OpenGL Abstraction

Generated by Doxygen 1.16.1

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

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Directory Documentation

7.1 inc/GLA Directory Reference

Files

- file buffer.h
- file debug.h
- file program.h
- file shader.h
- file vertexArray.h
- file windowContext.h

7.2 inc Directory Reference

Directories

- directory GLA

Chapter 8

Namespace Documentation

8.1 gla Namespace Reference

Data Structures

- class Buffer
- class ProgramLinkError

  Exception thrown when Program linking fails.
- class ProgramValidateError

  Exception thrown when Program validation fails, mainly used in debug builds.
- struct UniformData
- class Program

  Program class to abstract OpenGL Shader Programs.
- class ShaderCompileError

  Exception thrown when Shader compilation fails.
- class Shader

  Shader class to abstract OpenGL Shaders.
- struct VertexAttribute

  Defines a vertex attribute for the gla::VertexArray.
- class VertexArray

  VertexArray class to abstract the OpenGL vertex array.
- class WindowContext

  An abstract class to contain an window based application.

Enumerations

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

  Enum to indicate the type of Buffer.
- enum class BufferUsage {
  StreamDraw , StreamRead , StreamCopy , StaticDraw ,
  StaticRead , StaticCopy , DynamicDraw , DynamicRead ,
  DynamicCopy }

Enum to indicate Buffer Usage.

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

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

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

  ShaderType enum to indicate usage.
- enum class VertexAttribType {
Byte , UnsignedByte , Short , UnsignedShort ,
Int , UnsignedInt , HalfFloat , Float ,
Double , Fixed }
- enum class VertexAttribInterp { Float , Integer }

Functions

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

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

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

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

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

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

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

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

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

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

  Converts the ShaderType enum into an OpenGL enum.
- unsigned int toGLenum (VertexAttribType type)

  Converts a VertexAttribType into a GLenum.
- bool validateTypeInterpretation (VertexAttribType type, VertexAttribInterp interp, std::string &error)

  Checks if the type and interpretation combination is valid.

- int typeToBytes (VertexAttribType type)

  Gets the size of the given type in bytes.
- bool initGLFW ()

  Initialize GLFW.
- void terminateGLFW ()

  Terminates GLFW.
- void pollEvents ()

  Polls GLFW events.

8.1.1 Enumeration Type Documentation

8.1.1.1 BufferFlag

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

Enum falgs for explicit Buffer usage in setStorage.

Enumerator

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

8.1.1.2 BufferType

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

Enum to indicate the type of Buffer.

Enumerator

Array	Vertex attributes.
AtomicCounter	Atomic counter storage.

CopyRead	Buffer copy source.
CopyWrite	Buffer copy destination.
DispatchIndirect	Indirect compute dispatch commands.
DrawIndirect	Indirect command arguments.
ElementArray	Vertex array indices.
PixelPack	Pixel read target.
PixelUnpack	Texture data source.
Query	Query result buffer.
ShaderStorage	Read-write storage for shaders.
Texture	Texture data buffer.
TransformFeedback	Transform feedback buffer.
Uniform	Uniform block storage.

8.1.1.3 BufferUsage

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

Enum to indicate Buffer Usage.

The Usage of a Buffer can be split into two parts as follows:

The frequency of usage may be one of these:

Stream

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

Static

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

Dynamic

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

The nature of usage may be one of these:

Draw

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

Read

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

Copy

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

Enumerator

StreamDraw	
StreamRead	
StreamCopy	
StaticDraw	
StaticRead	
StaticCopy	
DynamicDraw	
DynamicRead	
DynamicCopy	

8.1.1.4 MapUsage

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

Enum flags to indicate Buffer map usage.

Enumerator

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

8.1.1.5 ShaderType

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

ShaderType enum to indicate usage.

Enumerator

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

8.1.1.6 VertexAttribInterp

```
enum class 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 occured.

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

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

err	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

type	The ShaderType to convert to a std::string.

Returns

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

8.1.2.12 terminateGLFW()

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

Terminates GLFW.

Note

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

8.1.2.13 toGLenum() [1/6]

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

Converts a BufferFlag enum into a GLenum.

8.1.2.14 toGLenum() [2/6]

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

Converts a BufferType enum into a GLenum.

Exceptions

std::invalid_argument	If the BufferType is invalid.

8.1.2.15 toGLenum() [3/6]

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

Converts a BufferUsage enum into a GLenum.

Exceptions

std::invalid_argument	If the BufferUsage is invalid.

8.1.2.16 toGLenum() [4/6]

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

Converts a MapUsage enum into a GLenum.

8.1.2.17 toGLenum() [5/6]

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

Converts the ShaderType enum into an OpenGL enum.

Exceptions

std::logic_error	If the ShaderType is invalid.

Parameters

type	The ShaderType to convert to a std::string.

Returns

The resulting OpenGL enum.

8.1.2.18 toGLenum() [6/6]

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

Converts a VertexAttribType into a GLenum.

Exceptions

std::invalid_argument	If the given VertexAttribType is invalid

8.1.2.19 typeToBytes()

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

Gets the size of the given type in bytes.

Exceptions

std::invalid_argument	If the given VertexAttribType is invalid

8.1.2.20 validateBufferFlag()

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

Checks if the given BufferFlag flag combination is valid.

Parameters

flag	BufferFlag flag combination to check
error	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

usage	MapUsage flag combination to check
error	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

error	The error string output

Returns

true if it is valid, false otherwise

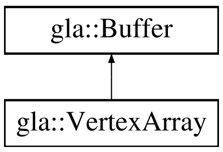
Chapter 9

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

std::runtime_error	If offset is negativ
std::runtime_error	If size is negativ
std::runtime_error	If offset + size is greater than the size of the Buffer
std::runtime_error	If the Buffer is mapped and MapUsage::Persistent is not set

Parameters

offset	The offset of the start of the subset to get in bytes
size	The size of the subset to get in bytes
data	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

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

Parameters

offset	The offset of the map range into the Buffer in bytes
length	The length of the map range in the Buffer in bytes
access	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

std::runtime_error	If size is negative

Parameters

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

9.1.2.10 setData() [2/2]

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

Set the data of the Buffer with a given usage.

Parameters

data	The data to store in the Buffer
usage	The usage hint of the Buffer

9.1.2.11 setStorage() [1/2]

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

Set the data of the Buffer with given Buffer flags.

Exceptions

std::runtime_error	If size is not greater than 0
std::runtime_error	If the BufferFlag combination is invalid

Parameters

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

9.1.2.12 setStorage() [2/2]

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

Set the data of the Buffer with given Buffer flags.

Exceptions

std::runtime_error	If size is not greater than 0
std::runtime_error	If the BufferFlag combination is invalid

Parameters

data	The data to store in the Buffer
flags	The Buffer usage flags

9.1.2.13 setSubData()

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

Set a subset of the data in the Buffer.

Exceptions

std::runtime_error	If offset is negativ
std::runtime_error	If size is negativ
std::runtime_error	If offset + size is greater than the size of the Buffer
std::runtime_error	If the Buffer is mapped and MapUsage::Persistent is not set

Parameters

offset	The offset of the start of the subset to set in bytes
size	The size of the subset to set in bytes

data	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

std::runtime_error	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)

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

- void setUniform (const std::string &name, int data)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void setUniform (const std::string &name, const glm::ivec2 &data)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Gets a uniform at the given location.

- void getUniform (int location, glm::vec2 &data) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void getUniform (int location, glm::vec3 &data) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void getUniform (int location, glm::vec4 &data) const

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- void getUniform (int location, glm::mat3x4 &data) const

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

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

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

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void getUniform (const std::string &name, glm::vec3 &data) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void getUniform (const std::string &name, glm::vec4 &data) const

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Program & operator= (Program &&other)
- Program & operator= (const Program &)=delete
- UniformProxy operator[ ] (int location)

  Returns a Proxy to get / set the uniform at the given location with assignment and type conversion operators.
- UniformProxy operator[ ] (const std::string &name)

  Returns a Proxy to get / set the uniform with the given name with assignment and type conversion operators.
- UniformProxyConst operator[ ] (int location) const

  Returns a constant only Proxy to get the uniform at the given location with type conversion operators.
- UniformProxyConst operator[ ] (const std::string &name) const

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

Static Public Member Functions

- static void unbind ()

  Unbinds a Program by binding id 0.

Protected Member Functions

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

Protected Attributes

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

9.2.1 Detailed Description

Program class to abstract OpenGL Shader Programs.

Warning

Program must be deconstructed before the OpenGL context is destroyed.

This class is not guaranteed to be thread-safe.

Note

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

9.2.2 Constructor & Destructor Documentation

9.2.2.1 Program() [1/3]

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

Construct an empty Program object.

9.2.2.2 Program() [2/3]

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

9.2.2.3 Program() [3/3]

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

9.2.2.4 ∼Program()

```
gla::Program::∼Program ()
```

9.2.3 Member Function Documentation

9.2.3.1 _check()

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

9.2.3.2 _delete()

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

9.2.3.3 _ensure()

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

9.2.3.4 _getError()

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

9.2.3.5 _queryUniformData()

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

9.2.3.6 _setupUniform()

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

9.2.3.7 attach()

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

Attaches Shader to Program for linking.

Note

Attaching a Shader invalidates the current link state.

Warning

Attaching a destroyed Shader is undefined behavior.

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

Exceptions

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

Parameters

shader	The Shader that should be attached to the Program.

9.2.3.8 attached()

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

Checks if the given Shader is attached to the Program.

Parameters

shader	The Shader to check.

Exceptions

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

Returns

true if the Shader is attached, false otherwise.

9.2.3.9 bind()

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

Binds this Program.

Exceptions

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

9.2.3.10 detach()

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

Detaches Shader from Program for linking.

Note

Detaching a Shader invalidates the current link state.

Exceptions

std::runtime_error	If the Shader was not attached.

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

Parameters

shader	The Shader that should be detached from the Program.

9.2.3.11 getUniform() [1/42]

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

Gets a uniform at the given name.

Exceptions

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

Parameters

name	The uniform name
data	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

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

Parameters

location	The uniform location returned by getUniformLocation()
data	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

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

Parameters

name	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

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

9.2.3.55 linked()

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

Gets if the Program has been successfully linked.

Returns

true if the Program is linked, false otherwise.

9.2.3.56 operator=() [1/2]

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

9.2.3.57 operator=() [2/2]

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

9.2.3.58 operator[]() [1/4]

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

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

Note

Possible errors when assigning can be found under gla::Program::setUniform.

Possible errors on explicit type conversion can be found under gla::Program::getUniform.

Warning

Setting a uniform binds the program.

Exceptions

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

Parameters

name	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

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

Parameters

name	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

location	The uniform location returned by getUniformLocation()

9.2.3.61 operator[]() [4/4]

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

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

Note

Possible errors on explicit type conversion can be found under gla::Program::getUniform.

Parameters

location	The uniform location returned by getUniformLocation()

9.2.3.62 reset()

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

Resets the Program to an empty State.

Exceptions

std::runtime_error	If OpenGL failed to create a new Program.

9.2.3.63 setUniform() [1/42]

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

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

9.2.3.64 setUniform() [2/42]

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

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

9.2.3.65 setUniform() [3/42]

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

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

9.2.3.66 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

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

Warning

Setting a uniform binds the program.

Parameters

name	The uniform name
data	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

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

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

Warning

Setting a uniform binds the program.

Parameters

location	The uniform location returned by getUniformLocation()
data	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:

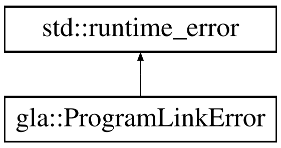
- inc/GLA/program.h

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

infoLog	InfoLog buffer from glGetProgramInfoLog.

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

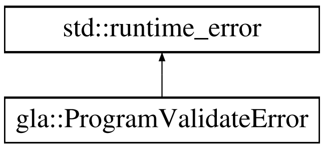
- inc/GLA/program.h

9.4 gla::ProgramValidateError Class Reference

Exception thrown when Program validation fails, mainly used in debug builds.

`#include <program.h>`

Inheritance diagram for gla::ProgramValidateError:



Public Member Functions

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

9.4.1 Detailed Description

Exception thrown when Program validation fails, mainly used in debug builds.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 ProgramValidateError()

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

Construct a new Program Validate Error object.

Parameters

infoLog	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

std::logic_error	If the given ShaderType is invalid.
std::runtime_error	If OpenGL failed to create a Shader 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

std::logic_error	If the given ShaderType is invalid.
std::runtime_error	If OpenGL failed to create a Shader object.
std::invalid_argument	If the Shader source is NULL.
gla::ShaderCompileError	If the Shader 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

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

9.5.2.5 Shader() [5/8]

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

Constructs and compiles a new Shader object of given type.

Exceptions

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

9.5.2.6 Shader() [6/8]

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

Constructs and compiles a new Shader object of given type.

Exceptions

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

9.5.2.7 Shader() [7/8]

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

9.5.2.8 Shader() [8/8]

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

9.5.2.9 ∼Shader()

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

9.5.3 Member Function Documentation

9.5.3.1 _check()

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

9.5.3.2 _delete()

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

9.5.3.3 _ensure()

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

9.5.3.4 _getError()

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

9.5.3.5 compile() [1/4]

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

Compiles the Shader with the given source.

Exceptions

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

Parameters

src	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

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

Parameters

str	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

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

Parameters

in	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

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

Parameters

in	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

std::runtime_error	If OpenGL failed to create a new Shader object.

9.5.4 Field Documentation

9.5.4.1 _compiled

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

9.5.4.2 _id

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

9.5.4.3 _type

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

9.5.4.4 Program

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

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

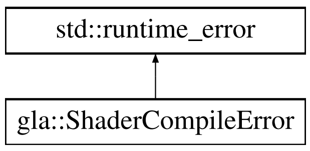
- inc/GLA/shader.h

9.6 gla::ShaderCompileError Class Reference

Exception thrown when Shader compilation fails.

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

Inheritance diagram for gla::ShaderCompileError:



Public Member Functions

- ShaderCompileError (ShaderType ShaderType, const std::string &infoLog)

  Construct a new Shader Compile Error object.

9.6.1 Detailed Description

Exception thrown when Shader compilation fails.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 ShaderCompileError()

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

Construct a new Shader Compile Error object.

Parameters

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

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

- inc/GLA/shader.h

9.7 gla::UniformData Struct Reference

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

Data Fields

- int location
- unsigned int glType
- int arraySize

9.7.1 Field Documentation

9.7.1.1 arraySize

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

9.7.1.2 glType

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

9.7.1.3 location

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

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

- inc/GLA/program.h

9.8 gla::Program::UniformProxy Class Reference

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

Public Member Functions

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

- UniformProxy & operator= (const glm::ivec4 &other)
- UniformProxy & operator= (unsigned int other)
- UniformProxy & operator= (const glm::uvec2 &other)
- UniformProxy & operator= (const glm::uvec3 &other)
- UniformProxy & operator= (const glm::uvec4 &other)
- UniformProxy & operator= (const glm::mat2 &other)
- UniformProxy & operator= (const glm::mat3 &other)
- UniformProxy & operator= (const glm::mat4 &other)
- UniformProxy & operator= (const glm::mat2x3 &other)
- UniformProxy & operator= (const glm::mat3x2 &other)
- UniformProxy & operator= (const glm::mat2x4 &other)
- UniformProxy & operator= (const glm::mat4x2 &other)
- UniformProxy & operator= (const glm::mat3x4 &other)
- UniformProxy & operator= (const glm::mat4x3 &other)
- UniformProxy & operator= (const UniformProxy &other)=delete
- UniformProxy & operator= (UniformProxy &&other)=delete

9.8.1 Constructor & Destructor Documentation

9.8.1.1 UniformProxy() [1/3]

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

9.8.1.2 UniformProxy() [2/3]

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

9.8.1.3 UniformProxy() [3/3]

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

9.8.2 Member Function Documentation

9.8.2.1 operator float()

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

9.8.2.2 operator glm::ivec2()

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

9.8.2.3 operator glm::ivec3()

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

9.8.2.4 operator glm::ivec4()

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

9.8.2.5 operator glm::mat2()

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

9.8.2.6 operator glm::mat2x3()

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

9.8.2.7 operator glm::mat2x4()

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

9.8.2.8 operator glm::mat3()

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

9.8.2.9 operator glm::mat3x2()

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

9.8.2.10 operator glm::mat3x4()

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

9.8.2.11 operator glm::mat4()

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

9.8.2.12 operator glm::mat4x2()

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

9.8.2.13 operator glm::mat4x3()

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

9.8.2.14 operator glm::uvec2()

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

9.8.2.15 operator glm::uvec3()

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

9.8.2.16 operator glm::uvec4()

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

9.8.2.17 operator glm::vec2()

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

9.8.2.18 operator glm::vec3()

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

9.8.2.19 operator glm::vec4()

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

9.8.2.20 operator int()

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

9.8.2.21 operator unsigned int()

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

9.8.2.22 operator=() [1/23]

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

9.8.2.23 operator=() [2/23]

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

9.8.2.24 operator=() [3/23]

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

9.8.2.25 operator=() [4/23]

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

9.8.2.26 operator=() [5/23]

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

9.8.2.27 operator=() [6/23]

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

9.8.2.28 operator=() [7/23]

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

9.8.2.29 operator=() [8/23]

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

9.8.2.30 operator=() [9/23]

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

9.8.2.31 operator=() [10/23]

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

9.8.2.32 operator=() [11/23]

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

9.8.2.33 operator=() [12/23]

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

9.8.2.34 operator=() [13/23]

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

9.8.2.35 operator=() [14/23]

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

9.8.2.36 operator=() [15/23]

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

9.8.2.37 operator=() [16/23]

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

9.8.2.38 operator=() [17/23]

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

9.8.2.39 operator=() [18/23]

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

9.8.2.40 operator=() [19/23]

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

9.8.2.41 operator=() [20/23]

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

9.8.2.42 operator=() [21/23]

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

9.8.2.43 operator=() [22/23]

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

9.8.2.44 operator=() [23/23]

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

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

- inc/GLA/program.h

9.9 gla::Program::UniformProxyConst Class Reference

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

Public Member Functions

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

9.9.1 Constructor & Destructor Documentation

9.9.1.1 UniformProxyConst() [1/3]

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

9.9.1.2 UniformProxyConst() [2/3]

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

9.9.1.3 UniformProxyConst() [3/3]

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

9.9.2 Member Function Documentation

9.9.2.1 operator float()

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

9.9.2.2 operator glm::ivec2()

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

9.9.2.3 operator glm::ivec3()

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

9.9.2.4 operator glm::ivec4()

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

9.9.2.5 operator glm::mat2()

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

9.9.2.6 operator glm::mat2x3()

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

9.9.2.7 operator glm::mat2x4()

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

9.9.2.8 operator glm::mat3()

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

9.9.2.9 operator glm::mat3x2()

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

9.9.2.10 operator glm::mat3x4()

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

9.9.2.11 operator glm::mat4()

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

9.9.2.12 operator glm::mat4x2()

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

9.9.2.13 operator glm::mat4x3()

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

9.9.2.14 operator glm::uvec2()

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

9.9.2.15 operator glm::uvec3()

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

9.9.2.16 operator glm::uvec4()

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

9.9.2.17 operator glm::vec2()

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

9.9.2.18 operator glm::vec3()

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

9.9.2.19 operator glm::vec4()

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

9.9.2.20 operator int()

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

9.9.2.21 operator unsigned int()

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

9.9.2.22 operator=() [1/2]

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

9.9.2.23 operator=() [2/2]

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

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

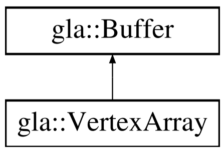
- inc/GLA/program.h

9.10 gla::VertexArray Class Reference

VertexArray class to abstract the OpenGL vertex array.

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

Inheritance diagram for gla::VertexArray:



Public Member Functions

- VertexArray ()
- VertexArray (VertexArray &&other)
- VertexArray (const VertexArray &other)=delete
- void setAttributes (const std::vector< VertexAttribute > &attribs, int stride)

  Set the Attributes for a vertex array.
- VertexArray & operator= (VertexArray &&other)
- VertexArray & operator= (const VertexArray &other)=delete

Public Member Functions inherited from gla::Buffer

- Buffer ()=delete
- Buffer (BufferType type)

  Construct a new Buffer object of given type.
- Buffer (Buffer &&other)
- Buffer (const Buffer &other)=delete
- ∼Buffer () noexcept
- void bind () const

  Binds the Buffer to the appropriate binding point.
- int64_t size () const

  Returns the size in bytes of the Buffer.
- BufferType getType () const

  Get the Type of the Buffer.
- void setData (int64_t size, const void ∗data, BufferUsage usage)

  Set the data of the Buffer with a given usage.
- template<typename T>
  void setData (std::vector< T > data, BufferUsage usage)

  Set the data of the Buffer with a given usage.
- void setStorage (int64_t size, const void ∗data, BufferFlag flags)

  Set the data of the Buffer with given Buffer flags.
- template<typename T>
  void setStorage (std::vector< T > data, BufferFlag flags)

  Set the data of the Buffer with given Buffer flags.
- void setSubData (int64_t offset, int64_t size, const void ∗data)

  Set a subset of the data in the Buffer.
- void getSubData (int64_t offset, int64_t size, void ∗data)

  Get a subset of the data in the Buffer.
- void ∗ map (int64_t offset, int64_t length, MapUsage access)

  Map a part of the Buffer data to the client's address space.
- void unmap ()

  Unmaps the Buffer.
- Buffer & operator= (Buffer &&other)
- Buffer & operator= (const Buffer &other)=delete

Additional Inherited Members

Protected Member Functions inherited from gla::Buffer

- void _delete ()
- void _check ()

Protected Attributes inherited from gla::Buffer

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

9.10.1 Detailed Description

VertexArray class to abstract the OpenGL vertex array.

Warning

Program must be deconstructed before the OpenGL context is destroyed.

This class is not guaranteed to be thread-safe.

Note

Inherits from gla::Buffer.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 VertexArray() [1/3]

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

9.10.2.2 VertexArray() [2/3]

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

9.10.2.3 VertexArray() [3/3]

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

9.10.3 Member Function Documentation

9.10.3.1 operator=() [1/2]

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

9.10.3.2 operator=() [2/2]

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

9.10.3.3 setAttributes()

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

Set the Attributes for a vertex array.

Exceptions

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

Note

Calling this function binds this Buffer.

Parameters

attribs	Vector of Attributes to assign to the VertexArray

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

- inc/GLA/vertexArray.h

9.11 gla::VertexAttribute Struct Reference

Defines a vertex attribute for the gla::VertexArray.

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

Data Fields

- unsigned int index

    VertexAttribute location in the GLSL shader program.
- int numComponents

    number of components for VertexAttribute may be [1;4]. For example 3 for vec3.
- VertexAttribType type

    Type of the VertexAttribute.
- VertexAttribInterp interp

    Interpretation of the VertexAttribute, for example is type Byte is specified, but should be used as a float.
- bool normalized

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

    Offset to the start of the current VertexAttribute.

9.11.1 Detailed Description

Defines a vertex attribute for the gla::VertexArray.

9.11.2 Field Documentation

9.11.2.1 index

```
unsigned int gla::VertexAttribute::index
```

VertexAttribute location in the GLSL shader program.

9.11.2.2 interp

```
VertexAttribInterp gla::VertexAttribute::interp
```

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

9.11.2.3 normalized

```
bool gla::VertexAttribute::normalized
```

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

9.11.2.4 numComponents

```
int gla::VertexAttribute::numComponents
```

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

9.11.2.5 offset

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

Offset to the start of the current VertexAttribute.

9.11.2.6 type

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

Type of the VertexAttribute.

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

- inc/GLA/vertexArray.h

9.12 gla::WindowContext Class Reference

An abstract class to contain an window based application.

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

Public Member Functions

- WindowContext (int width, int height, const char ∗windowName)

  Construct a new Window Context object with given height, width and name.
- WindowContext (WindowContext &&other)
- WindowContext (const WindowContext &other)=delete
- ∼WindowContext ()
- virtual void run ()=0

  Virtual run method to overload when implementing a child class.
- WindowContext & operator= (WindowContext &&other)
- WindowContext & operator= (const WindowContext &other)=delete

Protected Member Functions

- void useContext ()

  Makes the owned window the current context.
- bool shouldClose ()

  Checks if the Window should close.
- void swapBuffers ()

  Swaps the display buffers.

Protected Attributes

- GLFWwindow ∗ window = NULL

  GLFW window handle for low level GLFW access.

9.12.1 Detailed Description

An abstract class to contain an window based application.

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

Note

Calling useContext is recomended 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

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

Parameters

width	Width of the Window in pixels.
height	Height of the Window in pixels.
windowName	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 recomended 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

std::runtime_error	If the GLFW Window is invalid.

9.12.3.5 swapBuffers()

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

Swaps the display buffers.

Exceptions

std::runtime_error	If the GLFW Window is invalid.

9.12.3.6 useContext()

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

Makes the owned window the current context.

Exceptions

std::runtime_error	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 falgs for explicit Buffer usage in setStorage.

Functions

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

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

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

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

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

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

    Checks if the given BufferFlag flag combination is valid.

10.2 buffer.h

Go to the documentation of this file.

```
#ifndef GLA_BUFFER_H
#define GLA_BUFFER_H

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

namespace gla {

enum class BufferType {
    Array,
    AtomicCounter,
    CopyRead,
    CopyWrite,
    DispatchIndirect,
    DrawIndirect,
    ElementArray,
    PixelPack,
    PixelUnpack,
    Query,
    ShaderStorage,
    Texture,
    TransformFeedback,
    Uniform
};

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
};

inline MapUsage operator|(MapUsage a, MapUsage b) {
    return static_cast<MapUsage>(
        static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
    );
}

inline MapUsage operator&(MapUsage a, MapUsage b) {
    return static_cast<MapUsage>(
        static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
    );
}

inline MapUsage& operator|=(MapUsage& a, MapUsage b) {
    a = a | b;
    return a;
}

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
};

inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
    return static_cast<BufferFlag>(
        static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
    );
}

inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
    return static_cast<BufferFlag>(
        static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
    );
}

inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
    a = a | b;
    return a;
}

unsigned int toGLenum(BufferType type);

unsigned int toGLenum(BufferUsage usage);

unsigned int toGLenum(MapUsage usage);

unsigned int toGLenum(BufferFlag flag);

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

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

class Buffer {
protected:
    unsigned int _id = 0;
    bool _mapped = false;
    MapUsage _mapUsage = MapUsage::None;
    BufferFlag _flags = BufferFlag::None;
    BufferType _type;

    void _delete();
    void _check();

public:
    Buffer() = delete;
    Buffer(BufferType type);
    Buffer(Buffer&& other);
    Buffer(const Buffer& other) = delete;
    ~Buffer() noexcept;

    void bind() const;

    int64_t size() const;

    BufferType getType() const;

    void setData(int64_t size, const void* data, BufferUsage usage);

    template <typename T>
    void setData(std::vector<T> data, BufferUsage usage) { setData(data.size() * sizeof(T),
(void*)data.data(), usage); }

    void setStorage(int64_t size, const void* data, BufferFlag flags);

    template <typename T>
    void setStorage(std::vector<T> data, BufferFlag flags) { setStorage(data.size() * sizeof(T),
(void*)data.data(), flags); }

    void setSubData(int64_t offset, int64_t size, const void* data);

    void getSubData(int64_t offset, int64_t size, void* data);

    void* map(int64_t offset, int64_t length, MapUsage access);

    void unmap();

    Buffer& operator=(Buffer&& other);
    Buffer& operator=(const Buffer& other) = delete;
};

}

#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 occured.

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.

```
#ifndef GLA_DEBUG_H
#define GLA_DEBUG_H

namespace gla {

const char* glErrorString(unsigned int err);

void glCheckError(const char* func, const char* file, int line);

};



#ifdef DEBUG_BUILD

    #define GL_CALL(x) do { x; gla::glCheckError(#x, __FILE__, __LINE__); } while(0)

    #define DEBUG_ONLY(x) x

#else

    #define GL_CALL(x) x

    #define DEBUG_ONLY(x) ((void)0)

#endif

#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.

```
#ifndef GLA_PROGRAM_H
#define GLA_PROGRAM_H

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

namespace gla {

class Shader;

class ProgramLinkError : public std::runtime_error {
public:
    ProgramLinkError(const std::string& infoLog)
        : std::runtime_error(
              "Program link failed:\n" + infoLog) {}
};

class ProgramValidateError : public std::runtime_error {
public:
    ProgramValidateError(const std::string& infoLog)
        : std::runtime_error(
              "Program validation failed:\n" + infoLog) {}
};

struct UniformData {
    int location;
    unsigned int glType;
    int arraySize;
};

class Program {
protected:
    unsigned int _id = 0;
    bool _linked = false;

    std::unordered_map<std::string, int> _uniformIndexMap = {}; // name to uniform index conversion
    std::unordered_map<int, int> _uniformLocationIndexMap = {}; // location to uniform index
conversion
    std::vector<UniformData> _uniformData = {}; // uniform data per index

    void _delete();
    void _check() const;
    void _ensure() const;
    void _queryUniformData();
    void _setupUniform(int loc, int sizeCheck, int typeCheck) const;
    std::string _getError();
```

```cpp

    class UniformProxy {
    private:
        Program& _parent;
        int _location;

    public:
        UniformProxy(Program& parent, int location) : _parent(parent), _location(location) {}
        UniformProxy(UniformProxy&& other) = delete;
        UniformProxy(const UniformProxy& other) = delete;

        explicit operator float() const { float val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec2() const { glm::vec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2() const {glm::mat2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3() const {glm::mat3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4() const {glm::mat4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2x3() const {glm::mat2x3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3x2() const {glm::mat3x2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2x4() const {glm::mat2x4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4x2() const {glm::mat4x2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3x4() const {glm::mat3x4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4x3() const {glm::mat4x3 val; _parent.getUniform(_location, val); return val; }

        UniformProxy& operator=(float other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::vec2& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::vec3& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::vec4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(int other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::ivec2& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::ivec3& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::ivec4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(unsigned int other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::uvec2& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::uvec3& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::uvec4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat2& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat3& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat2x3& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat3x2& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat2x4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat4x2& other) { _parent.setUniform(_location, other);
```

```
return *this; }
        UniformProxy& operator=(const glm::mat3x4& other) { _parent.setUniform(_location, other); return *this; }
        UniformProxy& operator=(const glm::mat4x3& other) { _parent.setUniform(_location, other); return *this; }

        UniformProxy& operator=(const UniformProxy& other) = delete;
        UniformProxy& operator=(UniformProxy&& other) = delete;
    };

    class UniformProxyConst {
    private:
        const Program& _parent;
        int _location;

    public:
        UniformProxyConst(const Program& parent, int location) : _parent(parent), _location(location) {}
        UniformProxyConst(UniformProxyConst&& other) = delete;
        UniformProxyConst(const UniformProxyConst& other) = delete;

        explicit operator float() const { float val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec2() const { glm::vec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec3() const { glm::vec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::vec4() const { glm::vec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator int() const { int val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec2() const { glm::ivec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec3() const { glm::ivec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::ivec4() const { glm::ivec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator unsigned int() const { unsigned int val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec2() const { glm::uvec2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec3() const { glm::uvec3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::uvec4() const { glm::uvec4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2() const {glm::mat2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3() const {glm::mat3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4() const {glm::mat4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2x3() const {glm::mat2x3 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3x2() const {glm::mat3x2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat2x4() const {glm::mat2x4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4x2() const {glm::mat4x2 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat3x4() const {glm::mat3x4 val; _parent.getUniform(_location, val); return val; }
        explicit operator glm::mat4x3() const {glm::mat4x3 val; _parent.getUniform(_location, val); return val; }

        UniformProxyConst& operator=(const UniformProxyConst& other) = delete;
        UniformProxyConst& operator=(UniformProxyConst&& other) = delete;
    };

public:
    Program();
    Program(Program&& other);
    Program(const Program&) = delete; // OpenGL Programs are not copy safe
    ~Program();

    void reset();

    bool attached(const Shader& shader);

    void attach(const Shader& shader);

    void detach(const Shader& shader);

    bool linked() const { return _linked; }

    void link();

    void bind() const;

    static void unbind();
    int getUniformLocation(const std::string& name) const;

    void setUniform(int location, float data);
    void setUniform(int location, const glm::vec2& data);
    void setUniform(int location, const glm::vec3& data);
    void setUniform(int location, const glm::vec4& data);
    void setUniform(int location, int data);
    void setUniform(int location, const glm::ivec2& data);
    void setUniform(int location, const glm::ivec3& data);
    void setUniform(int location, const glm::ivec4& data);
    void setUniform(int location, unsigned int data);
    void setUniform(int location, const glm::uvec2& data);
    void setUniform(int location, const glm::uvec3& data);
    void setUniform(int location, const glm::uvec4& data);
    void setUniform(int location, const glm::mat2& data);
    void setUniform(int location, const glm::mat3& data);
    void setUniform(int location, const glm::mat4& data);
    void setUniform(int location, const glm::mat2x3& data);
    void setUniform(int location, const glm::mat3x2& data);
    void setUniform(int location, const glm::mat2x4& data);
    void setUniform(int location, const glm::mat4x2& data);
    void setUniform(int location, const glm::mat3x4& data);
    void setUniform(int location, const glm::mat4x3& data);
    void setUniform(const std::string& name, float data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::vec2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::vec3& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::vec4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, int data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::ivec2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::ivec3& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::ivec4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, unsigned int data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::uvec2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::uvec3& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::uvec4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat3& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat2x3& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat3x2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat2x4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat4x2& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat3x4& data) { setUniform(getUniformLocation(name), data); }
    void setUniform(const std::string& name, const glm::mat4x3& data) { setUniform(getUniformLocation(name), data); }

    void getUniform(int location, float& data) const;
    void getUniform(int location, glm::vec2& data) const;
    void getUniform(int location, glm::vec3& data) const;
    void getUniform(int location, glm::vec4& data) const;
    void getUniform(int location, int& data) const;
    void getUniform(int location, glm::ivec2& data) const;
    void getUniform(int location, glm::ivec3& data) const;
    void getUniform(int location, glm::ivec4& data) const;
    void getUniform(int location, unsigned int& data) const;
    void getUniform(int location, glm::uvec2& data) const;
    void getUniform(int location, glm::uvec3& data) const;
    void getUniform(int location, glm::uvec4& data) const;
    void getUniform(int location, glm::mat2& data) const;
    void getUniform(int location, glm::mat3& data) const;
    void getUniform(int location, glm::mat4& data) const;
    void getUniform(int location, glm::mat2x3& data) const;
    void getUniform(int location, glm::mat3x2& data) const;
    void getUniform(int location, glm::mat2x4& data) const;
    void getUniform(int location, glm::mat4x2& data) const;
    void getUniform(int location, glm::mat3x4& data) const;
    void getUniform(int location, glm::mat4x3& data) const;
    void getUniform(const std::string& name, float& data) const { getUniform(getUniformLocation(name),
data); }
    void getUniform(const std::string& name, glm::vec2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::vec3& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::vec4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, int& data) const { getUniform(getUniformLocation(name),
data); }
    void getUniform(const std::string& name, glm::ivec2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::ivec3& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::ivec4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, unsigned int& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::uvec2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::uvec3& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::uvec4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat3& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat2x3& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat3x2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat2x4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat4x2& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat3x4& data) const {
getUniform(getUniformLocation(name), data); }
    void getUniform(const std::string& name, glm::mat4x3& data) const {
getUniform(getUniformLocation(name), data); }

    Program& operator=(Program&& other);
    Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe

    UniformProxy operator[](int location) { return UniformProxy(*this, location); }
    UniformProxy operator[](const std::string& name) { return UniformProxy(*this,
getUniformLocation(name)); }
    UniformProxyConst operator[](int location) const { return UniformProxyConst(*this, location); }
    UniformProxyConst operator[](const std::string& name) const { return UniformProxyConst(*this,
getUniformLocation(name)); }
};

}

#endif
```

10.8 inc/GLA/shader.h File Reference

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

Data Structures

- class gla::ShaderCompileError

  Exception thrown when Shader compilation fails.
- class gla::Shader

  Shader class to abstract OpenGL Shaders.

Namespaces

- namespace gla

Enumerations

- enum class gla::ShaderType {
  gla::Fragment , gla::Vertex , gla::Geometry , gla::TessEvaluation ,
  gla::TessControl , gla::Compute }

  ShaderType enum to indicate usage.

Functions

- constexpr std::string gla::shaderTypeToString (ShaderType type)

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

  Converts the ShaderType enum into an OpenGL enum.

10.9 shader.h

Go to the documentation of this file.

```
#ifndef GLA_SHADER_H
#define GLA_SHADER_H

#include <string>
#include <stdexcept>
#include <iosfwd> // std::istream forward-declared

namespace gla {

class Program;

enum class ShaderType {
    Fragment,
    Vertex,
    Geometry,
    TessEvaluation,
    TessControl,
    Compute
};

constexpr std::string shaderTypeToString(ShaderType type);

unsigned int toGLenum(ShaderType type);

class ShaderCompileError : public std::runtime_error {
public:
    ShaderCompileError(ShaderType ShaderType,
                       const std::string& infoLog)
        : std::runtime_error(
              "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
};

class Shader {
protected:
    unsigned int _id = 0;
    ShaderType _type;
    bool _compiled = false;

    void _delete();
    void _check();
    void _ensure();
    std::string _getError();

public:
    Shader() = delete;
    Shader(ShaderType type);
    Shader(ShaderType type, const char* src);
    Shader(ShaderType type, const std::string& src);
    Shader(ShaderType type, std::istream& in);
    Shader(ShaderType type, std::istream&& in);

    Shader(Shader&& other);
    Shader(const Shader& other) = delete; // OpenGL Shaders are not copy safe
    ~Shader() noexcept;

    void reset();

    bool compiled() const { return _compiled; }

    ShaderType getType() const { return _type; }

    void compile(const char* src);

    void compile(std::istream& in);

    void compile(std::istream&& in);

    void compile(const std::string& str);

    friend Program;

    Shader& operator=(Shader&& other);
    Shader& operator=(const Shader& other) = delete; // OpenGL Shaders are not copy safe
};

}

#endif
```

10.10 inc/GLA/vertexArray.h File Reference

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

Data Structures

- struct gla::VertexAttribute

  Defines a vertex attribute for the gla::VertexArray.
- class gla::VertexArray

  VertexArray class to abstract the OpenGL vertex array.

Namespaces

- namespace gla

Enumerations

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

Functions

- unsigned int gla::toGLenum (VertexAttribType type)

  Converts a VertexAttribType into a GLenum.
- bool gla::validateTypeInterpretation (VertexAttribType type, VertexAttribInterp interp, std::string &error)

  Checks if the type and interpretation combination is valid.
- int gla::typeToBytes (VertexAttribType type)

  Gets the size of the given type in bytes.

10.11 vertexArray.h

Go to the documentation of this file.

```
#ifndef GLA_VERTEX_ARRAY_H
#define GLA_VERTEX_ARRAY_H

#include <vector>
#include <stdexcept>

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

namespace gla {

enum class VertexAttribType {
    Byte,
    UnsignedByte,
    Short,
    UnsignedShort,
    Int,
    UnsignedInt,
    HalfFloat,
    Float,
    Double,
    Fixed
};

enum class VertexAttribInterp {
    Float,
    Integer
};


unsigned int toGLenum(VertexAttribType type);

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

int typeToBytes(VertexAttribType type);

struct VertexAttribute {
    unsigned int index;
    int numComponents;
    VertexAttribType type;
    VertexAttribInterp interp;
    bool normalized;
    int offset;
};

class VertexArray : public Buffer {
private:
    std::vector<unsigned int> _enabledVertexAttribs = {};

public:
    VertexArray() : Buffer(BufferType::Array) {}
    VertexArray(VertexArray&& other) : Buffer(std::move(other)) {}
    VertexArray(const VertexArray& other) = delete;

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

    VertexArray& operator=(VertexArray&& other) { Buffer::operator=(std::move(other)); return *this; }
    VertexArray& operator=(const VertexArray& other) = delete;
};

}

#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.

```
#ifndef GLA_WINDOW_CONTEXT
#define GLA_WINDOW_CONTEXT

#include <GLFW/glfw3.h>

namespace gla {

bool initGLFW();

void terminateGLFW();

void pollEvents();

class WindowContext {
private:
    bool _ownsGLFW = false;

protected:
    GLFWwindow* window = NULL;

    void useContext();

    bool shouldClose();

    void swapBuffers();

public:
    WindowContext(int width, int height, const char* windowName);
    WindowContext(WindowContext&& other);
    WindowContext(const WindowContext& other) = delete;
    ~WindowContext();

    virtual void run() = 0;

    WindowContext& operator=(WindowContext&& other);
    WindowContext& operator=(const WindowContext& other) = delete;
};

}

#endif
```

Index