

# OpenGL Abstraction

Generated by Doxygen 1.16.1



---

<b>1 Introduction</b>	<b>1</b>
1.1 Recommended Setup . . . . .	1
<b>2 Directory Hierarchy</b>	<b>3</b>
2.1 Directories . . . . .	3
<b>3 Namespace Index</b>	<b>5</b>
3.1 Namespace List . . . . .	5
<b>4 Hierarchical Index</b>	<b>7</b>
4.1 Class Hierarchy . . . . .	7
<b>5 Data Structure Index</b>	<b>9</b>
5.1 Data Structures . . . . .	9
<b>6 File Index</b>	<b>11</b>
6.1 File List . . . . .	11
<b>7 Directory Documentation</b>	<b>13</b>
7.1 inc/GLA Directory Reference . . . . .	13
7.2 inc Directory Reference . . . . .	13
<b>8 Namespace Documentation</b>	<b>15</b>
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
<b>9 Data Structure Documentation</b> . . . . .	<b>27</b>
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::VertexAttrib 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
<b>10 File Documentation</b> . . . . .	<b>93</b>
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
<b>Index</b>	<b>107</b>

# 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:

<a href="#">gla</a> . . . . .	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::VertexAttrib . . . . .	86
gla::WindowContext . . . . .	88



# Chapter 5

## Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

<code>gla::Buffer</code> . . . . .	27
<code>gla::Program</code> Program class to abstract OpenGL Shader Programs . . . . .	34
<code>gla::ProgramLinkError</code> Exception thrown when Program linking fails . . . . .	63
<code>gla::ProgramValidateError</code> Exception thrown when Program validation fails, mainly used in debug builds . . . . .	64
<code>gla::Shader</code> Shader class to abstract OpenGL Shaders . . . . .	65
<code>gla::ShaderCompileError</code> Exception thrown when Shader compilation fails . . . . .	72
<code>gla::UniformData</code> . . . . .	73
<code>gla::Program::UniformProxy</code> . . . . .	74
<code>gla::Program::UniformProxyConst</code> . . . . .	80
<code>gla::VertexArray</code> VertexArray class to abstract the OpenGL vertex array . . . . .	83
<code>gla::VertexAttrib</code> Defines a vertex attribute for the <code>gla::VertexArray</code> . . . . .	86
<code>gla::WindowContext</code> 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/ <a href="#">buffer.h</a>	93
inc/GLA/ <a href="#">debug.h</a>	96
inc/GLA/ <a href="#">program.h</a>	97
inc/GLA/ <a href="#">shader.h</a>	102
inc/GLA/ <a href="#">vertexArray.h</a>	104
inc/GLA/ <a href="#">windowContext.h</a>	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 [VertexAttrib](#)

*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 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 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 to indicate Buffer map usage.*
- 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)`
  
*Converts a `MapUsage` enum into a `GLenum`.*
- `MapUsage operator& (MapUsage a, MapUsage b)`
  
*Converts a `MapUsage` enum into a `GLenum`.*
- `MapUsage & operator|= (MapUsage &a, MapUsage b)`
  
*Converts a `MapUsage` enum into a `GLenum`.*
- `BufferFlag operator| (BufferFlag a, BufferFlag b)`
  
*Converts a `BufferFlag` enum into a `GLenum`.*
- `BufferFlag operator& (BufferFlag a, BufferFlag b)`
  
*Converts a `BufferFlag` enum into a `GLenum`.*
- `BufferFlag & operator|= (BufferFlag &a, BufferFlag b)`
  
*Converts a `BufferFlag` enum into a `GLenum`.*
- `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 flags for explicit `Buffer` usage in `setStorage`.

#### Enumerator

None	
DynamicStorage	Indicates that the contents of the data store may be updated after creation through calls to <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 that shared access to buffers that are simultaneously mapped for client access and are used by the server will be coherent, so long as that mapping is performed using <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	<a href="#">Buffer</a> copy source.
CopyWrite	<a href="#">Buffer</a> 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 fashion and that the client intends to hold and use the returned pointer during subsequent GL operation.
Coherent	Indicates that a persistent mapping is also to be coherent. Coherent maps guarantee that the effect of writes to a buffer's data store by either the client or server will eventually become visible to the other without further intervention from the application.
InvalidRange	Indicates that the previous contents of the specified range may be discarded. Data within this range are undefined with the exception of subsequently written data.
InvalidateBuffer	Indicates that the previous contents of the entire buffer may be discarded. Data within the entire buffer are undefined with the exception of subsequently written data.
FlushExplicit	Indicates that one or more discrete subranges of the mapping may be modified. When this flag is set, modifications to each subrange must be explicitly flushed by calling <code>glFlushMappedBufferRange</code> .
Unsynchronized	Indicates that the GL should not attempt to synchronize pending operations on the buffer prior to returning from <code>glMapBufferRange</code> or <code>glMapNamedBufferRange</code> .

### 8.1.1.5 ShaderType

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

`ShaderType` enum to indicate usage.

#### Enumerator

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

### 8.1.1.6 VertexAttribInterp

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

#### Enumerator

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

### 8.1.1.7 VertexAttribType

```
enum class gla::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**

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

**Returns**

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

**8.1.2.12 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

<code>std::invalid_argument</code>	If the <a href="#">BufferType</a> is invalid.
------------------------------------	---

### 8.1.2.15 toGLenum() [3/6]

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

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

#### Exceptions

<code>std::invalid_argument</code>	If the <a href="#">BufferUsage</a> 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

<code>std::logic_error</code>	If the <a href="#">ShaderType</a> is invalid.
-------------------------------	---

#### Parameters

<code>type</code>	The <a href="#">ShaderType</a> 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 <a href="#">VertexAttribType</a> 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 <a href="#">VertexAttribType</a> 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

<code>flag</code>	<a href="#">BufferFlag</a> flag combination to check
<code>error</code>	Error string output if the combination is invalid

#### Returns

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

### 8.1.2.21 validateMapUsage()

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

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

#### Parameters

<code>usage</code>	<code>MapUsage</code> flag combination to check
<code>error</code>	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

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

#### Returns

true if it is valid, false otherwise

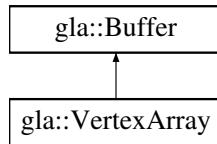
# Chapter 9

## 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

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

#### Parameters

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

### 9.1.2.5 `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

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

#### Parameters

<code>offset</code>	The offset of the map range into the <a href="#">Buffer</a> in bytes
<code>length</code>	The length of the map range in the <a href="#">Buffer</a> in bytes
<code>access</code>	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 <a href="#">Buffer</a> (must have at least size bytes of data)
<code>usage</code>	The usage hint of the <a href="#">Buffer</a>

### 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 <a href="#">Buffer</a>
<code>usage</code>	The usage hint of the <a href="#">Buffer</a>

### 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 <a href="#">BufferFlag</a> combination is invalid

**Parameters**

<i>size</i>	The size of the data in bytes
<i>data</i>	The data to store in the <a href="#">Buffer</a> (must have at least size bytes of data)
<i>flags</i>	The <a href="#">Buffer</a> 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 <a href="#">BufferFlag</a> combination is invalid

**Parameters**

<i>data</i>	The data to store in the <a href="#">Buffer</a>
<i>flags</i>	The <a href="#">Buffer</a> 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 <a href="#">Buffer</a>
<i>std::runtime_error</i>	If the <a href="#">Buffer</a> is mapped and <a href="#">MapUsage::Persistent</a> 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

<code>data</code>	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

<code>std::runtime_error</code>	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.*



- 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 destroyed [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 <a href="#">Shader</a> has already been attached.
<code>std::logic_error</code>	If the current <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state)

**Parameters**

<code>shader</code>	The <a href="#">Shader</a> that should be attached to the <a href="#">Program</a> .
---------------------	---

### 9.2.3.8 attached()

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

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

#### Parameters

<code>shader</code>	The <a href="#">Shader</a> to check.
---------------------	--------------------------------------

#### Exceptions

<code>std::logic_error</code>	If the current <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> 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

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

<code>std::runtime_error</code>	If the <a href="#">Shader</a> was not attached.
---------------------------------	---

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

**Parameters**

<i>shader</i>	The <a href="#">Shader</a> that should be detached from the <a href="#">Program</a> .
---------------	---

**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 <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> 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 <a href="#">Program</a> 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 <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> 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 <a href="#">Program</a> is unlinked

**Parameters**

<i>location</i>	The uniform location returned by <a href="#">getUniformLocation()</a>
<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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 getUniform() [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 <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state)
<i>std::runtime_error</i>	If the current <a href="#">Program</a> 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 <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state)
<i>gla::ProgramLinkError</i>	If the <a href="#">Program</a> fails to link. The <a href="#">Program</a> remains in a valid state afterwards.
<i>gla::ProgramValidateError</i>	If the <a href="#">Program</a> 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

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

#### Parameters

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

### 9.2.3.59 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**

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

**Parameters**

<code>name</code>	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**

<code>location</code>	The uniform location returned by <a href="#">getUniformLocation()</a>
-----------------------	---

**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 <a href="#">getUniformLocation()</a>
-----------------	---

**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 <a href="#">Program</a> .
---------------------------	--

**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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> 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 <a href="#">Program</a> 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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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 setUniform() [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**

<code>std::logic_error</code>	If the current <a href="#">Program</a> object does not exist ( <a href="#">reset()</a> 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 <a href="#">Program</a> is unlinked

**Warning**

Setting a uniform binds the program.

**Parameters**

<code>location</code>	The uniform location returned by <a href="#">getUniformLocation()</a>
<code>data</code>	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:

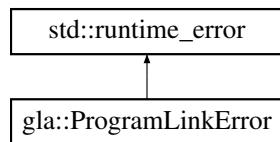
- inc/GLA/program.h

## 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:

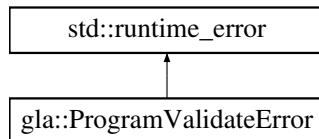
- inc/GLA/program.h

## 9.4 gla::ProgramValidationError Class Reference

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

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

Inheritance diagram for gla::ProgramValidationError:



### Public Member Functions

- [ProgramValidationError](#) (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]

```
gla::Shader::Shader (
    ShaderType type)
```

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

#### Exceptions

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

### 9.5.2.3 Shader() [3/8]

```
gla::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 <a href="#">ShaderType</a> is invalid.
<i>std::runtime_error</i>	If OpenGL failed to create a <a href="#">Shader</a> object.
<i>std::invalid_argument</i>	If the <a href="#">Shader</a> source is NULL.
<i>gla::ShaderCompileError</i>	If the <a href="#">Shader</a> fails to compile.

### 9.5.2.4 Shader() [4/8]

```
gla::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 <a href="#">ShaderType</a> is invalid.
<i>std::runtime_error</i>	If OpenGL failed to create a <a href="#">Shader</a> object.
<i>std::invalid_argument</i>	If the <a href="#">Shader</a> source is NULL.
<i>gla::ShaderCompileError</i>	If the <a href="#">Shader</a> 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

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

<code>std::logic_error</code>	If the given <code>ShaderType</code> is invalid.
<code>std::runtime_error</code>	If OpenGL failed to create a <code>Shader</code> object.
<code>std::invalid_argument</code>	If the istream is not good().
<code>std::invalid_argument</code>	If in fails to read the <code>Shader</code> source.
<code>std::invalid_argument</code>	If the given input stream is invalid.
<code>std::invalid_argument</code>	If the <code>Shader</code> source is NULL.
<code>gla::ShaderCompileError</code>	If the <code>Shader</code> 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 <a href="#">Shader</a> source is NULL.
<i>std::logic_error</i>	If the current <a href="#">Shader</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state).
<i>gla::ShaderCompileError</i>	If the <a href="#">Shader</a> 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 <a href="#">Shader</a> source is NULL.
<i>std::logic_error</i>	If the current <a href="#">Shader</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state).
<i>gla::ShaderCompileError</i>	If the <a href="#">Shader</a> 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 <a href="#">Shader</a> source.
<i>std::invalid_argument</i>	If the <a href="#">Shader</a> source is NULL.
<i>std::logic_error</i>	If the current <a href="#">Shader</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state).
<i>gla::ShaderCompileError</i>	If the <a href="#">Shader</a> 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 <a href="#">Shader</a> source.
<i>std::invalid_argument</i>	If the <a href="#">Shader</a> source is NULL.
<i>std::logic_error</i>	If the current <a href="#">Shader</a> object does not exist ( <a href="#">reset()</a> is recommended to return to a valid state).
<a href="#">gla::ShaderCompileError</a>	If the <a href="#">Shader</a> 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 <a href="#">Shader</a> 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:

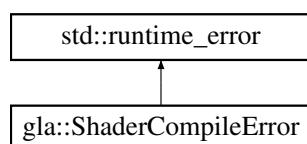
- inc/GLA/shader.h

## 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

<code>ShaderType</code>	The type of <code>Shader</code> that failed to compile.
<code>infoLog</code>	InfoLog buffer from <code>glGetShaderInfoLog</code> .

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

### 9.8.2.2 operator glm::ivec2()

```
gla::Program::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:

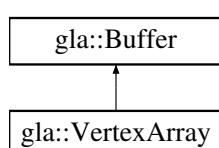
- inc/GLA/program.h

## 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 > & attrs,
    int stride)
```

Set the Attributes for a vertex array.

#### Exceptions

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

#### Note

Calling this function binds this `Buffer`.

#### Parameters

<code>attrs</code>	Vector of Attributes to assign to the <code>VertexArray</code>
--------------------	--

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

- inc/GLA/`vertexArray.h`

## 9.11 gla::VertexAttrib Struct Reference

Defines a vertex attribute for the `gla::VertexArray`.

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

## Data Fields

- `unsigned int index`  
*VertexAttrib location in the GLSL shader program.*
- `int numComponents`  
*number of components for `VertexAttrib` may be [1;4]. For example 3 for vec3.*
- `VertexAttribType type`  
*Type of the `VertexAttrib`.*
- `VertexAttribInterp interp`  
*Interpretation of the `VertexAttrib`, 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 `VertexAttrib`.*

### 9.11.1 Detailed Description

Defines a vertex attribute for the `gla::VertexArray`.

### 9.11.2 Field Documentation

#### 9.11.2.1 `index`

```
unsigned int gla::VertexAttrib::index
```

`VertexAttrib` location in the GLSL shader program.

#### 9.11.2.2 `interp`

```
VertexAttribInterp gla::VertexAttrib::interp
```

Interpretation of the `VertexAttrib`, for example is type Byte is specified, but should be used as a float.

#### 9.11.2.3 `normalized`

```
bool gla::VertexAttrib::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 gla::VertexAttrib::numComponents
```

number of components for `VertexAttrib` may be [1;4]. For example 3 for vec3.

### 9.11.2.5 offset

`int gla::VertexAttrib::offset`

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

### 9.11.2.6 type

`VertexAttribType gla::VertexAttrib::type`

Type of the [VertexAttrib](#).

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

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

#### Parameters

<code>width</code>	Width of the Window in pixels.
<code>height</code>	Height of the Window in pixels.
<code>windowName</code>	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 <cstdint>
#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 flags 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 <cstdint>
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
00103 enum class BufferFlag : uint32_t {
00104     None          = 0,
00105     DynamicStorage = 1 << 0,
00106     MapRead        = 1 << 1,
00107     MapWrite        = 1 << 2,
00108     MapPersistent   = 1 << 3,
00109     MapCoherent     = 1 << 4,
00110     ClientStorage   = 1 << 5
00111 };
00112
00113 inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
00114     return static_cast<BufferFlag>(
00115         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00116     );
00117 }
00118
00119 inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
00120     return static_cast<BufferFlag>(
00121         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00122     );
00123 }
00124
00125 inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
00126     a = a | b;
00127     return a;
00128 }
00129
00130 unsigned int toGLenum(BufferType type);
00131
00132
00133 unsigned int toGLenum(BufferUsage usage);
00134
00135 unsigned int toGLenum(MapUsage usage);
00136
00137 unsigned int toGLenum(BufferFlag flag);
00138
00139
00140 bool validateMapUsage(MapUsage usage, std::string& error);
00141
00142 bool validateBufferFlag(BufferFlag flag, std::string& error);
00143
00144
00145 class Buffer {
00146 protected:
00147     unsigned int _id = 0;
00148     bool _mapped = false;
00149     MapUsage _mapUsage = MapUsage::None;
00150     BufferFlag _flags = BufferFlag::None;
00151     BufferType _type;
00152
00153     void _delete();
00154     void _check();
00155
00156 public:
00157     Buffer() = delete;
00158     Buffer(BufferType type);
00159     Buffer(Buffer&& other);
00160     Buffer(const Buffer& other) = delete;
00161     ~Buffer() noexcept;
00162
00163     void bind() const;
00164
00165     int64_t size() const;
00166
00167     BufferType getType() const;
00168 }
```

```

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

```

## 10.3 inc/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:
00021     ProgramLinkError(const std::string& infoLog)
00022         : std::runtime_error(
00023             "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     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; }
00090         explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
00091             return val; }
00092         explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
00093             return val; }
00094         explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
00095         explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
00096             return val; }
00097         explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
00098             return val; }
00099         explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
00100             return val; }
00101         explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val);
00102             return val; }
00103         explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val);
00104             return val; }
00105         explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val);
00106             return val; }
00107         explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val);
00108             return val; }
00109         explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
00110             val; }
00111         explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
00112             val; }
00113         explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
00114             val; }
00115         explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
00116             return val; }
00117         explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
00118             return val; }
00119         explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
00120             return val; }
00121         explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
00122             return val; }
00123         explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
00124             return val; }
00125         explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
00126             return val; }
00127
00128         UniformProxy& operator=(float other) { _parent.setUniform(_location, other); return *this; }
00129         UniformProxy& operator=(const glm::vec2& other) { _parent.setUniform(_location, other); return
00130             *this; }
00131         UniformProxy& operator=(const glm::vec3& other) { _parent.setUniform(_location, other); return
00132             *this; }
00133         UniformProxy& operator=(const glm::vec4& other) { _parent.setUniform(_location, other); return
00134             *this; }
00135         UniformProxy& operator=(int other) { _parent.setUniform(_location, other); return *this; }
00136         UniformProxy& operator=(const glm::ivec2& other) { _parent.setUniform(_location, other);
00137             return *this; }
00138         UniformProxy& operator=(const glm::ivec3& other) { _parent.setUniform(_location, other);
00139             return *this; }
00140         UniformProxy& operator=(const glm::ivec4& other) { _parent.setUniform(_location, other);
00141             return *this; }
00142         UniformProxy& operator=(unsigned int other) { _parent.setUniform(_location, other); return
00143             *this; }
00144         UniformProxy& operator=(const glm::uvec2& other) { _parent.setUniform(_location, other);
00145             return *this; }
00146         UniformProxy& operator=(const glm::uvec3& other) { _parent.setUniform(_location, other);
00147             return *this; }
00148         UniformProxy& operator=(const glm::uvec4& other) { _parent.setUniform(_location, other);
00149             return *this; }
00150         UniformProxy& operator=(const glm::mat2& other) { _parent.setUniform(_location, other); return
00151             *this; }
00152         UniformProxy& operator=(const glm::mat3& other) { _parent.setUniform(_location, other); return
00153             *this; }
00154         UniformProxy& operator=(const glm::mat4& other) { _parent.setUniform(_location, other); return
00155             *this; }
00156         UniformProxy& operator=(const glm::mat2x3& other) { _parent.setUniform(_location, other);
00157             return *this; }
00158         UniformProxy& operator=(const glm::mat3x2& other) { _parent.setUniform(_location, other);
00159             return *this; }
00160         UniformProxy& operator=(const glm::mat2x4& other) { _parent.setUniform(_location, other);
00161             return *this; }
00162         UniformProxy& operator=(const glm::mat4x2& other) { _parent.setUniform(_location, other);
00163             return *this; }
00164         UniformProxy& operator=(const glm::mat3x4& other) { _parent.setUniform(_location, other);
00165             return *this; }
00166         UniformProxy& operator=(const glm::mat4x3& other) { _parent.setUniform(_location, other);
00167             return *this; }
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
00999

```

```

        return *this; }
00128     UniformProxy& operator=(const glm::mat3x4& other) { _parent.setUniform(_location, other);
00129     return *this; }
00130     UniformProxy& operator=(const glm::mat4x3& other) { _parent.setUniform(_location, other);
00131     return *this; }
00132     UniformProxy& operator=(const UniformProxy& other) = delete;
00133     UniformProxy& operator=(UniformProxy&& other) = delete;
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     {}
00143     UniformProxyConst(UniformProxyConst&& other) = delete;
00144     UniformProxyConst(const UniformProxyConst& other) = delete;
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);
00147     return val; }
00148     explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val);
00149     return val; }
00150     explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val);
00151     return val; }
00152     explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val);
00153     return val; }
00154     explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val);
00155     return val; }
00156     explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val);
00157     return val; }
00158     explicit operator glm::mat2() const { glm::mat2 val; _parent.getUniform(_location, val); return
00159     val; }
00160     explicit operator glm::mat3() const { glm::mat3 val; _parent.getUniform(_location, val); return
00161     val; }
00162     explicit operator glm::mat4() const { glm::mat4 val; _parent.getUniform(_location, val); return
00163     val; }
00164     explicit operator glm::mat2x3() const { glm::mat2x3 val; _parent.getUniform(_location, val);
00165     return val; }
00166     explicit operator glm::mat3x2() const { glm::mat3x2 val; _parent.getUniform(_location, val);
00167     return val; }
00168     explicit operator glm::mat2x4() const { glm::mat2x4 val; _parent.getUniform(_location, val);
00169     return val; }
00170     explicit operator glm::mat4x2() const { glm::mat4x2 val; _parent.getUniform(_location, val);
00171     return val; }
00172     explicit operator glm::mat3x4() const { glm::mat3x4 val; _parent.getUniform(_location, val);
00173     return val; }
00174     explicit operator glm::mat4x3() const { glm::mat4x3 val; _parent.getUniform(_location, val);
00175     return val; }
00176     UniformProxyConst& operator=(const UniformProxyConst& other) = delete;
00177     UniformProxyConst& operator=(UniformProxyConst&& other) = delete;
00178 };
00179
00180 public:
00181     Program();
00182     Program(Program&& other);
00183     Program(const Program&) = delete; // OpenGL Programs are not copy safe
00184     ~Program();
00185
00186     void reset();
00187
00188     bool attached(const Shader& shader);
00189
00190     void attach(const Shader& shader);
00191
00192     void detach(const Shader& shader);
00193
00194     bool linked() const { return _linked; }
00195
00196     void link();
00197
00198     void bind() const;
00199
00200     static void unbind();

```

```
00253
00264     int getUniformLocation(const std::string& name) const;
00265
00282     void setUniform(int location, float data);
00283     void setUniform(int location, const glm::vec2& data);
00284     void setUniform(int location, const glm::vec3& data);
00285     void setUniform(int location, const glm::vec4& data);
00286     void setUniform(int location, int data);
00287     void setUniform(int location, const glm::ivec2& data);
00288     void setUniform(int location, const glm::ivec3& data);
00289     void setUniform(int location, const glm::ivec4& data);
00290     void setUniform(int location, unsigned int data);
00291     void setUniform(int location, const glm::uvec2& data);
00292     void setUniform(int location, const glm::uvec3& data);
00293     void setUniform(int location, const glm::uvec4& data);
00294     void setUniform(int location, const glm::mat2& data);
00295     void setUniform(int location, const glm::mat3& data);
00296     void setUniform(int location, const glm::mat4& data);
00297     void setUniform(int location, const glm::mat2x3& data);
00298     void setUniform(int location, const glm::mat3x2& data);
00299     void setUniform(int location, const glm::mat2x4& data);
00300     void setUniform(int location, const glm::mat4x2& data);
00301     void setUniform(int location, const glm::mat3x4& data);
00302     void setUniform(int location, const glm::mat4x3& data);
00319     void setUniform(const std::string& name, float data) { setUniform(getUniformLocation(name), data); }
00320   }
00320     void setUniform(const std::string& name, const glm::vec2& data) {
00321       setUniform(getUniformLocation(name), data); }
00321     void setUniform(const std::string& name, const glm::vec3& data) {
00322       setUniform(getUniformLocation(name), data); }
00322     void setUniform(const std::string& name, const glm::vec4& data) {
00323       setUniform(getUniformLocation(name), data); }
00323     void setUniform(const std::string& name, int data) { setUniform(getUniformLocation(name), data); }
00324     void setUniform(const std::string& name, const glm::ivec2& data) {
00325       setUniform(getUniformLocation(name), data); }
00325     void setUniform(const std::string& name, const glm::ivec3& data) {
00326       setUniform(getUniformLocation(name), data); }
00326     void setUniform(const std::string& name, const glm::ivec4& data) {
00327       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(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(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(const std::string& name, const glm::mat2x4& data) {
00337         setUniform(const std::string& name, const glm::mat4x2& data) {
00338           setUniform(const std::string& name, const glm::mat3x4& data) {
00339             setUniform(const std::string& name, const glm::mat4x3& data) {
00340               setUniform(getUniformLocation(name), data); }
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 getUniform(const std::string& name, float& data) const { getUniform(getUniformLocation(name), data); }
00391     void getUniform(const std::string& name, glm::vec2& data) const {
00392         getUniform(getUniformLocation(name), data);
00393     void getUniform(const std::string& name, glm::vec3& data) const {
00394         getUniform(getUniformLocation(name), data);
00395     void getUniform(const std::string& name, glm::vec4& data) const {
00396         getUniform(getUniformLocation(name), data);
00397     void getUniform(const std::string& name, glm::ivec2& data) const {
00398         getUniform(getUniformLocation(name), data);
00399     void getUniform(const std::string& name, glm::ivec3& data) const {
00400         getUniform(getUniformLocation(name), data);
00401     void getUniform(const std::string& name, glm::ivec4& data) const {
00402         getUniform(getUniformLocation(name), data);
00403     void getUniform(const std::string& name, unsigned int& data) const {
00404         getUniform(getUniformLocation(name), data);
00405     void getUniform(const std::string& name, glm::uvec2& data) const {
00406         getUniform(getUniformLocation(name), data);
00407     void getUniform(const std::string& name, glm::uvec3& data) const {
00408         getUniform(getUniformLocation(name), data);
00409     void getUniform(const std::string& name, glm::uvec4& data) const {
00410         getUniform(getUniformLocation(name), data);
00411     void getUniform(const std::string& name, glm::mat2& data) const {
00412         getUniform(getUniformLocation(name), data);
00413     void getUniform(const std::string& name, glm::mat3& data) const {
00414         getUniform(getUniformLocation(name), data);
00415     void getUniform(const std::string& name, glm::mat4& data) const {
00416         getUniform(getUniformLocation(name), data);
00417     void getUniform(const std::string& name, glm::mat2x3& data) const {
00418         getUniform(getUniformLocation(name), data);
00419     void getUniform(const std::string& name, glm::mat3x2& data) const {
00420         getUniform(getUniformLocation(name), data);
00421     void getUniform(const std::string& name, glm::mat2x4& data) const {
00422         getUniform(getUniformLocation(name), data);
00423     void getUniform(const std::string& name, glm::mat4x2& data) const {
00424         getUniform(getUniformLocation(name), data);
00425     void getUniform(const std::string& name, glm::mat3x4& data) const {
00426         getUniform(getUniformLocation(name), data);
00427     void getUniform(const std::string& name, glm::mat4x3& data) const {
00428         getUniform(getUniformLocation(name), data);
00429     Program& operator=(Program&& other);
00430     Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe
00431
00432     UniformProxy operator[](int location) { return UniformProxy(*this, location); }
00433     UniformProxy operator[](const std::string& name) { return UniformProxy(*this,
00434         getUniformLocation(name)); }
00435     UniformProxyConst operator[](int location) const { return UniformProxyConst(*this, location); }
00436     UniformProxyConst operator[](const std::string& name) const { return UniformProxyConst(*this,
00437         getUniformLocation(name)); }
00438 };
00439
00440
00441
00442
00443
00444
00445 #endif
```

## 10.8 inc/GLA/shader.h File Reference

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

## 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 <iostream> // std::istream forward-declared
00007
00008 namespace gla {
00009
00010 class Program;
00011
00015 enum class ShaderType {
00016     Fragment,
00017     Vertex,
00018     Geometry,
00019     TessEvaluation,
00020     TessControl,
00021     Compute
00022 };
00023
00030 constexpr std::string shaderTypeToString(ShaderType type);
00031
00040 unsigned int toGLenum(ShaderType type);
00041
00045 class ShaderCompileError : public std::runtime_error {
00046 public:
00053     ShaderCompileError(ShaderType ShaderType,
00054             const std::string& infoLog)
00055         : std::runtime_error(
00056             "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
00057 };
00058
00068 class Shader {
00069 protected:
00070     unsigned int _id = 0;
00071     ShaderType _type;
00072     bool _compiled = false;
00073
00074     void _delete();
00075     void _check();
00076     void _ensure();
00077     std::string _getError();
00078
00079 public:
00080     Shader() = delete;
00081     Shader(ShaderType type);
00082     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::VertexAttrib](#)  
*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
00031 unsigned int toGLenum(VertexAttribType type);
00032
00033 bool validateTypeInterpretation(VertexAttribType type, VertexAttribInterp interp, std::string& error);
00034
00035 int typeToBytes(VertexAttribType type);
00036
00037
00038 struct VertexAttribute {
00039     unsigned int index;
00040     int numComponents;
00041     VertexAttribType type;
00042     VertexAttribInterp interp;
00043     bool normalized;
00044     int offset;
00045 };
00046
00047
00048 class VertexArray : public Buffer {
00049 private:
00050     std::vector<unsigned int> _enabledVertexAttrs = {};
00051
00052 public:
00053     VertexArray() : Buffer(BufferType::Array) {}
00054     VertexArray(VertexArray&& other) : Buffer(std::move(other)) {}
00055     VertexArray(const VertexArray& other) = delete;
00056
00057     void setAttributes(const std::vector<VertexAttribute>& attrs, int stride);
00058
00059     VertexArray& operator=(VertexArray&& other) { Buffer::operator=(std::move(other)); return *this; }
00060     VertexArray& operator=(const VertexArray& other) = delete;
00061 };
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
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::VertexAttrib**, 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::VertexAttrib, 87  
 initGLFW  
     gla, 21  
 Int  
     gla, 20  
 Integer  
     gla, 20  
 interp  
     gla::VertexAttrib, 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::VertexAttrib, 87  
 numComponents  
     gla::VertexAttrib, 87  
  
 offset  
     gla::VertexAttrib, 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::VertexAttrib, 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