
00194     void compile(std::istream&& in);
00195
00206     void compile(const std::string& str);
00207
00208     friend Program;
00209
00210     Shader& operator=(Shader&& other);
00211     Shader& operator=(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00212 };
00213
00214 }
00215
00216 #endif

```

10.10 inc/EGL/vertexBuffer.h File Reference

Data Structures

- class [egl::VertexBuffer](#)

Namespaces

- namespace [egl](#)

10.11 vertexBuffer.h

[Go to the documentation of this file.](#)

```

00001 #ifndef EGL_VERTEX_BUFFER_H
00002 #define EGL_VERTEX_BUFFER_H
00003
00004 namespace egl {
00005
00006     class VertexBuffer {
00007
00008     };
00009
00010 }
00011
00012 #endif

```

10.12 inc/EGL/windowContext.h File Reference

```
#include <GLFW/glfw3.h>
```

Data Structures

- class [egl::WindowContext](#)
An abstract class to contain an window based application.

Namespaces

- namespace [egl](#)

Functions

- bool [egl::initGLFW \(\)](#)
Initialize GLFW.
- void [egl::terminateGLFW \(\)](#)
Terminates GLFW.
- void [egl::pollEvents \(\)](#)
Polls GLFW events.

10.13 windowContext.h

Go to the documentation of this file.

```
00001 #ifndef WINDOW_CONTEXT
00002 #define WINDOW_CONTEXT
00003
00004 #include <GLFW/glfw3.h>
00005
00006 namespace egl {
00007
00017 bool initGLFW();
00018
00024 void terminateGLFW();
00025
00031 void pollEvents();
00032
00042 class WindowContext {
00043 private:
00044     bool _ownsGLFW = false;
00045
00046 protected:
00050     GLFWwindow* window = NULL;
00051
00057     void useContext();
00058
00064     bool shouldClose();
00065
00071     void swapBuffers();
00072
00073 public:
00085     WindowContext(int width, int height, const char* windowName);
00086     WindowContext(WindowContext&& other);
00087     WindowContext(const WindowContext& other) = delete;
00088     ~WindowContext();
00089
00097     virtual void run() = 0;
00098
00099     WindowContext& operator=(WindowContext&& other);
00100     WindowContext& operator=(const WindowContext& other) = delete;
00101 };
00102
00103 }
00104
00105 #endif
```


Index

_check
 egl::Buffer, 29
 egl::Program, 41
 egl::Shader, 70
_compiled
 egl::Shader, 73
_delete
 egl::Buffer, 29
 egl::Program, 41
 egl::Shader, 70
_ensure
 egl::Program, 41
 egl::Shader, 70
_flags
 egl::Buffer, 34
_getError
 egl::Program, 41
 egl::Shader, 70
_id
 egl::Buffer, 34
 egl::Program, 63
 egl::Shader, 73
_linked
 egl::Program, 63
_mapUsage
 egl::Buffer, 34
_mapped
 egl::Buffer, 34
_queryUniformData
 egl::Program, 41
_setupUniform
 egl::Program, 41
_type
 egl::Buffer, 34
 egl::Shader, 73
_uniformData
 egl::Program, 64
_uniformIndexMap
 egl::Program, 64
_uniformLocationIndexMap
 egl::Program, 64
~Buffer
 egl::Buffer, 29
~Program
 egl::Program, 41
~Shader
 egl::Shader, 69
~WindowContext
 egl::WindowContext, 86

Array
 egl, 17
arraySize
 egl::_uniformData, 27
AtomicCounter
 egl, 17
attach
 egl::Program, 42
attached
 egl::Program, 42
bind
 egl::Buffer, 29
 egl::Program, 43
Buffer
 egl::Buffer, 29
BufferFlag
 egl, 17
BufferType
 egl, 17
BufferUsage
 egl, 18
ClientStorage
 egl, 17
Coherent
 egl, 19
compile
 egl::Shader, 70, 71
compiled
 egl::Shader, 72
Compute
 egl, 20
CopyRead
 egl, 17
CopyWrite
 egl, 17
debug.h
 DEBUG_ONLY, 92
 GL_CALL, 92
DEBUG_ONLY
 debug.h, 92
detach
 egl::Program, 43
DispatchIndirect
 egl, 17
DrawIndirect
 egl, 18
DynamicCopy

egl, 19
 DynamicDraw
 egl, 19
 DynamicRead
 egl, 19
 DynamicStorage
 egl, 17

 egl, 15
 Array, 17
 AtomicCounter, 17
 BufferFlag, 17
 BufferType, 17
 BufferUsage, 18
 ClientStorage, 17
 Coherent, 19
 Compute, 20
 CopyRead, 17
 CopyWrite, 17
 DispatchIndirect, 17
 DrawIndirect, 18
 DynamicCopy, 19
 DynamicDraw, 19
 DynamicRead, 19
 DynamicStorage, 17
 ElementArray, 18
 FlushExplicit, 19
 Fragment, 20
 Geometry, 20
 glCheckError, 20
 glErrorString, 20
 initGLFW, 21
 InvalidateBuffer, 19
 InvalidRange, 19
 MapCoherent, 17
 MapPersistent, 17
 MapRead, 17
 MapUsage, 19
 MapWrite, 17
 None, 17, 19
 operator&, 21
 operator | , 21
 operator |=, 22
 Persistent, 19
 PixelPack, 18
 PixelUnpack, 18
 pollEvents, 22
 Query, 18
 Read, 19
 ShaderStorage, 18
 ShaderType, 19
 shaderTypeToString, 22
 StaticCopy, 19
 StaticDraw, 19
 StaticRead, 19
 StreamCopy, 19
 StreamDraw, 19
 StreamRead, 19
 terminateGLFW, 22

 TessControl, 20
 TessEvaluation, 20
 Texture, 18
 toGLenum, 23
 TransformFeedback, 18
 Uniform, 18
 UniformData, 17
 Unsynchronized, 19
 validateBufferFlag, 24
 validateMapUsage, 24
 Vertex, 20
 Write, 19

 egl::uniformData, 27
 arraySize, 27
 glType, 27
 location, 27

 egl::Buffer, 28
 _check, 29
 _delete, 29
 _flags, 34
 _id, 34
 _mapUsage, 34
 _mapped, 34
 _type, 34
 ~Buffer, 29
 bind, 29
 Buffer, 29
 getSubData, 29
 getType, 30
 map, 30
 operator=, 31
 setData, 31
 setStorage, 32
 setSubData, 33
 size, 33
 unmap, 33

 egl::Program, 34
 _check, 41
 _delete, 41
 _ensure, 41
 _getError, 41
 _id, 63
 _linked, 63
 _queryUniformData, 41
 _setupUniform, 41
 _uniformData, 64
 _uniformIndexMap, 64
 _uniformLocationIndexMap, 64
 ~Program, 41
 attach, 42
 attached, 42
 bind, 43
 detach, 43
 getUniform, 43–51
 getUniformLocation, 51
 link, 52
 linked, 52
 operator=, 52, 53

operator[], 53, 54
Program, 41
reset, 55
setUniform, 55–63
unbind, 63
egl::Program::UniformProxy, 74
 operator float, 76
 operator glm::ivec2, 76
 operator glm::ivec3, 76
 operator glm::ivec4, 76
 operator glm::mat2, 76
 operator glm::mat2x3, 76
 operator glm::mat2x4, 76
 operator glm::mat3, 76
 operator glm::mat3x2, 76
 operator glm::mat3x4, 76
 operator glm::mat4, 77
 operator glm::mat4x2, 77
 operator glm::mat4x3, 77
 operator glm::uvec2, 77
 operator glm::uvec3, 77
 operator glm::uvec4, 77
 operator glm::vec2, 77
 operator glm::vec3, 77
 operator glm::vec4, 77
 operator int, 77
 operator unsigned int, 78
 operator=, 78–80
 UniformProxy, 75
egl::Program::UniformProxyConst, 80
 operator float, 82
 operator glm::ivec2, 82
 operator glm::ivec3, 82
 operator glm::ivec4, 82
 operator glm::mat2, 82
 operator glm::mat2x3, 82
 operator glm::mat2x4, 82
 operator glm::mat3, 82
 operator glm::mat3x2, 82
 operator glm::mat3x4, 82
 operator glm::mat4, 82
 operator glm::mat4x2, 83
 operator glm::mat4x3, 83
 operator glm::uvec2, 83
 operator glm::uvec3, 83
 operator glm::uvec4, 83
 operator glm::vec2, 83
 operator glm::vec3, 83
 operator glm::vec4, 83
 operator int, 83
 operator unsigned int, 83
 operator=, 84
 UniformProxyConst, 81
egl::ProgramLinkError, 64
 ProgramLinkError, 65
egl::ProgramValidateError, 65
 ProgramValidateError, 66
egl::Shader, 66
 __check, 70
 __compiled, 73
 __delete, 70
 __ensure, 70
 __getError, 70
 __id, 73
 __type, 73
 ~Shader, 69
 compile, 70, 71
 compiled, 72
 getType, 72
 operator=, 72
 Program, 73
 reset, 72
 Shader, 67–69
egl::ShaderCompileError, 73
 ShaderCompileError, 74
egl::VertexBuffer, 84
egl::WindowContext, 84
 ~WindowContext, 86
 operator=, 86
 run, 86
 shouldClose, 86
 swapBuffers, 87
 useContext, 87
 window, 87
 WindowContext, 85, 86
ElementArray
 egl, 18
FlushExplicit
 egl, 19
Fragment
 egl, 20
Geometry
 egl, 20
getSubData
 egl::Buffer, 29
getType
 egl::Buffer, 30
 egl::Shader, 72
getUniform
 egl::Program, 43–51
getUniformLocation
 egl::Program, 51
GL_CALL
 debug.h, 92
glCheckError
 egl, 20
glErrorString
 egl, 20
glType
 egl::_uniformData, 27
inc Directory Reference, 13
inc/EGL Directory Reference, 13
inc/EGL/buffer.h, 89, 90
inc/EGL/debug.h, 92, 93

inc/EGL/mainpage.md, 93
 inc/EGL/program.h, 93, 94
 inc/EGL/shader.h, 98, 99
 inc/EGL/vertexBuffer.h, 100
 inc/EGL/windowContext.h, 101
 initGLFW
 egl, 21
 Introduction, 1
 InvalidateBuffer
 egl, 19
 InvalidRange
 egl, 19
 link
 egl::Program, 52
 linked
 egl::Program, 52
 location
 egl::_uniformData, 27
 map
 egl::Buffer, 30
 MapCoherent
 egl, 17
 MapPersistent
 egl, 17
 MapRead
 egl, 17
 MapUsage
 egl, 19
 MapWrite
 egl, 17
 None
 egl, 17, 19
 operator float
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::ivec2
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::ivec3
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::ivec4
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::mat2
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::mat2x3
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::mat2x4
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
 operator glm::mat3
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
operator glm::mat3x2
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
operator glm::mat3x4
 egl::Program::UniformProxy, 76
 egl::Program::UniformProxyConst, 82
operator glm::mat4
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 82
operator glm::mat4x2
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::mat4x3
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::uvec2
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::uvec3
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::uvec4
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::vec2
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::vec3
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator glm::vec4
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator int
 egl::Program::UniformProxy, 77
 egl::Program::UniformProxyConst, 83
operator unsigned int
 egl::Program::UniformProxy, 78
 egl::Program::UniformProxyConst, 83
operator=
 egl::Buffer, 31
 egl::Program, 52, 53
 egl::Program::UniformProxy, 78–80
 egl::Program::UniformProxyConst, 84
 egl::Shader, 72
 egl::WindowContext, 86
operator&
 egl, 21
operator[]
 egl::Program, 53, 54
operator |
 egl, 21
operator |=
 egl, 22
Persistent
 egl, 19
PixelPack

egl, 18
PixelUnpack
 egl, 18
pollEvents
 egl, 22
Program
 egl::Program, 41
 egl::Shader, 73
ProgramLinkError
 egl::ProgramLinkError, 65
ProgramValidateError
 egl::ProgramValidateError, 66

Query
 egl, 18

Read
 egl, 19
reset
 egl::Program, 55
 egl::Shader, 72
run
 egl::WindowContext, 86

setData
 egl::Buffer, 31
setStorage
 egl::Buffer, 32
setSubData
 egl::Buffer, 33
setUniform
 egl::Program, 55–63
Shader
 egl::Shader, 67–69
ShaderCompileError
 egl::ShaderCompileError, 74
ShaderStorage
 egl, 18
ShaderType
 egl, 19
shaderTypeToString
 egl, 22
shouldClose
 egl::WindowContext, 86
size
 egl::Buffer, 33
StaticCopy
 egl, 19
StaticDraw
 egl, 19
StaticRead
 egl, 19
StreamCopy
 egl, 19
StreamDraw
 egl, 19
StreamRead
 egl, 19
swapBuffers
 egl::WindowContext, 87
terminateGLFW
 egl, 22
TessControl
 egl, 20
TessEvaluation
 egl, 20
Texture
 egl, 18
toGLenum
 egl, 23
TransformFeedback
 egl, 18

unbind
 egl::Program, 63
Uniform
 egl, 18
UniformData
 egl, 17
UniformProxy
 egl::Program::UniformProxy, 75
UniformProxyConst
 egl::Program::UniformProxyConst, 81
unmap
 egl::Buffer, 33
Unsynchronized
 egl, 19
useContext
 egl::WindowContext, 87

validateBufferFlag
 egl, 24
validateMapUsage
 egl, 24
Vertex
 egl, 20

window
 egl::WindowContext, 87
WindowContext
 egl::WindowContext, 85, 86
Write
 egl, 19