

OpenGL Abstraction

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/GLA Directory Reference	13
7.2 inc Directory Reference	13
8 Namespace Documentation	15
8.1 gla Namespace Reference	15
8.1.1 Enumeration Type Documentation	17
8.1.1.1 BufferFlag	17
8.1.1.2 BufferType	17
8.1.1.3 BufferUsage	18
8.1.1.4 MapUsage	19
8.1.1.5 ShaderType	20
8.1.1.6 VertexAttribInterp	20
8.1.1.7 VertexAttribType	20
8.1.2 Function Documentation	21
8.1.2.1 glCheckError()	21
8.1.2.2 glErrorString()	21
8.1.2.3 initGLFW()	22
8.1.2.4 operator&() [1/2]	22
8.1.2.5 operator&() [2/2]	22
8.1.2.6 operator" () [1/2]	22
8.1.2.7 operator" () [2/2]	22
8.1.2.8 operator" =() [1/2]	22
8.1.2.9 operator" =() [2/2]	23
8.1.2.10 pollEvents()	23
8.1.2.11 shaderTypeToString()	23
8.1.2.12 terminateGLFW()	23

8.1.2.13 toGLenum() [1/6]	23
8.1.2.14 toGLenum() [2/6]	24
8.1.2.15 toGLenum() [3/6]	24
8.1.2.16 toGLenum() [4/6]	24
8.1.2.17 toGLenum() [5/6]	24
8.1.2.18 toGLenum() [6/6]	25
8.1.2.19 typeToBytes()	25
8.1.2.20 validateBufferFlag()	25
8.1.2.21 validateMapUsage()	26
8.1.2.22 validateTypeInterpretation()	26
9 Data Structure Documentation	27
9.1 gla::Buffer Class Reference	27
9.1.1 Constructor & Destructor Documentation	28
9.1.1.1 Buffer() [1/4]	28
9.1.1.2 Buffer() [2/4]	28
9.1.1.3 Buffer() [3/4]	28
9.1.1.4 Buffer() [4/4]	28
9.1.1.5 ~Buffer()	29
9.1.2 Member Function Documentation	29
9.1.2.1 _check()	29
9.1.2.2 _delete()	29
9.1.2.3 bind()	29
9.1.2.4 getSubData()	29
9.1.2.5 getType()	30
9.1.2.6 map()	30
9.1.2.7 operator=() [1/2]	30
9.1.2.8 operator=() [2/2]	30
9.1.2.9 setData() [1/2]	31
9.1.2.10 setData() [2/2]	31
9.1.2.11 setStorage() [1/2]	31
9.1.2.12 setStorage() [2/2]	32
9.1.2.13 setSubData()	32
9.1.2.14 size()	33
9.1.2.15 unmap()	33
9.1.3 Field Documentation	33
9.1.3.1 _flags	33
9.1.3.2 _id	33
9.1.3.3 _mapped	33
9.1.3.4 _mapUsage	33
9.1.3.5 _type	33
9.2 gla::Program Class Reference	34

9.2.1 Detailed Description	40
9.2.2 Constructor & Destructor Documentation	40
9.2.2.1 Program() [1/3]	40
9.2.2.2 Program() [2/3]	40
9.2.2.3 Program() [3/3]	40
9.2.2.4 ~Program()	40
9.2.3 Member Function Documentation	40
9.2.3.1 _check()	40
9.2.3.2 _delete()	40
9.2.3.3 _ensure()	41
9.2.3.4 _getError()	41
9.2.3.5 _queryUniformData()	41
9.2.3.6 _setupUniform()	41
9.2.3.7 attach()	41
9.2.3.8 attached()	42
9.2.3.9 bind()	42
9.2.3.10 detach()	42
9.2.3.11 getUniform() [1/42]	43
9.2.3.12 getUniform() [2/42]	43
9.2.3.13 getUniform() [3/42]	43
9.2.3.14 getUniform() [4/42]	44
9.2.3.15 getUniform() [5/42]	44
9.2.3.16 getUniform() [6/42]	44
9.2.3.17 getUniform() [7/42]	44
9.2.3.18 getUniform() [8/42]	44
9.2.3.19 getUniform() [9/42]	44
9.2.3.20 getUniform() [10/42]	45
9.2.3.21 getUniform() [11/42]	45
9.2.3.22 getUniform() [12/42]	45
9.2.3.23 getUniform() [13/42]	45
9.2.3.24 getUniform() [14/42]	45
9.2.3.25 getUniform() [15/42]	45
9.2.3.26 getUniform() [16/42]	46
9.2.3.27 getUniform() [17/42]	46
9.2.3.28 getUniform() [18/42]	46
9.2.3.29 getUniform() [19/42]	46
9.2.3.30 getUniform() [20/42]	46
9.2.3.31 getUniform() [21/42]	46
9.2.3.32 getUniform() [22/42]	47
9.2.3.33 getUniform() [23/42]	47
9.2.3.34 getUniform() [24/42]	47
9.2.3.35 getUniform() [25/42]	47

9.2.3.36 getUniform() [26/42]	48
9.2.3.37 getUniform() [27/42]	48
9.2.3.38 getUniform() [28/42]	48
9.2.3.39 getUniform() [29/42]	48
9.2.3.40 getUniform() [30/42]	48
9.2.3.41 getUniform() [31/42]	48
9.2.3.42 getUniform() [32/42]	49
9.2.3.43 getUniform() [33/42]	49
9.2.3.44 getUniform() [34/42]	49
9.2.3.45 getUniform() [35/42]	49
9.2.3.46 getUniform() [36/42]	49
9.2.3.47 getUniform() [37/42]	49
9.2.3.48 getUniform() [38/42]	50
9.2.3.49 getUniform() [39/42]	50
9.2.3.50 getUniform() [40/42]	50
9.2.3.51 getUniform() [41/42]	50
9.2.3.52 getUniform() [42/42]	50
9.2.3.53 getUniformLocation()	51
9.2.3.54 link()	51
9.2.3.55 linked()	51
9.2.3.56 operator=() [1/2]	52
9.2.3.57 operator=() [2/2]	52
9.2.3.58 operator[]() [1/4]	52
9.2.3.59 operator[]() [2/4]	52
9.2.3.60 operator[]() [3/4]	53
9.2.3.61 operator[]() [4/4]	53
9.2.3.62 reset()	54
9.2.3.63 setUniform() [1/42]	54
9.2.3.64 setUniform() [2/42]	54
9.2.3.65 setUniform() [3/42]	54
9.2.3.66 setUniform() [4/42]	55
9.2.3.67 setUniform() [5/42]	55
9.2.3.68 setUniform() [6/42]	55
9.2.3.69 setUniform() [7/42]	55
9.2.3.70 setUniform() [8/42]	55
9.2.3.71 setUniform() [9/42]	55
9.2.3.72 setUniform() [10/42]	56
9.2.3.73 setUniform() [11/42]	56
9.2.3.74 setUniform() [12/42]	56
9.2.3.75 setUniform() [13/42]	56
9.2.3.76 setUniform() [14/42]	56
9.2.3.77 setUniform() [15/42]	56

9.2.3.78 setUniform() [16/42]	57
9.2.3.79 setUniform() [17/42]	57
9.2.3.80 setUniform() [18/42]	57
9.2.3.81 setUniform() [19/42]	57
9.2.3.82 setUniform() [20/42]	58
9.2.3.83 setUniform() [21/42]	58
9.2.3.84 setUniform() [22/42]	58
9.2.3.85 setUniform() [23/42]	58
9.2.3.86 setUniform() [24/42]	59
9.2.3.87 setUniform() [25/42]	59
9.2.3.88 setUniform() [26/42]	59
9.2.3.89 setUniform() [27/42]	59
9.2.3.90 setUniform() [28/42]	59
9.2.3.91 setUniform() [29/42]	59
9.2.3.92 setUniform() [30/42]	60
9.2.3.93 setUniform() [31/42]	60
9.2.3.94 setUniform() [32/42]	60
9.2.3.95 setUniform() [33/42]	60
9.2.3.96 setUniform() [34/42]	60
9.2.3.97 setUniform() [35/42]	60
9.2.3.98 setUniform() [36/42]	61
9.2.3.99 setUniform() [37/42]	61
9.2.3.100 setUniform() [38/42]	61
9.2.3.101 setUniform() [39/42]	61
9.2.3.102 setUniform() [40/42]	61
9.2.3.103 setUniform() [41/42]	62
9.2.3.104 setUniform() [42/42]	62
9.2.3.105 unbind()	62
9.2.4 Field Documentation	62
9.2.4.1 _id	62
9.2.4.2 _linked	63
9.2.4.3 _uniformData	63
9.2.4.4 _uniformIndexMap	63
9.2.4.5 _uniformLocationIndexMap	63
9.3 gla::ProgramLinkError Class Reference	63
9.3.1 Detailed Description	63
9.3.2 Constructor & Destructor Documentation	64
9.3.2.1 ProgramLinkError()	64
9.4 gla::ProgramValidateError Class Reference	64
9.4.1 Detailed Description	64
9.4.2 Constructor & Destructor Documentation	65
9.4.2.1 ProgramValidateError()	65

9.5 <code>gla::Shader</code> Class Reference	65
9.5.1 Detailed Description	66
9.5.2 Constructor & Destructor Documentation	66
9.5.2.1 <code>Shader()</code> [1/8]	66
9.5.2.2 <code>Shader()</code> [2/8]	67
9.5.2.3 <code>Shader()</code> [3/8]	67
9.5.2.4 <code>Shader()</code> [4/8]	67
9.5.2.5 <code>Shader()</code> [5/8]	68
9.5.2.6 <code>Shader()</code> [6/8]	68
9.5.2.7 <code>Shader()</code> [7/8]	68
9.5.2.8 <code>Shader()</code> [8/8]	68
9.5.2.9 <code>~Shader()</code>	69
9.5.3 Member Function Documentation	69
9.5.3.1 <code>_check()</code>	69
9.5.3.2 <code>_delete()</code>	69
9.5.3.3 <code>_ensure()</code>	69
9.5.3.4 <code>_getError()</code>	69
9.5.3.5 <code>compile()</code> [1/4]	69
9.5.3.6 <code>compile()</code> [2/4]	70
9.5.3.7 <code>compile()</code> [3/4]	70
9.5.3.8 <code>compile()</code> [4/4]	71
9.5.3.9 <code>compiled()</code>	71
9.5.3.10 <code>getType()</code>	71
9.5.3.11 <code>operator=()</code> [1/2]	71
9.5.3.12 <code>operator=()</code> [2/2]	71
9.5.3.13 <code>reset()</code>	72
9.5.4 Field Documentation	72
9.5.4.1 <code>_compiled</code>	72
9.5.4.2 <code>_id</code>	72
9.5.4.3 <code>_type</code>	72
9.5.4.4 <code>Program</code>	72
9.6 <code>gla::ShaderCompileError</code> Class Reference	72
9.6.1 Detailed Description	73
9.6.2 Constructor & Destructor Documentation	73
9.6.2.1 <code>ShaderCompileError()</code>	73
9.7 <code>gla::UniformData</code> Struct Reference	73
9.7.1 Field Documentation	73
9.7.1.1 <code>arraySize</code>	73
9.7.1.2 <code>glType</code>	74
9.7.1.3 <code>location</code>	74
9.8 <code>gla::Program::UniformProxy</code> 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]	75
9.8.2 Member Function Documentation	75
9.8.2.1 operator float()	75
9.8.2.2 operator glm::ivec2()	75
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()	76
9.8.2.11 operator glm::mat4()	76
9.8.2.12 operator glm::mat4x2()	76
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()	77
9.8.2.21 operator unsigned int()	77
9.8.2.22 operator=() [1/23]	77
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]	78
9.8.2.30 operator=() [9/23]	78
9.8.2.31 operator=() [10/23]	78
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]	79

9.8.2.39 operator=() [18/23]	79
9.8.2.40 operator=() [19/23]	79
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 gla::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	81
9.9.2.1 operator float()	81
9.9.2.2 operator glm::ivec2()	81
9.9.2.3 operator glm::ivec3()	81
9.9.2.4 operator glm::ivec4()	81
9.9.2.5 operator glm::mat2()	81
9.9.2.6 operator glm::mat2x3()	81
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()	82
9.9.2.12 operator glm::mat4x2()	82
9.9.2.13 operator glm::mat4x3()	82
9.9.2.14 operator glm::uvec2()	82
9.9.2.15 operator glm::uvec3()	82
9.9.2.16 operator glm::uvec4()	82
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()	83
9.9.2.22 operator=() [1/2]	83
9.9.2.23 operator=() [2/2]	83
9.10 gla::VertexArray Class Reference	83
9.10.1 Detailed Description	85
9.10.2 Constructor & Destructor Documentation	85
9.10.2.1 VertexArray() [1/3]	85
9.10.2.2 VertexArray() [2/3]	85
9.10.2.3 VertexArray() [3/3]	85
9.10.3 Member Function Documentation	85

9.10.3.1 operator=() [1/2]	85
9.10.3.2 operator=() [2/2]	86
9.10.3.3 setAttributes()	86
9.11 gla::VertexAttribute Struct Reference	86
9.11.1 Detailed Description	87
9.11.2 Field Documentation	87
9.11.2.1 index	87
9.11.2.2 interp	87
9.11.2.3 normalized	87
9.11.2.4 numComponents	87
9.11.2.5 offset	88
9.11.2.6 type	88
9.12 gla::WindowContext Class Reference	88
9.12.1 Detailed Description	89
9.12.2 Constructor & Destructor Documentation	89
9.12.2.1 WindowContext() [1/3]	89
9.12.2.2 WindowContext() [2/3]	89
9.12.2.3 WindowContext() [3/3]	89
9.12.2.4 ~WindowContext()	90
9.12.3 Member Function Documentation	90
9.12.3.1 operator=() [1/2]	90
9.12.3.2 operator=() [2/2]	90
9.12.3.3 run()	90
9.12.3.4 shouldClose()	90
9.12.3.5 swapBuffers()	91
9.12.3.6 useContext()	91
9.12.4 Field Documentation	91
9.12.4.1 window	91
10 File Documentation	93
10.1 inc/GLA/buffer.h File Reference	93
10.2 buffer.h	94
10.3 inc/GLA/debug.h File Reference	96
10.3.1 Macro Definition Documentation	96
10.3.1.1 DEBUG_ONLY	96
10.3.1.2 GL_CALL	97
10.4 debug.h	97
10.5 inc/GLA/mainpage.md File Reference	97
10.6 inc/GLA/program.h File Reference	97
10.7 program.h	98
10.8 inc/GLA/shader.h File Reference	102
10.9 shader.h	103

10.10 inc/GLA/vertexArray.h File Reference	104
10.11 vertexArray.h	105
10.12 inc/GLA/windowContext.h File Reference	106
10.13 windowContext.h	106
Index	107

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 `gla::WindowContext` and implement your logic in the provided virtual run function.

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

Inside of `gla::WindowContext::run` any OpenGL Abstraction functionality can be used safely.

Chapter 2

Directory Hierarchy

2.1 Directories

GLA	13
buffer.h	93
debug.h	96
program.h	97
shader.h	102
vertexArray.h	104
windowContext.h	106
inc	13
GLA	13
buffer.h	93
debug.h	96
program.h	97
shader.h	102
vertexArray.h	104
windowContext.h	106

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

gla	15
-------------------------------	----

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

gla::Buffer	27
gla::VertexArray	83
gla::Program	34
std::runtime_error	
gla::ProgramLinkError	63
gla::ProgramValidateError	64
gla::ShaderCompileError	72
gla::Shader	65
gla::UniformData	73
gla::Program::UniformProxy	74
gla::Program::UniformProxyConst	80
gla::VertexAttribute	86
gla::WindowContext	88

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

gla::Buffer	27
gla::Program	
Program class to abstract OpenGL Shader Programs	34
gla::ProgramLinkError	
Exception thrown when Program linking fails	63
gla::ProgramValidateError	
Exception thrown when Program validation fails, mainly used in debug builds	64
gla::Shader	
Shader class to abstract OpenGL Shaders	65
gla::ShaderCompileError	
Exception thrown when Shader compilation fails	72
gla::UniformData	73
gla::Program::UniformProxy	74
gla::Program::UniformProxyConst	80
gla::VertexArray	
VertexArray class to abstract the OpenGL vertex array	83
gla::VertexAttribute	
Defines a vertex attribute for the gla::VertexArray	86
gla::WindowContext	
An abstract class to contain an window based application	88

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

inc/GLA/buffer.h	93
inc/GLA/debug.h	96
inc/GLA/program.h	97
inc/GLA/shader.h	102
inc/GLA/vertexArray.h	104
inc/GLA/windowContext.h	106

Chapter 7

Directory Documentation

7.1 inc/GLA Directory Reference

Files

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

7.2 inc Directory Reference

Directories

- directory [GLA](#)

Chapter 8

Namespace Documentation

8.1 gla 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.
- struct [VertexAttribute](#)
Defines a vertex attribute for the [gla::VertexArray](#).
- class [VertexArray](#)
[VertexArray](#) class to abstract the OpenGL vertex array.
- class [WindowContext](#)
An abstract class to contain an window based application.

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.

- enum class [VertexAttribType](#) {
[Byte](#) , [UnsignedByte](#) , [Short](#) , [UnsignedShort](#) ,
[Int](#) , [UnsignedInt](#) , [HalfFloat](#) , [Float](#) ,
[Double](#) , [Fixed](#) }
- enum class [VertexAttribInterp](#) { [Float](#) , [Integer](#) }

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.
- unsigned int [toGLenum](#) ([VertexAttribType](#) type)
Converts a *VertexAttribType* into a *GLenum*.
- bool [validateTypeInterpretation](#) ([VertexAttribType](#) type, [VertexAttribInterp](#) interp, std::string &error)
Checks if the type and interpretation combination is valid.

- int [typeToBytes](#) ([VertexAttribType](#) type)
Gets the size of the given type in bytes.
- bool [initGLFW](#) ()
Initialize GLFW.
- void [terminateGLFW](#) ()
Terminates GLFW.
- void [pollEvents](#) ()
Polls GLFW events.

8.1.1 Enumeration Type Documentation

8.1.1.1 BufferFlag

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

Enum falgs 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 <code>glBufferSubData</code> .
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 thar 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 <code>glMapBufferRange</code> .
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.

8.1.1.2 BufferType

```
enum class gla::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.

8.1.1.3 BufferUsage

```
enum class gla::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	

8.1.1.4 MapUsage

```
enum class gla::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 fassion 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>glFlush↵MappedBufferRange</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> .

8.1.1.5 ShaderType

```
enum class glsl::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.

8.1.1.6 VertexAttribInterp

```
enum class glsl::VertexAttribInterp [strong]
```

Enumerator

Float	Interprets the vertex attribute as a float.
Integer	Interprets the vertex attribute as an integer.

8.1.1.7 VertexAttribType

```
enum class glsl::VertexAttribType [strong]
```

Enumerator

Byte	GL_BYTE.
UnsignedByte	GL_UNSIGNED_BYTE.
Short	GL_SHORT.
UnsignedShort	GL_UNSIGNED_SHORT.
Int	GL_INT.
UnsignedInt	GL_UNSIGNED_INT.
HalfFloat	GL_HALF_FLOAT.
Float	GL_FLOAT.
Double	GL_DOUBLE.
Fixed	GL_FIXED.

8.1.2 Function Documentation

8.1.2.1 glCheckError()

```
void gla::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

8.1.2.2 glErrorString()

```
const char * gla::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

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

Returns

c-style NULL terminated string

8.1.2.3 initGLFW()

```
bool gla::initGLFW ()
```

Initialize GLFW.

Sets up error callbacks and calls glfwInit.

Note

[gla::initGLFW](#) is not thread safe and may only be called from the main thread.

Returns

true if GLFW has been successfully initialized.

8.1.2.4 operator&() [1/2]

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

8.1.2.5 operator&() [2/2]

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

8.1.2.6 operator" | () [1/2]

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

8.1.2.7 operator" | () [2/2]

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

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

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

8.1.2.9 operator" |=([2/2]

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

8.1.2.10 pollEvents()

```
void gla::pollEvents ()
```

Polls GLFW events.

Note

[gla::pollEvents](#) is not thread safe and may only be called from the main thread.

8.1.2.11 shaderTypeToString()

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

Converts the [ShaderType](#) enum into a std::string.

Parameters

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

Returns

The name of the given [ShaderType](#) (if unknown "INVALID" is returned).

8.1.2.12 terminateGLFW()

```
void gla::terminateGLFW ()
```

Terminates GLFW.

Note

[gla::terminateGLFW](#) is not thread safe and may only be called from the main thread.

8.1.2.13 toGLenum() [1/6]

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

Converts a [BufferFlag](#) enum into a GLenum.

8.1.2.14 toGLenum() [2/6]

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

Converts a [BufferType](#) enum into a GLenum.

Exceptions

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

8.1.2.15 toGLenum() [3/6]

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

Converts a [BufferUsage](#) enum into a GLenum.

Exceptions

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

8.1.2.16 toGLenum() [4/6]

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

Converts a [MapUsage](#) enum into a GLenum.

8.1.2.17 toGLenum() [5/6]

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

Converts the [ShaderType](#) enum into an OpenGL enum.

Exceptions

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

Parameters

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

Returns

The resulting OpenGL enum.

8.1.2.18 toGLenum() [6/6]

```
unsigned int gla::toGLenum (  
    VertexAttribType type)
```

Converts a [VertexAttribType](#) into a GLenum.

Exceptions

<code>std::invalid_argument</code>	If the given VertexAttribType is invalid
------------------------------------	--

8.1.2.19 typeToBytes()

```
int gla::typeToBytes (  
    VertexAttribType type)
```

Gets the size of the given type in bytes.

Exceptions

<code>std::invalid_argument</code>	If the given VertexAttribType is invalid
------------------------------------	--

8.1.2.20 validateBufferFlag()

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

Checks if the given [BufferFlag](#) flag combination is valid.

Parameters

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

Returns

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

8.1.2.21 validateMapUsage()

```
bool gla::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

8.1.2.22 validateTypeInterpretation()

```
bool gla::validateTypeInterpretation (
    VertexAttribType type,
    VertexAttribInterp interp,
    std::string & error)
```

Checks if the type and interpretation combination is valid.

Parameters

<i>error</i>	The error string output
--------------	-------------------------

Returns

true if it is valid, false otherwise

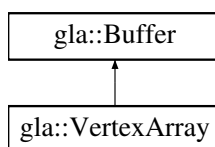
Chapter 9

Data Structure Documentation

9.1 gla::Buffer Class Reference

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

Inheritance diagram for gla::Buffer:



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](#)

9.1.1 Constructor & Destructor Documentation

9.1.1.1 [Buffer\(\)](#) [1/4]

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

9.1.1.2 [Buffer\(\)](#) [2/4]

```
gla::Buffer::Buffer (
    BufferType type)
```

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

9.1.1.3 [Buffer\(\)](#) [3/4]

```
gla::Buffer::Buffer (
    Buffer && other)
```

9.1.1.4 [Buffer\(\)](#) [4/4]

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


9.1.1.5 ~Buffer()

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

9.1.2 Member Function Documentation

9.1.2.1 _check()

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

9.1.2.2 _delete()

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

9.1.2.3 bind()

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

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

9.1.2.4 getSubData()

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

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

Exceptions

<i>std::runtime_error</i>	If offset is negativ
<i>std::runtime_error</i>	If size is negativ
<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 get in bytes
<i>size</i>	The size of the subset to get in bytes
<i>data</i>	The destination of the data of the subset (must be at least size bytes big)

9.1.2.5 getType()

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

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

9.1.2.6 map()

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

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

Exceptions

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

Parameters

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

Returns

A pointer to the beginning of the mapped range

9.1.2.7 operator=() [1/2]

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

9.1.2.8 operator=() [2/2]

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

9.1.2.9 setData() [1/2]

```
void gla::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

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

9.1.2.10 setData() [2/2]

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

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

Parameters

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

9.1.2.11 setStorage() [1/2]

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

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

Exceptions

<code>std::runtime_error</code>	If size is not greater than 0
<code>std::runtime_error</code>	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

9.1.2.12 `setStorage()` [2/2]

```
template<typename T>
void gla::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

9.1.2.13 `setSubData()`

```
void gla::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 negativ
<i>std::runtime_error</i>	If size is negativ
<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)
-------------	--

9.1.2.14 size()

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

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

9.1.2.15 unmap()

```
void gla::Buffer::unmap ()
```

Unmaps the [Buffer](#).

Exceptions

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

9.1.3 Field Documentation

9.1.3.1 _flags

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

9.1.3.2 _id

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

9.1.3.3 _mapped

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

9.1.3.4 _mapUsage

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

9.1.3.5 _type

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

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

- inc/GLA/[buffer.h](#)

9.2 gla::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)

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::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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (const std::string &name, 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` (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` = {}

9.2.1 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\(\)](#).

9.2.2 Constructor & Destructor Documentation

9.2.2.1 Program() [1/3]

```
gla::Program::Program ()
```

Construct an empty [Program](#) object.

9.2.2.2 Program() [2/3]

```
gla::Program::Program (  
    Program && other)
```

9.2.2.3 Program() [3/3]

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

9.2.2.4 ~Program()

```
gla::Program::~~Program ()
```

9.2.3 Member Function Documentation

9.2.3.1 _check()

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

9.2.3.2 _delete()

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

9.2.3.3 _ensure()

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

9.2.3.4 _getError()

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

9.2.3.5 _queryUniformData()

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

9.2.3.6 _setupUniform()

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

9.2.3.7 attach()

```
void gla::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 (reset() is recommended to return to a valid state)

Parameters

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

9.2.3.8 attached()

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

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

Parameters

<i>shader</i>	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.

9.2.3.9 bind()

```
void gla::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.

9.2.3.10 detach()

```
void gla::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 .
---------------	---

9.2.3.11 getUniform() [1/42]

```
void gla::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.

9.2.3.12 getUniform() [2/42]

```
void gla::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.

9.2.3.13 getUniform() [3/42]

```
void gla::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.

9.2.3.14 getUniform() [4/42]

```
void gla::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.

9.2.3.15 getUniform() [5/42]

```
void gla::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.

9.2.3.16 getUniform() [6/42]

```
void gla::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.

9.2.3.17 getUniform() [7/42]

```
void gla::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.

9.2.3.18 getUniform() [8/42]

```
void gla::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.

9.2.3.19 getUniform() [9/42]

```
void gla::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.

9.2.3.20 getUniform() [10/42]

```
void gla::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.

9.2.3.21 getUniform() [11/42]

```
void gla::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.

9.2.3.22 getUniform() [12/42]

```
void gla::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.

9.2.3.23 getUniform() [13/42]

```
void gla::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.

9.2.3.24 getUniform() [14/42]

```
void gla::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.

9.2.3.25 getUniform() [15/42]

```
void gla::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.

9.2.3.26 getUniform() [16/42]

```
void gla::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.

9.2.3.27 getUniform() [17/42]

```
void gla::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.

9.2.3.28 getUniform() [18/42]

```
void gla::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.

9.2.3.29 getUniform() [19/42]

```
void gla::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.

9.2.3.30 getUniform() [20/42]

```
void gla::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.

9.2.3.31 getUniform() [21/42]

```
void gla::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.

9.2.3.32 getUniform() [22/42]

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

Gets a uniform at the given location.

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

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

9.2.3.33 getUniform() [23/42]

```
void gla::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.

9.2.3.34 getUniform() [24/42]

```
void gla::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.

9.2.3.35 getUniform() [25/42]

```
void gla::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.

9.2.3.36 glUniform() [26/42]

```
void gla::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.

9.2.3.37 glUniform() [27/42]

```
void gla::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.

9.2.3.38 glUniform() [28/42]

```
void gla::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.

9.2.3.39 glUniform() [29/42]

```
void gla::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.

9.2.3.40 glUniform() [30/42]

```
void gla::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.

9.2.3.41 glUniform() [31/42]

```
void gla::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.

9.2.3.42 getUniform() [32/42]

```
void gla::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.

9.2.3.43 getUniform() [33/42]

```
void gla::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.

9.2.3.44 getUniform() [34/42]

```
void gla::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.

9.2.3.45 getUniform() [35/42]

```
void gla::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.

9.2.3.46 getUniform() [36/42]

```
void gla::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.

9.2.3.47 getUniform() [37/42]

```
void gla::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.

9.2.3.48 glUniform() [38/42]

```
void gla::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.

9.2.3.49 glUniform() [39/42]

```
void gla::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.

9.2.3.50 glUniform() [40/42]

```
void gla::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.

9.2.3.51 glUniform() [41/42]

```
void gla::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.

9.2.3.52 glUniform() [42/42]

```
void gla::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.

9.2.3.53 getUniformLocation()

```
int gla::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

9.2.3.54 link()

```
void gla::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>gla::ProgramLinkError</i>	If the Program fails to link. The Program remains in a valid state afterwards.
<i>gla::ProgramValidateError</i>	If the Program fails to validate. This only occurs when DEBUG_BUILD is defined else the validity is not checked.

9.2.3.55 linked()

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

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

Returns

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

9.2.3.56 operator=() [1/2]

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

9.2.3.57 operator=() [2/2]

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

9.2.3.58 operator[]() [1/4]

```
UniformProxy gla::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 [gla::Program::setUniform](#).

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

Warning

Setting a uniform binds the 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 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
-------------	------------------

9.2.3.59 operator[]() [2/4]

```
UniformProxyConst gla::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 [gla::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
-------------	------------------

9.2.3.60 operator[]() [3/4]

```
UniformProxy gla::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 [gla::Program::setUniform](#).

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

Warning

Setting a uniform binds the program.

Parameters

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

9.2.3.61 operator[]() [4/4]

```
UniformProxyConst gla::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 [gla::Program::getUniform](#).

Parameters

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

9.2.3.62 reset()

```
void gla::Program::reset ()
```

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

Exceptions

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

9.2.3.63 setUniform() [1/42]

```
void gla::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.

9.2.3.64 setUniform() [2/42]

```
void gla::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.

9.2.3.65 setUniform() [3/42]

```
void gla::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.

9.2.3.66 glUniform() [4/42]

```
void gla::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.

9.2.3.67 glUniform() [5/42]

```
void gla::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.

9.2.3.68 glUniform() [6/42]

```
void gla::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.

9.2.3.69 glUniform() [7/42]

```
void gla::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.

9.2.3.70 glUniform() [8/42]

```
void gla::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.

9.2.3.71 glUniform() [9/42]

```
void gla::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.

9.2.3.72 glUniform() [10/42]

```
void gla::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.

9.2.3.73 glUniform() [11/42]

```
void gla::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.

9.2.3.74 glUniform() [12/42]

```
void gla::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.

9.2.3.75 glUniform() [13/42]

```
void gla::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.

9.2.3.76 glUniform() [14/42]

```
void gla::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.

9.2.3.77 glUniform() [15/42]

```
void gla::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.

9.2.3.78 setUniform() [16/42]

```
void gla::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.

9.2.3.79 setUniform() [17/42]

```
void gla::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.

9.2.3.80 setUniform() [18/42]

```
void gla::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.

9.2.3.81 setUniform() [19/42]

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

Sets 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 set a GLSL array
<i>std::runtime_error</i>	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.

9.2.3.82 glUniform() [20/42]

```
void gla::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.

9.2.3.83 glUniform() [21/42]

```
void gla::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.

9.2.3.84 glUniform() [22/42]

```
void gla::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.

9.2.3.85 glUniform() [23/42]

```
void gla::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.

9.2.3.86 glUniform() [24/42]

```
void gla::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.

9.2.3.87 glUniform() [25/42]

```
void gla::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.

9.2.3.88 glUniform() [26/42]

```
void gla::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.

9.2.3.89 glUniform() [27/42]

```
void gla::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.

9.2.3.90 glUniform() [28/42]

```
void gla::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.

9.2.3.91 glUniform() [29/42]

```
void gla::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.

9.2.3.92 setUniform() [30/42]

```
void gla::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.

9.2.3.93 setUniform() [31/42]

```
void gla::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.

9.2.3.94 setUniform() [32/42]

```
void gla::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.

9.2.3.95 setUniform() [33/42]

```
void gla::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.

9.2.3.96 setUniform() [34/42]

```
void gla::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.

9.2.3.97 setUniform() [35/42]

```
void gla::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.

9.2.3.98 setUniform() [36/42]

```
void gla::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.

9.2.3.99 setUniform() [37/42]

```
void gla::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.

9.2.3.100 setUniform() [38/42]

```
void gla::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.

9.2.3.101 setUniform() [39/42]

```
void gla::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.

9.2.3.102 setUniform() [40/42]

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

Sets a uniform at the given location.

Exceptions

<i>std::logic_error</i>	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>location</i>	The uniform location returned by getUniformLocation()
<i>data</i>	The value to assign to the uniform

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

9.2.3.103 setUniform() [41/42]

```
void gla::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.

9.2.3.104 setUniform() [42/42]

```
void gla::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.

9.2.3.105 unbind()

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

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

9.2.4 Field Documentation**9.2.4.1 _id**

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

9.2.4.2 `_linked`

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

9.2.4.3 `_uniformData`

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

9.2.4.4 `_uniformIndexMap`

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

9.2.4.5 `_uniformLocationIndexMap`

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

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

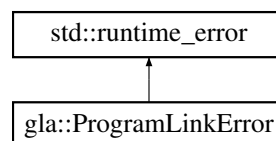
- [inc/GLA/program.h](#)

9.3 gla::ProgramLinkError Class Reference

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

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

Inheritance diagram for gla::ProgramLinkError:



Public Member Functions

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

9.3.1 Detailed Description

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

9.3.2 Constructor & Destructor Documentation

9.3.2.1 ProgramLinkError()

```
gla::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:

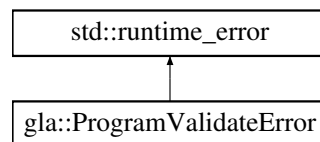
- inc/GLA/[program.h](#)

9.4 gla::ProgramValidateError Class Reference

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

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

Inheritance diagram for gla::ProgramValidateError:



Public Member Functions

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

9.4.1 Detailed Description

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

9.4.2 Constructor & Destructor Documentation

9.4.2.1 ProgramValidateError()

```
gla::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/GLA/[program.h](#)

9.5 gla::Shader Class Reference

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

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

Public Member Functions

- [Shader](#) ()=delete
- [Shader](#) ([ShaderType](#) type)
- Construct 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)
- [Shader](#) (const Shader &other)=delete
- [~Shader](#) () noexcept
- 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)
- [Shader](#) & [operator=](#) (const [Shader](#) &other)=delete

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

9.5.1 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\(\)](#).

9.5.2 Constructor & Destructor Documentation

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

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

9.5.2.2 Shader() [2/8]

```
glsl::Shader::Shader (
    ShaderType type)
```

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

Exceptions

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

9.5.2.3 Shader() [3/8]

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

Constructs and compiles a new [Shader](#) object of given type.

Exceptions

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

9.5.2.4 Shader() [4/8]

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

Constructs and compiles a new [Shader](#) object of given type.

Exceptions

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

9.5.2.5 Shader() [5/8]

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

Constructs and compiles a new [Shader](#) object of given type.

Exceptions

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

9.5.2.6 Shader() [6/8]

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

Constructs and compiles a new [Shader](#) object of given type.

Exceptions

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

9.5.2.7 Shader() [7/8]

```
gla::Shader::Shader (
    Shader && other)
```

9.5.2.8 Shader() [8/8]

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


9.5.2.9 ~Shader()

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

9.5.3 Member Function Documentation

9.5.3.1 _check()

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

9.5.3.2 _delete()

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

9.5.3.3 _ensure()

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

9.5.3.4 _getError()

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

9.5.3.5 compile() [1/4]

```
void gla::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>gla::ShaderCompileError</i>	If the Shader fails to compile.

Parameters

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

9.5.3.6 compile() [2/4]

```
void gla::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).
gla::ShaderCompileError	If the Shader fails to compile.

Parameters

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

9.5.3.7 compile() [3/4]

```
void gla::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).
gla::ShaderCompileError	If the Shader fails to compile.

Parameters

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

9.5.3.8 compile() [4/4]

```
void gla::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>gla::ShaderCompileError</i>	If the Shader fails to compile.

Parameters

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

9.5.3.9 compiled()

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

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

9.5.3.10 getType()

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

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

9.5.3.11 operator=() [1/2]

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

9.5.3.12 operator=() [2/2]

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

9.5.3.13 reset()

```
void gla::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.
---------------------------------	---

9.5.4 Field Documentation

9.5.4.1 _compiled

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

9.5.4.2 _id

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

9.5.4.3 _type

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

9.5.4.4 Program

```
friend gla::Shader::Program
```

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

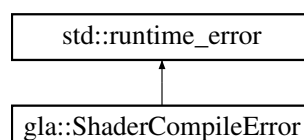
- inc/GLA/[shader.h](#)

9.6 gla::ShaderCompileError Class Reference

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

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

Inheritance diagram for gla::ShaderCompileError:



Public Member Functions

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

9.6.1 Detailed Description

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

9.6.2 Constructor & Destructor Documentation

9.6.2.1 ShaderCompileError()

```
gla::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/GLA/[shader.h](#)

9.7 gla::UniformData Struct Reference

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

Data Fields

- int [location](#)
- unsigned int [glType](#)
- int [arraySize](#)

9.7.1 Field Documentation

9.7.1.1 arraySize

```
int gla::UniformData::arraySize
```

9.7.1.2 glType

```
unsigned int gla::UniformData::glType
```

9.7.1.3 location

```
int gla::UniformData::location
```

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

- [inc/GLA/program.h](#)

9.8 gla::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]

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

9.8.1.2 UniformProxy() [2/3]

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

9.8.1.3 UniformProxy() [3/3]

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

9.8.2 Member Function Documentation

9.8.2.1 operator float()

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

9.8.2.2 operator glm::ivec2()

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

9.8.2.3 operator glm::ivec3()

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

9.8.2.4 operator glm::ivec4()

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

9.8.2.5 operator glm::mat2()

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

9.8.2.6 operator glm::mat2x3()

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

9.8.2.7 operator glm::mat2x4()

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

9.8.2.8 operator glm::mat3()

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

9.8.2.9 operator glm::mat3x2()

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

9.8.2.10 operator glm::mat3x4()

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

9.8.2.11 operator glm::mat4()

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

9.8.2.12 operator glm::mat4x2()

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


9.8.2.13 operator glm::mat4x3()

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

9.8.2.14 operator glm::uvec2()

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

9.8.2.15 operator glm::uvec3()

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

9.8.2.16 operator glm::uvec4()

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

9.8.2.17 operator glm::vec2()

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

9.8.2.18 operator glm::vec3()

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

9.8.2.19 operator glm::vec4()

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

9.8.2.20 operator int()

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

9.8.2.21 operator unsigned int()

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

9.8.2.22 operator=() [1/23]

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

9.8.2.23 operator=() [2/23]

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

9.8.2.24 operator=() [3/23]

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

9.8.2.25 operator=() [4/23]

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

9.8.2.26 operator=() [5/23]

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

9.8.2.27 operator=() [6/23]

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

9.8.2.28 operator=() [7/23]

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

9.8.2.29 operator=() [8/23]

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

9.8.2.30 operator=() [9/23]

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

9.8.2.31 operator=() [10/23]

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

9.8.2.32 operator=() [11/23]

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

9.8.2.33 operator=() [12/23]

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

9.8.2.34 operator=() [13/23]

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

9.8.2.35 operator=() [14/23]

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

9.8.2.36 operator=() [15/23]

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

9.8.2.37 operator=() [16/23]

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

9.8.2.38 operator=() [17/23]

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

9.8.2.39 operator=() [18/23]

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

9.8.2.40 operator=() [19/23]

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

9.8.2.41 operator=() [20/23]

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

9.8.2.42 operator=() [21/23]

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

9.8.2.43 operator=() [22/23]

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

9.8.2.44 operator=() [23/23]

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

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

- inc/GLA/[program.h](#)

9.9 gla::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]

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

9.9.1.2 UniformProxyConst() [2/3]

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

9.9.1.3 UniformProxyConst() [3/3]

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

9.9.2 Member Function Documentation

9.9.2.1 operator float()

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

9.9.2.2 operator glm::ivec2()

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

9.9.2.3 operator glm::ivec3()

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

9.9.2.4 operator glm::ivec4()

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

9.9.2.5 operator glm::mat2()

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

9.9.2.6 operator glm::mat2x3()

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

9.9.2.7 operator glm::mat2x4()

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

9.9.2.8 operator glm::mat3()

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

9.9.2.9 operator glm::mat3x2()

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

9.9.2.10 operator glm::mat3x4()

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

9.9.2.11 operator glm::mat4()

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

9.9.2.12 operator glm::mat4x2()

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

9.9.2.13 operator glm::mat4x3()

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

9.9.2.14 operator glm::uvec2()

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

9.9.2.15 operator glm::uvec3()

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

9.9.2.16 operator glm::uvec4()

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

9.9.2.17 operator glm::vec2()

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

9.9.2.18 operator glm::vec3()

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

9.9.2.19 operator glm::vec4()

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

9.9.2.20 operator int()

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

9.9.2.21 operator unsigned int()

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

9.9.2.22 operator=() [1/2]

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

9.9.2.23 operator=() [2/2]

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

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

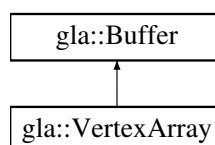
- inc/GLA/[program.h](#)

9.10 gla::VertexArray Class Reference

[VertexArray](#) class to abstract the OpenGL vertex array.

```
#include <vertexArray.h>
```

Inheritance diagram for gla::VertexArray:



Public Member Functions

- [VertexArray](#) ()
- [VertexArray](#) (VertexArray &&other)
- [VertexArray](#) (const VertexArray &other)=delete
- void [setAttributes](#) (const std::vector< [VertexAttribute](#) > &attribs, int stride)
Set the Attributes for a vertex array.
- [VertexArray](#) & [operator=](#) ([VertexArray](#) &&other)
- [VertexArray](#) & [operator=](#) (const [VertexArray](#) &other)=delete

Public Member Functions inherited from [gla::Buffer](#)

- [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

Additional Inherited Members

Protected Member Functions inherited from [gla::Buffer](#)

- void [_delete](#) ()
- void [_check](#) ()

Protected Attributes inherited from [gla::Buffer](#)

- unsigned int `_id` = 0
- bool `_mapped` = false
- [MapUsage](#) `_mapUsage` = [MapUsage::None](#)
- [BufferFlag](#) `_flags` = [BufferFlag::None](#)
- [BufferType](#) `_type`

9.10.1 Detailed Description

[VertexArray](#) class to abstract the OpenGL vertex array.

Warning

[Program](#) must be deconstructed before the OpenGL context is destroyed.
This class is not guaranteed to be thread-safe.

Note

Inherits from [gla::Buffer](#).

9.10.2 Constructor & Destructor Documentation

9.10.2.1 [VertexArray\(\)](#) [1/3]

```
gla::VertexArray::VertexArray () [inline]
```

9.10.2.2 [VertexArray\(\)](#) [2/3]

```
gla::VertexArray::VertexArray (  
    VertexArray && other) [inline]
```

9.10.2.3 [VertexArray\(\)](#) [3/3]

```
gla::VertexArray::VertexArray (  
    const VertexArray & other) [delete]
```

9.10.3 Member Function Documentation

9.10.3.1 [operator=\(\)](#) [1/2]

```
VertexArray & gla::VertexArray::operator= (  
    const VertexArray & other) [delete]
```

9.10.3.2 operator=() [2/2]

```
VertexArray & gla::VertexArray::operator= (
    VertexArray && other) [inline]
```

9.10.3.3 setAttributes()

```
void gla::VertexArray::setAttributes (
    const std::vector< VertexAttribute > & attribs,
    int stride)
```

Set the Attributes for a vertex array.

Exceptions

<i>std::invalid_argument</i>	If stride is less than or equal to 0
<i>std::runtime_error</i>	If it could not query GL_MAX_VERTEX_ATTRIBS
<i>std::runtime_error</i>	If the current GPU doesn't support the given amount of VertexAttribute (at least 16 are guaranteed)
<i>std::invalid_argument</i>	If the any given index goes above the amount of VertexAttributes supported by the GPU (at least 16 are guaranteed)
<i>std::invalid_argument</i>	If any VertexAttribute requests less than 1 or more than 4 numComponents
<i>std::invalid_argument</i>	If any of the given combinations of type and interpretation is invalid
<i>std::invalid_argument</i>	If the given VertexAttribType in any VertexAttribute is invalid
<i>std::invalid_argument</i>	If the given VertexAttributes extend over the given stride (only when DEBUG_MODE is defined)
<i>std::invalid_argument</i>	If the given VertexAttributes overlap (only when DEBUG_MODE is defined)

Note

Calling this function binds this [Buffer](#).

Parameters

<i>attribs</i>	Vector of Attributes to assign to the VertexArray
----------------	---

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

- inc/GLA/[vertexArray.h](#)

9.11 gla::VertexAttribute Struct Reference

Defines a vertex attribute for the [gla::VertexArray](#).

```
#include <vertexArray.h>
```

Data Fields

- unsigned int [index](#)
VertexAttribute location in the GLSL shader program.
- int [numComponents](#)
number of components for VertexAttribute may be [1;4]. For example 3 for vec3.
- [VertexAttribType](#) type
Type of the VertexAttribute.
- [VertexAttribInterp](#) interp
Interpretation of the VertexAttribute, for example is type Byte is specified, but should be used as a float.
- bool [normalized](#)
If it the vertex Attribute should be mapped to [-1;1] for signed values or [0;1] for unsigned values. (disregarded for int types).
- int [offset](#)
Offset to the start of the current VertexAttribute.

9.11.1 Detailed Description

Defines a vertex attribute for the [glsl::VertexArray](#).

9.11.2 Field Documentation

9.11.2.1 index

```
unsigned int glsl::VertexAttribute::index
```

[VertexAttribute](#) location in the GLSL shader program.

9.11.2.2 interp

```
VertexAttribInterp glsl::VertexAttribute::interp
```

Interpretation of the [VertexAttribute](#), for example is type Byte is specified, but should be used as a float.

9.11.2.3 normalized

```
bool glsl::VertexAttribute::normalized
```

If it the vertex Attribute should be mapped to [-1;1] for signed values or [0;1] for unsigned values. (disregarded for int types).

9.11.2.4 numComponents

```
int glsl::VertexAttribute::numComponents
```

number of components for [VertexAttribute](#) may be [1;4]. For example 3 for vec3.

9.11.2.5 offset

```
int gla::VertexAttribute::offset
```

Offset to the start of the current [VertexAttribute](#).

9.11.2.6 type

```
VertexAttribType gla::VertexAttribute::type
```

Type of the [VertexAttribute](#).

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

- inc/GLA/[vertexArray.h](#)

9.12 gla::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.

9.12.1 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.

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

9.12.2 Constructor & Destructor Documentation

9.12.2.1 WindowContext() [1/3]

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

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

Exceptions

<i>std::runtime_error</i>	If GLFW could not create a Window.
<i>std::runtime_error</i>	If the GLFW Window is invalid.
<i>std::runtime_error</i>	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.

9.12.2.2 WindowContext() [2/3]

```
gla::WindowContext::WindowContext (
    WindowContext && other)
```

9.12.2.3 WindowContext() [3/3]

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

9.12.2.4 ~WindowContext()

```
gla::WindowContext::~~WindowContext ()
```

9.12.3 Member Function Documentation

9.12.3.1 operator=() [1/2]

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

9.12.3.2 operator=() [2/2]

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

9.12.3.3 run()

```
virtual void gla::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.

9.12.3.4 shouldClose()

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

Checks if the Window should close.

Exceptions

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

9.12.3.5 swapBuffers()

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

Swaps the display buffers.

Exceptions

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

9.12.3.6 useContext()

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

Makes the owned window the current context.

Exceptions

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

9.12.4 Field Documentation

9.12.4.1 window

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

GLFW window handle for low level GLFW access.

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

- inc/GLA/[windowContext.h](#)

Chapter 10

File Documentation

10.1 inc/GLA/buffer.h File Reference

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

Data Structures

- class [gla::Buffer](#)

Namespaces

- namespace [gla](#)

Enumerations

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

Functions

- [MapUsage gla::operator|](#) ([MapUsage](#) a, [MapUsage](#) b)
- [MapUsage gla::operator&](#) ([MapUsage](#) a, [MapUsage](#) b)
- [MapUsage & gla::operator|=](#) ([MapUsage](#) &a, [MapUsage](#) b)
- [BufferFlag gla::operator|](#) ([BufferFlag](#) a, [BufferFlag](#) b)
- [BufferFlag gla::operator&](#) ([BufferFlag](#) a, [BufferFlag](#) b)
- [BufferFlag & gla::operator|=](#) ([BufferFlag](#) &a, [BufferFlag](#) b)
- unsigned int [gla::toGLenum](#) ([BufferType](#) type)
Converts a [BufferType](#) enum into a GLenum.
- unsigned int [gla::toGLenum](#) ([BufferUsage](#) usage)
Converts a [BufferUsage](#) enum into a GLenum.
- unsigned int [gla::toGLenum](#) ([MapUsage](#) usage)
Converts a [MapUsage](#) enum into a GLenum.
- unsigned int [gla::toGLenum](#) ([BufferFlag](#) flag)
Converts a [BufferFlag](#) enum into a GLenum.
- bool [gla::validateMapUsage](#) ([MapUsage](#) usage, std::string &error)
Checks if the given [MapUsage](#) flag combination is valid.
- bool [gla::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 GLA_BUFFER_H
00002 #define GLA_BUFFER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <stdint>
00007 #include <vector>
00008
00009 namespace gla {
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
00106 enum class BufferFlag : uint32_t {
00107     None                = 0,
00108     DynamicStorage      = 1 << 0,
00109     MapRead              = 1 << 1,
00110     MapWrite             = 1 << 2,
00111     MapPersistent       = 1 << 3,
00112     MapCoherent         = 1 << 4,
00113     ClientStorage       = 1 << 5
00114 };
00115
00116 inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
00117     return static_cast<BufferFlag>(
00118         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00119     );
00120 }
00121
00122 inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
00123     return static_cast<BufferFlag>(
00124         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00125     );
00126 }
00127
00128 inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
00129     a = a | b;
00130     return a;
00131 }
00132
00138 unsigned int toGLenum(BufferType type);
00139
00145 unsigned int toGLenum(BufferUsage usage);
00146
00150 unsigned int toGLenum(MapUsage usage);
00151
00155 unsigned int toGLenum(BufferFlag flag);
00156
00165 bool validateMapUsage(MapUsage usage, std::string& error);
00166
00175 bool validateBufferFlag(BufferFlag flag, std::string& error);
00176
00177 class Buffer {
00178 protected:
00179     unsigned int _id = 0;
00180     bool _mapped = false;
00181     MapUsage _mapUsage = MapUsage::None;
00182     BufferFlag _flags = BufferFlag::None;
00183     BufferType _type;
00184
00185     void _delete();
00186     void _check();
00187
00188 public:
00189     Buffer() = delete;
00193     Buffer(BufferType type);
00194     Buffer(Buffer&& other);
00195     Buffer(const Buffer& other) = delete;
00196     ~Buffer() noexcept;
00197
00201     void bind() const;
00202
00206     int64_t size() const;
00207
00211     BufferType getType() const;
00212

```

```

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),
(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),
(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/GLA/debug.h File Reference

Namespaces

- namespace [gla](#)

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 * [gla::glErrorString](#) (unsigned int err)
Gets a c-style string from from a OpenGL enum.
- void [gla::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 [gla::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 GLA_DEBUG_H
00002 #define GLA_DEBUG_H
00003
00004 namespace gla {
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; gla::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/GLA/mainpage.md File Reference

10.6 inc/GLA/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 [gla::ProgramLinkError](#)
Exception thrown when [Program](#) linking fails.
- class [gla::ProgramValidateError](#)
Exception thrown when [Program](#) validation fails, mainly used in debug builds.
- struct [gla::UniformData](#)
- class [gla::Program](#)
[Program](#) class to abstract OpenGL [Shader](#) Programs.
- class [gla::Program::UniformProxy](#)
- class [gla::Program::UniformProxyConst](#)

Namespaces

- namespace [gla](#)

10.7 program.h

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

```

00001 #ifndef GLA_PROGRAM_H
00002 #define GLA_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 gla {
00013
00014     class Shader;
00015
00019     class ProgramLinkError : public std::runtime_error {
00020     public:
00026         ProgramLinkError(const std::string& infoLog)
00027             : std::runtime_error(
00028                 "Program link failed:\n" + infoLog) {}
00029     };
00030
00034     class ProgramValidateError : public std::runtime_error {
00035     public:
00041         ProgramValidateError(const std::string& infoLog)
00042             : std::runtime_error(
00043                 "Program validation failed:\n" + infoLog) {}
00044     };
00045
00046     struct UniformData {
00047         int location;
00048         unsigned int glType;
00049         int arraySize;
00050     };
00051
00061     class Program {
00062     protected:
00063         unsigned int _id = 0;
00064         bool _linked = false;
00065
00066         std::unordered_map<std::string, int> _uniformIndexMap = {}; // name to uniform index conversion
00067         std::unordered_map<int, int> _uniformLocationIndexMap = {}; // location to uniform index
00068     conversion
00068         std::vector<UniformData> _uniformData = {}; // uniform data per index
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; }
00089         explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
00090     return val; }
00090         explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
00091     return val; }
00091         explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
00092         explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
00093     return val; }
00093         explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
00094     return val; }
00094         explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
00095     return val; }
00095         explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val);
00096     return val; }
00096         explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val);
00097     return val; }
00097         explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val);
00098     return val; }
00098         explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val);
00099     return val; }
00099         explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
00100     val; }
00100         explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
00101     val; }
00101         explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
00102     val; }
00102         explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
00103     return val; }
00103         explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
00104     return val; }
00104         explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
00105     return val; }
00105         explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
00106     return val; }
00106         explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
00107     return val; }
00107         explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
00108     return val; }
00108
00109         UniformProxy& operator=(float other) { _parent.setUniform(_location, other); return *this; }
00110         UniformProxy& operator=(const glm::vec2& other) { _parent.setUniform(_location, other); return
00111     *this; }
00111         UniformProxy& operator=(const glm::vec3& other) { _parent.setUniform(_location, other); return
00112     *this; }
00112         UniformProxy& operator=(const glm::vec4& other) { _parent.setUniform(_location, other); return
00113     *this; }
00113         UniformProxy& operator=(int other) { _parent.setUniform(_location, other); return *this; }
00114         UniformProxy& operator=(const glm::ivec2& other) { _parent.setUniform(_location, other);
00115     return *this; }
00115         UniformProxy& operator=(const glm::ivec3& other) { _parent.setUniform(_location, other);
00116     return *this; }
00116         UniformProxy& operator=(const glm::ivec4& other) { _parent.setUniform(_location, other);
00117     return *this; }
00117         UniformProxy& operator=(unsigned int other) { _parent.setUniform(_location, other); return
00118     *this; }
00118         UniformProxy& operator=(const glm::uvec2& other) { _parent.setUniform(_location, other);
00119     return *this; }
00119         UniformProxy& operator=(const glm::uvec3& other) { _parent.setUniform(_location, other);
00120     return *this; }
00120         UniformProxy& operator=(const glm::uvec4& other) { _parent.setUniform(_location, other);
00121     return *this; }
00121         UniformProxy& operator=(const glm::mat2& other) { _parent.setUniform(_location, other); return
00122     *this; }
00122         UniformProxy& operator=(const glm::mat3& other) { _parent.setUniform(_location, other); return
00123     *this; }
00123         UniformProxy& operator=(const glm::mat4& other) { _parent.setUniform(_location, other); return
00124     *this; }
00124         UniformProxy& operator=(const glm::mat2x3& other) { _parent.setUniform(_location, other);
00125     return *this; }
00125         UniformProxy& operator=(const glm::mat3x2& other) { _parent.setUniform(_location, other);
00126     return *this; }
00126         UniformProxy& operator=(const glm::mat2x4& other) { _parent.setUniform(_location, other);
00127     return *this; }
00127         UniformProxy& operator=(const glm::mat4x2& other) { _parent.setUniform(_location, other);

```

```

    return *this; }
00128     UniformProxy& operator=(const glm::mat3x4& other) { _parent.setUniform(_location, other);
    return *this; }
00129     UniformProxy& operator=(const glm::mat4x3& other) { _parent.setUniform(_location, other);
    return *this; }
00130
00131     UniformProxy& operator=(const UniformProxy& other) = delete;
00132     UniformProxy& operator=(UniformProxy&& other) = delete;
00133 };
00134
00135     class UniformProxyConst {
00136     private:
00137         const Program& _parent;
00138         int _location;
00139
00140     public:
00141         UniformProxyConst(const Program& parent, int location) : _parent(parent), _location(location)
    {}
00142         UniformProxyConst(UniformProxyConst&& other) = delete;
00143         UniformProxyConst(const UniformProxyConst& other) = delete;
00144
00145         explicit operator float() const { float val; _parent.getUniform(_location, val); return val; }
00146         explicit operator glm::vec2() const { glm::vec2 val; _parent.getUniform(_location, val);
    return val; }
00147         explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
    return val; }
00148         explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
    return val; }
00149         explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
00150         explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
    return val; }
00151         explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
    return val; }
00152         explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
    return val; }
00153         explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val);
    return val; }
00154         explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val);
    return val; }
00155         explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val);
    return val; }
00156         explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val);
    return val; }
00157         explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
    val; }
00158         explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
    val; }
00159         explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
    val; }
00160         explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
    return val; }
00161         explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
    return val; }
00162         explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
    return val; }
00163         explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
    return val; }
00164         explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
    return val; }
00165         explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
    return val; }
00166
00167         UniformProxyConst& operator=(const UniformProxyConst& other) = delete;
00168         UniformProxyConst& operator=(UniformProxyConst&& other) = delete;
00169     };
00170
00171     public:
00172         Program();
00173         Program(Program&& other);
00174         Program(const Program&) = delete; // OpenGL Programs are not copy safe
00175         ~Program();
00176
00177         void reset();
00178
00179         bool attached(const Shader& shader);
00180
00181         void attach(const Shader& shader);
00182
00183         void detach(const Shader& shader);
00184
00185         bool linked() const { return _linked; }
00186
00187         void link();
00188
00189         void bind() const;
00190
00191         static void unbind();

```



```

00253
00264     int glGetUniformLocation(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     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         setUniform(getUniformLocation(name), 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         setUniform(getUniformLocation(name), 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         setUniform(getUniformLocation(name), 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         setUniform(getUniformLocation(name), data); }
00338     void setUniform(const std::string& name, const glm::mat3x4& data) {
00339         setUniform(getUniformLocation(name), 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;

```

```

00390     void glUniform(const std::string& name, float& data) const { glUniform(getUniformLocation(name),
data); }
00391     void glUniform(const std::string& name, glm::vec2& data) const {
    glUniform(getUniformLocation(name), data); }
00392     void glUniform(const std::string& name, glm::vec3& data) const {
    glUniform(getUniformLocation(name), data); }
00393     void glUniform(const std::string& name, glm::vec4& data) const {
    glUniform(getUniformLocation(name), data); }
00394     void glUniform(const std::string& name, int& data) const { glUniform(getUniformLocation(name),
data); }
00395     void glUniform(const std::string& name, glm::ivec2& data) const {
    glUniform(getUniformLocation(name), data); }
00396     void glUniform(const std::string& name, glm::ivec3& data) const {
    glUniform(getUniformLocation(name), data); }
00397     void glUniform(const std::string& name, glm::ivec4& data) const {
    glUniform(getUniformLocation(name), data); }
00398     void glUniform(const std::string& name, unsigned int& data) const {
    glUniform(getUniformLocation(name), data); }
00399     void glUniform(const std::string& name, glm::uvec2& data) const {
    glUniform(getUniformLocation(name), data); }
00400     void glUniform(const std::string& name, glm::uvec3& data) const {
    glUniform(getUniformLocation(name), data); }
00401     void glUniform(const std::string& name, glm::uvec4& data) const {
    glUniform(getUniformLocation(name), data); }
00402     void glUniform(const std::string& name, glm::mat2& data) const {
    glUniform(getUniformLocation(name), data); }
00403     void glUniform(const std::string& name, glm::mat3& data) const {
    glUniform(getUniformLocation(name), data); }
00404     void glUniform(const std::string& name, glm::mat4& data) const {
    glUniform(getUniformLocation(name), data); }
00405     void glUniform(const std::string& name, glm::mat2x3& data) const {
    glUniform(getUniformLocation(name), data); }
00406     void glUniform(const std::string& name, glm::mat3x2& data) const {
    glUniform(getUniformLocation(name), data); }
00407     void glUniform(const std::string& name, glm::mat2x4& data) const {
    glUniform(getUniformLocation(name), data); }
00408     void glUniform(const std::string& name, glm::mat4x2& data) const {
    glUniform(getUniformLocation(name), data); }
00409     void glUniform(const std::string& name, glm::mat3x4& data) const {
    glUniform(getUniformLocation(name), data); }
00410     void glUniform(const std::string& name, glm::mat4x3& data) const {
    glUniform(getUniformLocation(name), data); }
00411
00412     Program& operator=(Program&& other);
00413     Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe
00414
00425     UniformProxy operator[](int location) { return UniformProxy(*this, location); }
00440     UniformProxy operator[](const std::string& name) { return UniformProxy(*this,
    glUniformLocation(name)); }
00448     UniformProxyConst operator[](int location) const { return UniformProxyConst(*this, location); }
00460     UniformProxyConst operator[](const std::string& name) const { return UniformProxyConst(*this,
    glUniformLocation(name)); }
00461 };
00462
00463 }
00464
00465 #endif

```

10.8 inc/GLA/shader.h File Reference

```

#include <string>
#include <stdexcept>
#include <iosfwd>

```

Data Structures

- class `gla::ShaderCompileError`
Exception thrown when `Shader` compilation fails.
- class `gla::Shader`
`Shader` class to abstract OpenGL Shaders.

Namespaces

- namespace `gla`

Enumerations

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

Functions

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

10.9 shader.h

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

```
00001 #ifndef GLA_SHADER_H
00002 #define GLA_SHADER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <iosfwd> // std::istream forward-declared
00007
00008 namespace gla {
00009
00010 class Program;
00011
00012 enum class ShaderType {
00013     Fragment,
00014     Vertex,
00015     Geometry,
00016     TessEvaluation,
00017     TessControl,
00018     Compute
00019 };
00020
00021 constexpr std::string shaderTypeToString(ShaderType type);
00022
00023 unsigned int toGLenum(ShaderType type);
00024
00025 class ShaderCompileError : public std::runtime_error {
00026 public:
00027     ShaderCompileError(ShaderType ShaderType,
00028                       const std::string& infoLog)
00029         : std::runtime_error(
00030             "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
00031 };
00032
00033 class Shader {
00034 protected:
00035     unsigned int _id = 0;
00036     ShaderType _type;
00037     bool _compiled = false;
00038
00039     void _delete();
00040     void _check();
00041     void _ensure();
00042     std::string _getError();
00043
00044 public:
00045     Shader() = delete;
00046     Shader(ShaderType type);
00047     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/GLA/vertexArray.h File Reference

```

#include <vector>
#include <stdexcept>
#include <GLA/buffer.h>
#include <GLA/debug.h>

```

Data Structures

- struct [gla::VertexAttribute](#)
Defines a vertex attribute for the [gla::VertexArray](#).
- class [gla::VertexArray](#)
[VertexArray](#) class to abstract the OpenGL vertex array.

Namespaces

- namespace [gla](#)

Enumerations

- enum class [gla::VertexAttribType](#) {
 [gla::Byte](#) , [gla::UnsignedByte](#) , [gla::Short](#) , [gla::UnsignedShort](#) ,
 [gla::Int](#) , [gla::UnsignedInt](#) , [gla::HalfFloat](#) , [gla::Float](#) ,
 [gla::Double](#) , [gla::Fixed](#) }
- enum class [gla::VertexAttribInterp](#) { [gla::Float](#) , [gla::Integer](#) }

Functions

- unsigned int `gla::toGLenum` (`VertexAttribType` type)
Converts a `VertexAttribType` into a `GLenum`.
- bool `gla::validateTypeInterpretation` (`VertexAttribType` type, `VertexAttribInterp` interp, `std::string &error`)
Checks if the type and interpretation combination is valid.
- int `gla::typeToBytes` (`VertexAttribType` type)
Gets the size of the given type in bytes.

10.11 vertexArray.h

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

```

00001 #ifndef GLA_VERTEX_ARRAY_H
00002 #define GLA_VERTEX_ARRAY_H
00003
00004 #include <vector>
00005 #include <stdexcept>
00006
00007 #include <GLA/buffer.h>
00008 #include <GLA/debug.h>
00009
00010 namespace gla {
00011
00012     enum class VertexAttribType {
00013         Byte,
00014         UnsignedByte,
00015         Short,
00016         UnsignedShort,
00017         Int,
00018         UnsignedInt,
00019         HalfFloat,
00020         Float,
00021         Double,
00022         Fixed
00023     };
00024
00025     enum class VertexAttribInterp {
00026         Float,
00027         Integer
00028     };
00029
00030
00036     unsigned int toGLenum(VertexAttribType type);
00037
00045     bool validateTypeInterpretation(VertexAttribType type, VertexAttribInterp interp, std::string& error);
00046
00052     int typeToBytes(VertexAttribType type);
00053
00057     struct VertexAttribute {
00058         unsigned int index;
00059         int numComponents;
00060         VertexAttribType type;
00061         VertexAttribInterp interp;
00062         bool normalized;
00063         int offset;
00064     };
00065
00074     class VertexArray : public Buffer {
00075     private:
00076         std::vector<unsigned int> _enabledVertexAttribs = {};
00077
00078     public:
00079         VertexArray() : Buffer(BufferType::Array) {}
00080         VertexArray(VertexArray&& other) : Buffer(std::move(other)) {}
00081         VertexArray(const VertexArray& other) = delete;
00082
00101         void setAttributes(const std::vector<VertexAttribute>& attribs, int stride);
00102
00103         VertexArray& operator=(VertexArray&& other) { Buffer::operator=(std::move(other)); return *this; }
00104         VertexArray& operator=(const VertexArray& other) = delete;
00105     };
00106
00107 }
00108
00109 #endif

```

10.12 inc/GLA/windowContext.h File Reference

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

Data Structures

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

Namespaces

- namespace [gla](#)

Functions

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

10.13 windowContext.h

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

```
00001 #ifndef GLA_WINDOW_CONTEXT
00002 #define GLA_WINDOW_CONTEXT
00003
00004 #include <GLFW/glfw3.h>
00005
00006 namespace gla {
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`
 - `gla::Buffer`, [29](#)
 - `gla::Program`, [40](#)
 - `gla::Shader`, [69](#)
- `_compiled`
 - `gla::Shader`, [72](#)
- `_delete`
 - `gla::Buffer`, [29](#)
 - `gla::Program`, [40](#)
 - `gla::Shader`, [69](#)
- `_ensure`
 - `gla::Program`, [40](#)
 - `gla::Shader`, [69](#)
- `_flags`
 - `gla::Buffer`, [33](#)
- `_getError`
 - `gla::Program`, [41](#)
 - `gla::Shader`, [69](#)
- `_id`
 - `gla::Buffer`, [33](#)
 - `gla::Program`, [62](#)
 - `gla::Shader`, [72](#)
- `_linked`
 - `gla::Program`, [62](#)
- `_mapUsage`
 - `gla::Buffer`, [33](#)
- `_mapped`
 - `gla::Buffer`, [33](#)
- `_queryUniformData`
 - `gla::Program`, [41](#)
- `_setupUniform`
 - `gla::Program`, [41](#)
- `_type`
 - `gla::Buffer`, [33](#)
 - `gla::Shader`, [72](#)
- `_uniformData`
 - `gla::Program`, [63](#)
- `_uniformIndexMap`
 - `gla::Program`, [63](#)
- `_uniformLocationIndexMap`
 - `gla::Program`, [63](#)
- `~Buffer`
 - `gla::Buffer`, [28](#)
- `~Program`
 - `gla::Program`, [40](#)
- `~Shader`
 - `gla::Shader`, [68](#)
- `~WindowContext`
 - `gla::WindowContext`, [89](#)

- `Array`
 - `gla`, [17](#)
- `arraySize`
 - `gla::UniformData`, [73](#)
- `AtomicCounter`
 - `gla`, [17](#)
- `attach`
 - `gla::Program`, [41](#)
- `attached`
 - `gla::Program`, [41](#)
- `bind`
 - `gla::Buffer`, [29](#)
 - `gla::Program`, [42](#)
- `Buffer`
 - `gla::Buffer`, [28](#)
- `BufferFlag`
 - `gla`, [17](#)
- `BufferType`
 - `gla`, [17](#)
- `BufferUsage`
 - `gla`, [18](#)
- `Byte`
 - `gla`, [20](#)
- `ClientStorage`
 - `gla`, [17](#)
- `Coherent`
 - `gla`, [19](#)
- `compile`
 - `gla::Shader`, [69](#), [70](#)
- `compiled`
 - `gla::Shader`, [71](#)
- `Compute`
 - `gla`, [20](#)
- `CopyRead`
 - `gla`, [18](#)
- `CopyWrite`
 - `gla`, [18](#)
- `debug.h`
 - `DEBUG_ONLY`, [96](#)
 - `GL_CALL`, [96](#)
 - `DEBUG_ONLY`
 - `debug.h`, [96](#)
- `detach`
 - `gla::Program`, [42](#)
- `DispatchIndirect`
 - `gla`, [18](#)
- `Double`

- gla, [20](#)
- DrawIndirect
 - gla, [18](#)
- DynamicCopy
 - gla, [19](#)
- DynamicDraw
 - gla, [19](#)
- DynamicRead
 - gla, [19](#)
- DynamicStorage
 - gla, [17](#)
- ElementArray
 - gla, [18](#)
- Fixed
 - gla, [20](#)
- Float
 - gla, [20](#)
- FlushExplicit
 - gla, [19](#)
- Fragment
 - gla, [20](#)
- Geometry
 - gla, [20](#)
- getSubData
 - gla::Buffer, [29](#)
- getType
 - gla::Buffer, [29](#)
 - gla::Shader, [71](#)
- getUniform
 - gla::Program, [43–50](#)
- getUniformLocation
 - gla::Program, [50](#)
- GL_CALL
 - debug.h, [96](#)
- gla, [15](#)
 - Array, [17](#)
 - AtomicCounter, [17](#)
 - BufferFlag, [17](#)
 - BufferType, [17](#)
 - BufferUsage, [18](#)
 - Byte, [20](#)
 - ClientStorage, [17](#)
 - Coherent, [19](#)
 - Compute, [20](#)
 - CopyRead, [18](#)
 - CopyWrite, [18](#)
 - DispatchIndirect, [18](#)
 - Double, [20](#)
 - DrawIndirect, [18](#)
 - DynamicCopy, [19](#)
 - DynamicDraw, [19](#)
 - DynamicRead, [19](#)
 - DynamicStorage, [17](#)
 - ElementArray, [18](#)
 - Fixed, [20](#)
 - Float, [20](#)
 - FlushExplicit, [19](#)
 - Fragment, [20](#)
 - Geometry, [20](#)
 - glCheckError, [21](#)
 - glErrorString, [21](#)
 - HalfFloat, [20](#)
 - initGLFW, [21](#)
 - Int, [20](#)
 - Integer, [20](#)
 - InvalidateBuffer, [19](#)
 - InvalidRange, [19](#)
 - MapCoherent, [17](#)
 - MapPersistent, [17](#)
 - MapRead, [17](#)
 - MapUsage, [19](#)
 - MapWrite, [17](#)
 - None, [17, 19](#)
 - operator&, [22](#)
 - operator|, [22](#)
 - operator|=, [22](#)
 - Persistent, [19](#)
 - PixelPack, [18](#)
 - PixelUnpack, [18](#)
 - pollEvents, [23](#)
 - Query, [18](#)
 - Read, [19](#)
 - ShaderStorage, [18](#)
 - ShaderType, [19](#)
 - shaderTypeToString, [23](#)
 - Short, [20](#)
 - StaticCopy, [19](#)
 - StaticDraw, [19](#)
 - StaticRead, [19](#)
 - StreamCopy, [19](#)
 - StreamDraw, [19](#)
 - StreamRead, [19](#)
 - terminateGLFW, [23](#)
 - TessControl, [20](#)
 - TessEvaluation, [20](#)
 - Texture, [18](#)
 - toGLenum, [23, 24](#)
 - TransformFeedback, [18](#)
 - typeToBytes, [25](#)
 - Uniform, [18](#)
 - UnsignedByte, [20](#)
 - UnsignedInt, [20](#)
 - UnsignedShort, [20](#)
 - Unsynchronized, [19](#)
 - validateBufferFlag, [25](#)
 - validateMapUsage, [25](#)
 - validateTypeInterpretation, [26](#)
 - Vertex, [20](#)
 - VertexAttribInterp, [20](#)
 - VertexAttribType, [20](#)
 - Write, [19](#)
- gla::Buffer, [27](#)
 - _check, [29](#)
 - _delete, [29](#)

- [_flags](#), 33
 - [_id](#), 33
 - [_mapUsage](#), 33
 - [_mapped](#), 33
 - [_type](#), 33
 - [~Buffer](#), 28
 - [bind](#), 29
 - [Buffer](#), 28
 - [getSubData](#), 29
 - [getType](#), 29
 - [map](#), 30
 - [operator=](#), 30
 - [setData](#), 30, 31
 - [setStorage](#), 31, 32
 - [setSubData](#), 32
 - [size](#), 33
 - [unmap](#), 33
- [gla::Program](#), 34
 - [_check](#), 40
 - [_delete](#), 40
 - [_ensure](#), 40
 - [_getError](#), 41
 - [_id](#), 62
 - [_linked](#), 62
 - [_queryUniformData](#), 41
 - [_setupUniform](#), 41
 - [_uniformData](#), 63
 - [_uniformIndexMap](#), 63
 - [_uniformLocationIndexMap](#), 63
 - [~Program](#), 40
 - [attach](#), 41
 - [attached](#), 41
 - [bind](#), 42
 - [detach](#), 42
 - [getUniform](#), 43–50
 - [getUniformLocation](#), 50
 - [link](#), 51
 - [linked](#), 51
 - [operator=](#), 51, 52
 - [operator\[\]](#), 52, 53
 - [Program](#), 40
 - [reset](#), 54
 - [setUniform](#), 54–62
 - [unbind](#), 62
- [gla::Program::UniformProxy](#), 74
 - [operator float](#), 75
 - [operator glm::ivec2](#), 75
 - [operator glm::ivec3](#), 75
 - [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](#), 76
 - [operator glm::mat4x2](#), 76
 - [operator glm::mat4x3](#), 76
 - [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](#), 77
 - [operator=](#), 77–80
 - [UniformProxy](#), 75
- [gla::Program::UniformProxyConst](#), 80
 - [operator float](#), 81
 - [operator glm::ivec2](#), 81
 - [operator glm::ivec3](#), 81
 - [operator glm::ivec4](#), 81
 - [operator glm::mat2](#), 81
 - [operator glm::mat2x3](#), 81
 - [operator glm::mat2x4](#), 81
 - [operator glm::mat3](#), 82
 - [operator glm::mat3x2](#), 82
 - [operator glm::mat3x4](#), 82
 - [operator glm::mat4](#), 82
 - [operator glm::mat4x2](#), 82
 - [operator glm::mat4x3](#), 82
 - [operator glm::uvec2](#), 82
 - [operator glm::uvec3](#), 82
 - [operator glm::uvec4](#), 82
 - [operator glm::vec2](#), 82
 - [operator glm::vec3](#), 83
 - [operator glm::vec4](#), 83
 - [operator int](#), 83
 - [operator unsigned int](#), 83
 - [operator=](#), 83
 - [UniformProxyConst](#), 81
- [gla::ProgramLinkError](#), 63
 - [ProgramLinkError](#), 64
- [gla::ProgramValidateError](#), 64
 - [ProgramValidateError](#), 65
- [gla::Shader](#), 65
 - [_check](#), 69
 - [_compiled](#), 72
 - [_delete](#), 69
 - [_ensure](#), 69
 - [_getError](#), 69
 - [_id](#), 72
 - [_type](#), 72
 - [~Shader](#), 68
 - [compile](#), 69, 70
 - [compiled](#), 71
 - [getType](#), 71
 - [operator=](#), 71
 - [Program](#), 72
 - [reset](#), 71
 - [Shader](#), 66–68
- [gla::ShaderCompileError](#), 72
 - [ShaderCompileError](#), 73
- [gla::UniformData](#), 73
 - [arraySize](#), 73

- glType, 73
- location, 74
- gla::VertexArray, 83
 - operator=, 85
 - setAttributes, 86
 - VertexArray, 85
- gla::VertexAttribute, 86
 - index, 87
 - interp, 87
 - normalized, 87
 - numComponents, 87
 - offset, 87
 - type, 88
- gla::WindowContext, 88
 - ~WindowContext, 89
 - operator=, 90
 - run, 90
 - shouldClose, 90
 - swapBuffers, 90
 - useContext, 91
 - window, 91
 - WindowContext, 89
- glCheckError
 - gla, 21
- glErrorString
 - gla, 21
- glType
 - gla::UniformData, 73
- HalfFloat
 - gla, 20
- inc Directory Reference, 13
- inc/GLA Directory Reference, 13
- inc/GLA/buffer.h, 93, 94
- inc/GLA/debug.h, 96, 97
- inc/GLA/mainpage.md, 97
- inc/GLA/program.h, 97, 98
- inc/GLA/shader.h, 102, 103
- inc/GLA/vertexArray.h, 104, 105
- inc/GLA/windowContext.h, 106
- index
 - gla::VertexAttribute, 87
- initGLFW
 - gla, 21
- Int
 - gla, 20
- Integer
 - gla, 20
- interp
 - gla::VertexAttribute, 87
- Introduction, 1
- InvalidateBuffer
 - gla, 19
- InvalidRange
 - gla, 19
- link
 - gla::Program, 51
- linked
 - gla::Program, 51
- location
 - gla::UniformData, 74
- map
 - gla::Buffer, 30
- MapCoherent
 - gla, 17
- MapPersistent
 - gla, 17
- MapRead
 - gla, 17
- MapUsage
 - gla, 19
- MapWrite
 - gla, 17
- None
 - gla, 17, 19
- normalized
 - gla::VertexAttribute, 87
- numComponents
 - gla::VertexAttribute, 87
- offset
 - gla::VertexAttribute, 87
- operator float
 - gla::Program::UniformProxy, 75
 - gla::Program::UniformProxyConst, 81
- operator glm::ivec2
 - gla::Program::UniformProxy, 75
 - gla::Program::UniformProxyConst, 81
- operator glm::ivec3
 - gla::Program::UniformProxy, 75
 - gla::Program::UniformProxyConst, 81
- operator glm::ivec4
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 81
- operator glm::mat2
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 81
- operator glm::mat2x3
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 81
- operator glm::mat2x4
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 81
- operator glm::mat3
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 82
- operator glm::mat3x2
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 82
- operator glm::mat3x4
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 82
- operator glm::mat4
 - gla::Program::UniformProxy, 76

- gla::Program::UniformProxyConst, 82
- operator glm::mat4x2
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 82
- operator glm::mat4x3
 - gla::Program::UniformProxy, 76
 - gla::Program::UniformProxyConst, 82
- operator glm::uvec2
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 82
- operator glm::uvec3
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 82
- operator glm::uvec4
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 82
- operator glm::vec2
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 82
- operator glm::vec3
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 83
- operator glm::vec4
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 83
- operator int
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 83
- operator unsigned int
 - gla::Program::UniformProxy, 77
 - gla::Program::UniformProxyConst, 83
- operator=
 - gla::Buffer, 30
 - gla::Program, 51, 52
 - gla::Program::UniformProxy, 77–80
 - gla::Program::UniformProxyConst, 83
 - gla::Shader, 71
 - gla::VertexArray, 85
 - gla::WindowContext, 90
- operator&
 - gla, 22
- operator[]
 - gla::Program, 52, 53
- operator |
 - gla, 22
- operator |=
 - gla, 22
- Persistent
 - gla, 19
- PixelPack
 - gla, 18
- PixelUnpack
 - gla, 18
- pollEvents
 - gla, 23
- Program
 - gla::Program, 40
 - gla::Shader, 72
- ProgramLinkError
 - gla::ProgramLinkError, 64
- ProgramValidateError
 - gla::ProgramValidateError, 65
- Query
 - gla, 18
- Read
 - gla, 19
- reset
 - gla::Program, 54
 - gla::Shader, 71
- run
 - gla::WindowContext, 90
- setAttributes
 - gla::VertexArray, 86
- setData
 - gla::Buffer, 30, 31
- setStorage
 - gla::Buffer, 31, 32
- setSubData
 - gla::Buffer, 32
- setUniform
 - gla::Program, 54–62
- Shader
 - gla::Shader, 66–68
- ShaderCompileError
 - gla::ShaderCompileError, 73
- ShaderStorage
 - gla, 18
- ShaderType
 - gla, 19
- shaderTypeToString
 - gla, 23
- Short
 - gla, 20
- shouldClose
 - gla::WindowContext, 90
- size
 - gla::Buffer, 33
- StaticCopy
 - gla, 19
- StaticDraw
 - gla, 19
- StaticRead
 - gla, 19
- StreamCopy
 - gla, 19
- StreamDraw
 - gla, 19
- StreamRead
 - gla, 19
- swapBuffers
 - gla::WindowContext, 90
- terminateGLFW
 - gla, 23

- TessControl
 - gla, [20](#)
- TessEvaluation
 - gla, [20](#)
- Texture
 - gla, [18](#)
- toGLenum
 - gla, [23](#), [24](#)
- TransformFeedback
 - gla, [18](#)
- type
 - gla::VertexAttribute, [88](#)
- typeToBytes
 - gla, [25](#)

- unbind
 - gla::Program, [62](#)
- Uniform
 - gla, [18](#)
- UniformProxy
 - gla::Program::UniformProxy, [75](#)
- UniformProxyConst
 - gla::Program::UniformProxyConst, [81](#)
- unmap
 - gla::Buffer, [33](#)
- UnsignedByte
 - gla, [20](#)
- UnsignedInt
 - gla, [20](#)
- UnsignedShort
 - gla, [20](#)
- Unsynchronized
 - gla, [19](#)
- useContext
 - gla::WindowContext, [91](#)

- validateBufferFlag
 - gla, [25](#)
- validateMapUsage
 - gla, [25](#)
- validateTypeInterpretation
 - gla, [26](#)
- Vertex
 - gla, [20](#)
- VertexArray
 - gla::VertexArray, [85](#)
- VertexAttribInterp
 - gla, [20](#)
- VertexAttribType
 - gla, [20](#)

- window
 - gla::WindowContext, [91](#)
- WindowContext
 - gla::WindowContext, [89](#)
- Write
 - gla, [19](#)