

Easy OpenGL

Generated by Doxygen 1.16.1

1 Introduction	1
1.1 Recommended Setup	1
2 Directory Hierarchy	3
2.1 Directories	3
3 Namespace Index	5
3.1 Namespace List	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Data Structure Index	9
5.1 Data Structures	9
6 File Index	11
6.1 File List	11
7 Directory Documentation	13
7.1 inc/EGL Directory Reference	13
7.2 inc Directory Reference	13
8 Namespace Documentation	15
8.1 egl Namespace Reference	15
8.1.1 Enumeration Type Documentation	16
8.1.1.1 BufferFlag	16
8.1.1.2 BufferType	17
8.1.1.3 BufferUsage	17
8.1.1.4 MapUsage	19
8.1.1.5 ShaderType	19
8.1.2 Function Documentation	19
8.1.2.1 glCheckError()	19
8.1.2.2 glErrorString()	20
8.1.2.3 initGLFW()	20
8.1.2.4 operator&() [1/2]	21
8.1.2.5 operator&() [2/2]	21
8.1.2.6 operator" () [1/2]	21
8.1.2.7 operator" () [2/2]	21
8.1.2.8 operator" =() [1/2]	21
8.1.2.9 operator" =() [2/2]	21
8.1.2.10 pollEvents()	21
8.1.2.11 shaderTypeToString()	22
8.1.2.12 terminateGLFW()	22
8.1.2.13 toGLenum() [1/5]	22
8.1.2.14 toGLenum() [2/5]	22

8.1.2.15 toGLenum() [3/5]	22
8.1.2.16 toGLenum() [4/5]	22
8.1.2.17 toGLenum() [5/5]	23
8.1.2.18 validateBufferFlag()	23
8.1.2.19 validateMapUsage()	23
9 Data Structure Documentation	25
9.1 egl::Buffer Class Reference	25
9.1.1 Constructor & Destructor Documentation	26
9.1.1.1 Buffer() [1/4]	26
9.1.1.2 Buffer() [2/4]	26
9.1.1.3 Buffer() [3/4]	26
9.1.1.4 Buffer() [4/4]	26
9.1.1.5 ~Buffer()	26
9.1.2 Member Function Documentation	26
9.1.2.1 _check()	26
9.1.2.2 _delete()	26
9.1.2.3 bind()	27
9.1.2.4 getSubData()	27
9.1.2.5 getType()	27
9.1.2.6 map()	27
9.1.2.7 operator=() [1/2]	27
9.1.2.8 operator=() [2/2]	27
9.1.2.9 setData() [1/2]	27
9.1.2.10 setData() [2/2]	27
9.1.2.11 setStorage() [1/2]	28
9.1.2.12 setStorage() [2/2]	28
9.1.2.13 setSubData()	28
9.1.2.14 size()	28
9.1.2.15 unmap()	28
9.1.3 Field Documentation	28
9.1.3.1 _flags	28
9.1.3.2 _id	28
9.1.3.3 _mapped	28
9.1.3.4 _mapUsage	29
9.1.3.5 _type	29
9.2 egl::Program Class Reference	29
9.2.1 Detailed Description	30
9.2.2 Constructor & Destructor Documentation	30
9.2.2.1 Program() [1/3]	30
9.2.2.2 Program() [2/3]	30
9.2.2.3 Program() [3/3]	30

9.2.2.4 ~Program()	30
9.2.3 Member Function Documentation	31
9.2.3.1 _check()	31
9.2.3.2 _delete()	31
9.2.3.3 _ensure()	31
9.2.3.4 _getError()	31
9.2.3.5 attach()	31
9.2.3.6 attached()	32
9.2.3.7 bind()	32
9.2.3.8 detach()	32
9.2.3.9 link()	33
9.2.3.10 linked()	33
9.2.3.11 operator=() [1/2]	33
9.2.3.12 operator=() [2/2]	33
9.2.3.13 reset()	34
9.2.3.14 unbind()	34
9.2.4 Field Documentation	34
9.2.4.1 _id	34
9.2.4.2 _linked	34
9.3 egl::ProgramLinkError Class Reference	34
9.3.1 Detailed Description	35
9.3.2 Constructor & Destructor Documentation	35
9.3.2.1 ProgramLinkError()	35
9.4 egl::ProgramValidateError Class Reference	35
9.4.1 Detailed Description	36
9.4.2 Constructor & Destructor Documentation	36
9.4.2.1 ProgramValidateError()	36
9.5 egl::Shader Class Reference	36
9.5.1 Detailed Description	37
9.5.2 Constructor & Destructor Documentation	37
9.5.2.1 Shader() [1/8]	37
9.5.2.2 Shader() [2/8]	38
9.5.2.3 Shader() [3/8]	38
9.5.2.4 Shader() [4/8]	38
9.5.2.5 Shader() [5/8]	39
9.5.2.6 Shader() [6/8]	39
9.5.2.7 Shader() [7/8]	39
9.5.2.8 Shader() [8/8]	39
9.5.2.9 ~Shader()	40
9.5.3 Member Function Documentation	40
9.5.3.1 _check()	40
9.5.3.2 _delete()	40

9.5.3.3 <code>_ensure()</code>	40
9.5.3.4 <code>_getError()</code>	40
9.5.3.5 <code>compile() [1/4]</code>	40
9.5.3.6 <code>compile() [2/4]</code>	41
9.5.3.7 <code>compile() [3/4]</code>	41
9.5.3.8 <code>compile() [4/4]</code>	42
9.5.3.9 <code>compiled()</code>	42
9.5.3.10 <code>getType()</code>	42
9.5.3.11 <code>operator=() [1/2]</code>	42
9.5.3.12 <code>operator=() [2/2]</code>	42
9.5.3.13 <code>reset()</code>	43
9.5.4 Field Documentation	43
9.5.4.1 <code>_compiled</code>	43
9.5.4.2 <code>_id</code>	43
9.5.4.3 <code>_type</code>	43
9.5.4.4 Program	43
9.6 <code>egl::ShaderCompileError</code> Class Reference	43
9.6.1 Detailed Description	44
9.6.2 Constructor & Destructor Documentation	44
9.6.2.1 <code>ShaderCompileError()</code>	44
9.7 <code>egl::VertexBuffer</code> Class Reference	44
9.8 <code>egl::WindowContext</code> Class Reference	44
9.8.1 Detailed Description	45
9.8.2 Constructor & Destructor Documentation	46
9.8.2.1 <code>WindowContext() [1/3]</code>	46
9.8.2.2 <code>WindowContext() [2/3]</code>	46
9.8.2.3 <code>WindowContext() [3/3]</code>	46
9.8.2.4 <code>~WindowContext()</code>	46
9.8.3 Member Function Documentation	46
9.8.3.1 <code>operator=() [1/2]</code>	46
9.8.3.2 <code>operator=() [2/2]</code>	47
9.8.3.3 <code>run()</code>	47
9.8.3.4 <code>shouldClose()</code>	47
9.8.3.5 <code>swapBuffers()</code>	47
9.8.3.6 <code>useContext()</code>	48
9.8.4 Field Documentation	48
9.8.4.1 <code>window</code>	48
10 File Documentation	49
10.1 <code>inc/EGL/buffer.h</code> File Reference	49
10.2 <code>buffer.h</code>	50
10.3 <code>inc/EGL/debug.h</code> File Reference	52

10.3.1 Macro Definition Documentation	52
10.3.1.1 DEBUG_ONLY	52
10.3.1.2 GL_CALL	53
10.4 debug.h	53
10.5 inc/EGL/mainpage.md File Reference	53
10.6 inc/EGL/program.h File Reference	53
10.7 program.h	54
10.8 inc/EGL/shader.h File Reference	55
10.9 shader.h	56
10.10 inc/EGL/vertexBuffer.h File Reference	57
10.11 vertexBuffer.h	57
10.12 inc/EGL/windowContext.h File Reference	57
10.13 windowContext.h	58
Index	59

Chapter 1

Introduction

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

1.1 Recommended Setup

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

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

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

Chapter 2

Directory Hierarchy

2.1 Directories

EGL	13
buffer.h	49
debug.h	52
program.h	53
shader.h	55
vertexBuffer.h	57
windowContext.h	57
inc	13
EGL	13
buffer.h	49
debug.h	52
program.h	53
shader.h	55
vertexBuffer.h	57
windowContext.h	57

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

egl::Buffer	25
egl::Program	29
std::runtime_error	
egl::ProgramLinkError	34
egl::ProgramValidateError	35
egl::ShaderCompileError	43
egl::Shader	36
egl::VertexBuffer	44
egl::WindowContext	44

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

<code>egl::Buffer</code>	25
<code>egl::Program</code>	
Program class to abstract OpenGL Shader Programs	29
<code>egl::ProgramLinkError</code>	
Exception thrown when Program linking fails	34
<code>egl::ProgramValidateError</code>	
Exception thrown when Program validation fails, mainly used in debug builds	35
<code>egl::Shader</code>	
Shader class to abstract OpenGL Shaders	36
<code>egl::ShaderCompileError</code>	
Exception thrown when Shader compilation fails	43
<code>egl::VertexBuffer</code>	44
<code>egl::WindowContext</code>	
An abstract class to contain an window based application	44

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

inc/EGL/ buffer.h	49
inc/EGL/ debug.h	52
inc/EGL/ program.h	53
inc/EGL/ shader.h	55
inc/EGL/ vertexBuffer.h	57
inc/EGL/ windowContext.h	57

Chapter 7

Directory Documentation

7.1 inc/EGL Directory Reference

Files

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

7.2 inc Directory Reference

Directories

- directory [EGL](#)

Chapter 8

Namespace Documentation

8.1 egl Namespace Reference

Data Structures

- class [Buffer](#)
Exception thrown when [Program](#) linking fails.
- class [ProgramLinkError](#)
- class [ProgramValidateError](#)
Exception thrown when [Program](#) validation fails, mainly used in debug builds.
- class [Program](#)
[Program](#) class to abstract OpenGL [Shader](#) Programs.
- class [ShaderCompileError](#)
Exception thrown when [Shader](#) compilation fails.
- class [Shader](#)
[Shader](#) class to abstract OpenGL Shaders.
- class [VertexBuffer](#)
- class [WindowContext](#)
An abstract class to contain an window based application.

Enumerations

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

- *Enum flags to indicate Buffer map usage.*
- enum class `BufferFlag` : `uint32_t` {

`None` = 0 , `DynamicStorage` = 1 << 0 , `MapRead` = 1 << 1 , `MapWrite` = 1 << 2 ,

`MapPersistent` = 1 << 3 , `MapCoherent` = 1 << 4 , `ClientStorage` = 1 << 5 }
- Enum flags for explicit Buffer usage in setStorage.*
- enum class `ShaderType` {

`Fragment` , `Vertex` , `Geometry` , `TessEvaluation` ,

`TessControl` , `Compute` }
- ShaderType enum to indicate usage.*

Functions

- `MapUsage operator| (MapUsage a, MapUsage b)`
- `MapUsage operator& (MapUsage a, MapUsage b)`
- `MapUsage & operator|= (MapUsage &a, MapUsage b)`
- `BufferFlag operator| (BufferFlag a, BufferFlag b)`
- `BufferFlag operator& (BufferFlag a, BufferFlag b)`
- `BufferFlag & operator|= (BufferFlag &a, BufferFlag b)`
- `unsigned int toGLenum (BufferType type)`
- `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)`
- `const char * glErrorString (unsigned int err)`

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

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

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

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

Initialize GLFW.
- `void terminateGLFW ()`

Terminates GLFW.
- `void pollEvents ()`

Polls GLFW events.

8.1.1 Enumeration Type Documentation

8.1.1.1 BufferFlag

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

Enum flags for explicit Buffer usage in setStorage.

Enumerator

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

8.1.1.2 BufferType

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

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

Enumerator

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

8.1.1.3 BufferUsage

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

Enum to indicate [Buffer Usage](#).

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

The frequency of usage may be one of these:

Stream

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

Static

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

Dynamic

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

The nature of usage may be one of these:

Draw

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

Read

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

Copy

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

Enumerator

StreamDraw	
StreamRead	
StreamCopy	
StaticDraw	
StaticRead	
StaticCopy	
DynamicDraw	
DynamicRead	
DynamicCopy	

8.1.1.4 MapUsage

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

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

Enumerator

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

8.1.1.5 ShaderType

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

[ShaderType](#) enum to indicate usage.

Enumerator

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

8.1.2 Function Documentation

8.1.2.1 glCheckError()

```
void egl::glCheckError (
    const char * func,
```

```
const char * file,
int line)
```

Checks and prints if any OpenGL error has occurred.

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

Note

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

Parameters

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

8.1.2.2 glErrorString()

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

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

Note

Returns "UNKNOWN_ERROR" if the error is not known.

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

Parameters

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

Returns

c-style NULL terminated string

8.1.2.3 initGLFW()

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

Initialize GLFW.

Sets up error callbacks and calls glfwInit.

Note

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

Returns

true if GLFW has been successfully initialized.

8.1.2.4 operator&() [1/2]

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

8.1.2.5 operator&() [2/2]

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

8.1.2.6 operator" | () [1/2]

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

8.1.2.7 operator" | () [2/2]

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

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

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

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

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

8.1.2.10 pollEvents()

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

Polls GLFW events.

Note

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

8.1.2.11 `shaderTypeToString()`

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

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

Parameters

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

Returns

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

8.1.2.12 `terminateGLFW()`

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

Terminates GLFW.

Note

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

8.1.2.13 `toGLenum()` [1/5]

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

8.1.2.14 `toGLenum()` [2/5]

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

8.1.2.15 `toGLenum()` [3/5]

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

8.1.2.16 `toGLenum()` [4/5]

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

8.1.2.17 toGLenum() [5/5]

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

Converts the `ShaderType` enum into an OpenGL enum.

Exceptions

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

Parameters

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

Returns

The resulting OpenGL enum.

8.1.2.18 validateBufferFlag()

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

8.1.2.19 validateMapUsage()

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


Chapter 9

Data Structure Documentation

9.1 egl::Buffer Class Reference

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

Public Member Functions

- `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)`
- `void setStorage (int64_t size, const void *data, BufferFlag flags)`
- `template<typename T> void setStorage (std::vector< T > data, BufferFlag flags)`
- `void setSubData (int64_t offset, int64_t size, const void *data)`
- `void * map (int64_t offset, int64_t length, MapUsage access)`
- `void unmap ()`
- `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]

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

9.1.1.2 `Buffer()` [2/4]

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

9.1.1.3 `Buffer()` [3/4]

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

9.1.1.4 `Buffer()` [4/4]

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

9.1.1.5 `~Buffer()`

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

9.1.2 Member Function Documentation

9.1.2.1 `_check()`

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

9.1.2.2 `_delete()`

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

9.1.2.3 bind()

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

9.1.2.4 getSubData()

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

9.1.2.5 getType()

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

9.1.2.6 map()

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

9.1.2.7 operator=() [1/2]

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

9.1.2.8 operator=() [2/2]

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

9.1.2.9 setData() [1/2]

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

9.1.2.10 setData() [2/2]

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

9.1.2.11 `setStorage()` [1/2]

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

9.1.2.12 `setStorage()` [2/2]

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

9.1.2.13 `setSubData()`

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

9.1.2.14 `size()`

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

9.1.2.15 `unmap()`

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

9.1.3 Field Documentation

9.1.3.1 `_flags`

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

9.1.3.2 `_id`

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

9.1.3.3 `_mapped`

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

9.1.3.4 `_mapUsage`

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

9.1.3.5 `_type`

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

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

- inc/EGL/buffer.h

9.2 egl::Program Class Reference

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

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

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 \(\)`](#)
Binds this Program.
- [`Program & operator= \(Program &&other\)`](#)
- [`Program & operator= \(const Program &\)=delete`](#)

Static Public Member Functions

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

Protected Member Functions

- void `_delete()`
- void `_check()`
- void `_ensure()`
- std::string `_getError()`

Protected Attributes

- unsigned int `_id` = 0
- bool `_linked` = false

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]

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

Construct an empty [Program](#) object.

9.2.2.2 Program() [2/3]

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

9.2.2.3 Program() [3/3]

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

9.2.2.4 ~Program()

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

9.2.3 Member Function Documentation

9.2.3.1 `_check()`

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

9.2.3.2 `_delete()`

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

9.2.3.3 `_ensure()`

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

9.2.3.4 `_getError()`

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

9.2.3.5 `attach()`

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

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

Note

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

Warning

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

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

Exceptions

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

Parameters

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

9.2.3.6 attached()

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

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

Parameters

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

Exceptions

<code>std::logic_error</code>	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.

9.2.3.7 bind()

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

Binds this [Program](#).

Exceptions

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

9.2.3.8 detach()

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

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

Note

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

Exceptions

<code>std::runtime_error</code>	If the Shader was not attached.
---------------------------------	---

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

Parameters

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

9.2.3.9 link()

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

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

Exceptions

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

9.2.3.10 linked()

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

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

Returns

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

9.2.3.11 operator=() [1/2]

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

9.2.3.12 operator=() [2/2]

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

9.2.3.13 reset()

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

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

Exceptions

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

9.2.3.14 unbind()

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

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

9.2.4 Field Documentation

9.2.4.1 _id

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

9.2.4.2 _linked

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

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

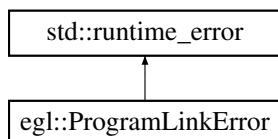
- inc/EGL/[program.h](#)

9.3 egl::ProgramLinkError Class Reference

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

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

Inheritance diagram for egl::ProgramLinkError:



Public Member Functions

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

9.3.1 Detailed Description

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

9.3.2 Constructor & Destructor Documentation

9.3.2.1 ProgramLinkError()

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

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

Parameters

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

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

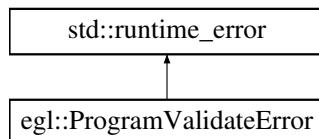
- inc/EGL/[program.h](#)

9.4 egl::ProgramValidateError Class Reference

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

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

Inheritance diagram for egl::ProgramValidateError:



Public Member Functions

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

9.4.1 Detailed Description

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

9.4.2 Constructor & Destructor Documentation

9.4.2.1 ProgramValidateError()

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

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

Parameters

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

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

- inc/EGL/program.h

9.5 egl::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]

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

9.5.2.2 `Shader()` [2/8]

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

Construct a new `Shader` object of given type.

Exceptions

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

9.5.2.3 `Shader()` [3/8]

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

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

Exceptions

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

9.5.2.4 `Shader()` [4/8]

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

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

Exceptions

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

9.5.2.5 Shader() [5/8]

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

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

Exceptions

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

9.5.2.6 Shader() [6/8]

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

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

Exceptions

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

9.5.2.7 Shader() [7/8]

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

9.5.2.8 Shader() [8/8]

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

9.5.2.9 ~Shader()

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

9.5.3 Member Function Documentation

9.5.3.1 _check()

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

9.5.3.2 _delete()

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

9.5.3.3 _ensure()

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

9.5.3.4 _getError()

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

9.5.3.5 compile() [1/4]

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

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

Exceptions

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

Parameters

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

9.5.3.6 compile() [2/4]

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

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

Exceptions

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

Parameters

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

9.5.3.7 compile() [3/4]

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

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

Note

Reads the entire stream and compiles it.

Exceptions

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

Parameters

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

9.5.3.8 compile() [4/4]

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

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

Note

Reads the entire stream and compiles it.

Exceptions

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

Parameters

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

9.5.3.9 compiled()

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

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

9.5.3.10 getType()

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

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

9.5.3.11 operator=() [1/2]

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

9.5.3.12 operator=() [2/2]

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

9.5.3.13 reset()

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

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

Note

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

Exceptions

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

9.5.4 Field Documentation

9.5.4.1 _compiled

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

9.5.4.2 _id

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

9.5.4.3 _type

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

9.5.4.4 Program

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

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

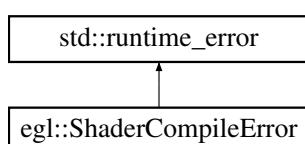
- inc/EGL/shader.h

9.6 egl::ShaderCompileError Class Reference

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

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

Inheritance diagram for egl::ShaderCompileError:



Public Member Functions

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

9.6.1 Detailed Description

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

9.6.2 Constructor & Destructor Documentation

9.6.2.1 [ShaderCompileError\(\)](#)

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

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

Parameters

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

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

- inc/EGL/shader.h

9.7 [egl::VertexBuffer](#) Class Reference

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

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

- inc/EGL/vertexBuffer.h

9.8 [egl::WindowContext](#) Class Reference

An abstract class to contain an window based application.

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

Public Member Functions

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

Protected Member Functions

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

Protected Attributes

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

9.8.1 Detailed Description

An abstract class to contain an window based application.

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

Note

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

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

9.8.2 Constructor & Destructor Documentation

9.8.2.1 WindowContext() [1/3]

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

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

Exceptions

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

Parameters

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

9.8.2.2 WindowContext() [2/3]

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

9.8.2.3 WindowContext() [3/3]

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

9.8.2.4 ~WindowContext()

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

9.8.3 Member Function Documentation

9.8.3.1 operator=() [1/2]

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

9.8.3.2 operator=() [2/2]

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

9.8.3.3 run()

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

Virtual run method to overload when implementing a child class.

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

Note

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

9.8.3.4 shouldClose()

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

Checks if the Window should close.

Exceptions

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

9.8.3.5 swapBuffers()

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

Swaps the display buffers.

Exceptions

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

9.8.3.6 useContext()

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

Makes the owned window the current context.

Exceptions

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

9.8.4 Field Documentation

9.8.4.1 window

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

GLFW window handle for low level GLFW access.

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

- inc/EGL/windowContext.h

Chapter 10

File Documentation

10.1 inc/EGL/buffer.h File Reference

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

Data Structures

- class [egl::Buffer](#)

Namespaces

- namespace [egl](#)

Enumerations

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

Enum to indicate the type of Buffer.
- enum class [egl::BufferUsage](#) {
 egl::StreamDraw , egl::StreamRead , egl::StreamCopy , egl::StaticDraw ,
 egl::StaticRead , egl::StaticCopy , egl::DynamicDraw , egl::DynamicRead ,
 egl::DynamicCopy }

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

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

Enum flags for explicit Buffer usage in setStorage.

Functions

- `MapUsage egl::operator| (MapUsage a, MapUsage b)`
- `MapUsage egl::operator& (MapUsage a, MapUsage b)`
- `MapUsage & egl::operator|= (MapUsage &a, MapUsage b)`
- `BufferFlag egl::operator| (BufferFlag a, BufferFlag b)`
- `BufferFlag egl::operator& (BufferFlag a, BufferFlag b)`
- `BufferFlag & egl::operator|= (BufferFlag &a, BufferFlag b)`
- `unsigned int egl::toGLenum (BufferType type)`
- `unsigned int egl::toGLenum (BufferUsage usage)`
- `unsigned int egl::toGLenum (MapUsage usage)`
- `unsigned int egl::toGLenum (BufferFlag flag)`
- `bool egl::validateMapUsage (MapUsage usage, std::string &error)`
- `bool egl::validateBufferFlag (BufferFlag flag, std::string &error)`

10.2 buffer.h

Go to the documentation of this file.

```
00001 #ifndef EGL_BUFFER_H
00002 #define EGL_BUFFER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <cstdint>
00007 #include <vector>
00008
00009 namespace egl {
00010
00014 enum class BufferType {
00015     Array,
00016     AtomicCounter,
00017     CopyRead,
00018     CopyWrite,
00019     DispatchIndirect,
00020     DrawIndirect,
00021     ElementArray,
00022     PixelPack,
00023     PixelUnpack,
00024     Query,
00025     ShaderStorage,
00026     Texture,
00027     TransformFeedback,
00028     Uniform
00029 };
00030
00059 enum class BufferUsage {
00060     StreamDraw,
00061     StreamRead,
00062     StreamCopy,
00063     StaticDraw,
00064     StaticRead,
00065     StaticCopy,
00066     DynamicDraw,
00067     DynamicRead,
00068     DynamicCopy
00069 };
00070
00074 enum class MapUsage : uint32_t {
00075     None          = 0,
00076     Read          = 1 << 0,
00077     Write         = 1 << 1,
00078     Persistent    = 1 << 2,
00079     Coherent      = 1 << 3,
00080     InvalidRange  = 1 << 4,
00081     InvalidateBuffer = 1 << 5,
00082     FlushExplicit = 1 << 6,
00083     Unynchronized = 1 << 7
00084 };
00085
00086 inline MapUsage operator| (MapUsage a, MapUsage b) {
00087     return static_cast<MapUsage>(
00088         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00089 );
```

```
00090 }
00091
00092 inline MapUsage operator&(MapUsage a, MapUsage b) {
00093     return static_cast<MapUsage>(
00094         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00095     );
00096 }
00097
00098 inline MapUsage& operator|=(MapUsage& a, MapUsage b) {
00099     a = a | b;
00100     return a;
00101 }
00102
00103 enum class BufferFlag : uint32_t {
00104     None          = 0,
00105     DynamicStorage = 1 << 0,
00106     MapRead        = 1 << 1,
00107     MapWrite        = 1 << 2,
00108     MapPersistent   = 1 << 3,
00109     MapCoherent     = 1 << 4,
00110     ClientStorage   = 1 << 5
00111 };
00112
00113 inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
00114     return static_cast<BufferFlag>(
00115         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00116     );
00117 }
00118
00119 inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
00120     return static_cast<BufferFlag>(
00121         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00122     );
00123 }
00124
00125 inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
00126     a = a | b;
00127     return a;
00128 }
00129
00130 unsigned int toGLenum(BufferType type);
00131 unsigned int toGLenum(BufferUsage usage);
00132 unsigned int toGLenum(MapUsage usage);
00133 unsigned int toGLenum(BufferFlag flag);
00134 bool validateMapUsage(MapUsage usage, std::string& error);
00135 bool validateBufferFlag(BufferFlag flag, std::string& error);
00136
00137 class Buffer {
00138 protected:
00139     unsigned int _id = 0;
00140     bool _mapped = false;
00141     MapUsage _mapUsage = MapUsage::None;
00142     BufferFlag _flags = BufferFlag::None;
00143     BufferType _type;
00144
00145     void _delete();
00146     void _check();
00147
00148 public:
00149     Buffer() = delete;
00150     Buffer(BufferType type);
00151     Buffer(Buffer&& other);
00152     Buffer(const Buffer& other) = delete;
00153     ~Buffer() noexcept;
00154
00155     void bind() const;
00156     int64_t size() const;
00157
00158     BufferType getType() const;
00159
00160     void setData(int64_t size, const void* data, BufferUsage usage); // size in bytes
00161     template <typename T>
00162     void setData(std::vector<T> data, BufferUsage usage) { setData(data.size() * sizeof(T),
00163     (void*)data.data(), usage); }
00164
00165     void setStorage(int64_t size, const void* data, BufferFlag flags); // size in bytes
00166     template <typename T>
00167     void setStorage(std::vector<T> data, BufferFlag flags) { setStorage(data.size() * sizeof(T),
00168     (void*)data.data(), flags); }
00169
00170     void setSubData(int64_t offset, int64_t size, const void* data); // offset and size in bytes
00171
00172     void getSubData(int64_t offset, int64_t size, void* data); // data must be at least be as big as
00173     size
00174
00175     void* map(int64_t offset, int64_t length, MapUsage access); // get a pointer to the buffer data
for a specific usage
```

```
00176     void unmap();
00177
00178     Buffer& operator=(Buffer&& other);
00179     Buffer& operator=(const Buffer& other) = delete;
00180 };
00181
00182 }
00183
00184 #endif
```

10.3 inc/EGL/debug.h File Reference

Namespaces

- namespace `egl`

Macros

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

Functions

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

10.3.1 Macro Definition Documentation

10.3.1.1 DEBUG_ONLY

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

Value:

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

Runs given code only if DEBUG_BUILD is defined.

10.3.1.2 GL_CALL

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

Value:

x

Runs the given code and checks for errors.

Note

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

Warning

The code is run in a scope below the current.

Parameters

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

10.4 debug.h

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

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

10.5 inc/EGL/mainpage.md File Reference

10.6 inc/EGL/program.h File Reference

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

Data Structures

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

Namespaces

- namespace `egl`

10.7 program.h

Go to the documentation of this file.

```

00001 #ifndef EGL_PROGRAM_H
00002 #define EGL_PROGRAM_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006
00007 namespace egl {
00008
00009 class Shader;
00010
00014 class ProgramLinkError : public std::runtime_error {
00015 public:
00021     ProgramLinkError(const std::string& infoLog)
00022         : std::runtime_error(
00023             "Program link failed:\n" + infoLog) {}
00024 };
00025
00029 class ProgramValidateError : public std::runtime_error {
00030 public:
00036     ProgramValidateError(const std::string& infoLog)
00037         : std::runtime_error(
00038             "Program validation failed:\n" + infoLog) {}
00039 };
00040
00050 class Program {
00051 protected:
00052     unsigned int _id = 0;
00053     bool _linked = false;
00054
00055     void _delete();
00056     void _check();
00057     void _ensure();
00058     std::string _getError();
00059
00060 public:
00064     Program();
00065     Program(Program&& other);
00066     Program(const Program&) = delete; // OpenGL Programs are not copy safe
00067     ~Program();
00068
00074     void reset();
00075
00085     bool attached(const Shader& shader);
00086
00100    void attach(const Shader& shader);
00101
00112    void detach(const Shader& shader);
00113
00119    bool linked() const { return _linked; }
00120
00128    void link();
00129
00136    void bind();
00137
00141    static void unbind();
00142

```

```
00143     Program& operator=(Program&& other);
00144     Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe
00145 };
00146
00147 }
00148
00149 #endif
```

10.8 inc/EGL/shader.h File Reference

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

Data Structures

- class [egl::ShaderCompileError](#)
Exception thrown when `Shader` compilation fails.
- class [egl::Shader](#)
`Shader` class to abstract OpenGL Shaders.

Namespaces

- namespace [egl](#)

Enumerations

- enum class [egl::ShaderType](#) {
 [egl::Fragment](#) , [egl::Vertex](#) , [egl::Geometry](#) , [egl::TessEvaluation](#) ,
 [egl::TessControl](#) , [egl::Compute](#) }
`ShaderType` enum to indicate usage.

Functions

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

10.9 shader.h

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

```

00001 #ifndef EGL_SHADER_H
00002 #define EGL_SHADER_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <iostream> // std::istream forward-declared
00007
00008 namespace egl {
00009
00010 class Program;
00011
00015 enum class ShaderType {
00016     Fragment,
00017     Vertex,
00018     Geometry,
00019     TessEvaluation,
00020     TessControl,
00021     Compute
00022 };
00023
00030 constexpr std::string shaderTypeToString(ShaderType type);
00031
00040 unsigned int toGLenum(ShaderType type);
00041
00045 class ShaderCompileError : public std::runtime_error {
00046 public:
00053     ShaderCompileError(ShaderType ShaderType,
00054             const std::string& infoLog)
00055         : std::runtime_error(
00056             "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
00057 };
00058
00068 class Shader {
00069 protected:
00070     unsigned int _id = 0;
00071     ShaderType _type;
00072     bool _compiled = false;
00073
00074     void _delete();
00075     void _check();
00076     void _ensure();
00077     std::string _getError();
00078
00079 public:
00080     Shader() = delete;
00081     Shader(ShaderType type);
00096     Shader(ShaderType type, const char* src);
00105     Shader(ShaderType type, const std::string& src);
00117     Shader(ShaderType type, std::istream& in);
00129     Shader(ShaderType type, std::istream&& in);
00130
00131     Shader(Shader&& other);
00132     Shader(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00133     ~Shader() noexcept;
00134
00143     void reset();
00144
00148     bool compiled() const { return _compiled; }
00149
00153     ShaderType getType() const { return _type; }
00154
00164     void compile(const char* src);
00165
00179     void compile(std::istream& in);
00180
00194     void compile(std::istream&& in);
00195
00206     void compile(const std::string& str);
00207
00208     friend Program;
00209
00210     Shader& operator=(Shader&& other);
00211     Shader& operator=(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00212 };
00213
00214 }
00215
00216 #endif

```

10.10 inc/EGL/vertexBuffer.h File Reference

Data Structures

- class [egl::VertexBuffer](#)

Namespaces

- namespace [egl](#)

10.11 vertexBuffer.h

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

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

10.12 inc/EGL/windowContext.h File Reference

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

Data Structures

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

Namespaces

- namespace [egl](#)

Functions

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

10.13 windowContext.h

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

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

Index

- _check
 - egl::Buffer, 26
 - egl::Program, 31
 - egl::Shader, 40
- _compiled
 - egl::Shader, 43
- _delete
 - egl::Buffer, 26
 - egl::Program, 31
 - egl::Shader, 40
- _ensure
 - egl::Program, 31
 - egl::Shader, 40
- _flags
 - egl::Buffer, 28
- _getError
 - egl::Program, 31
 - egl::Shader, 40
- _id
 - egl::Buffer, 28
 - egl::Program, 34
 - egl::Shader, 43
- _linked
 - egl::Program, 34
- _mapUsage
 - egl::Buffer, 28
- _mapped
 - egl::Buffer, 28
- _type
 - egl::Buffer, 29
 - egl::Shader, 43
- ~Buffer
 - egl::Buffer, 26
- ~Program
 - egl::Program, 30
- ~Shader
 - egl::Shader, 39
- ~WindowContext
 - egl::WindowContext, 46
- Array
 - egl, 17
- AtomicCounter
 - egl, 17
- attach
 - egl::Program, 31
- attached
 - egl::Program, 31
- bind
- egl::Buffer, 26
- egl::Program, 32
- Buffer
 - egl::Buffer, 26
- BufferFlag
 - egl, 16
- BufferType
 - egl, 17
- BufferUsage
 - egl, 17
- ClientStorage
 - egl, 17
- Coherent
 - egl, 19
- compile
 - egl::Shader, 40, 41
- compiled
 - egl::Shader, 42
- Compute
 - egl, 19
- CopyRead
 - egl, 17
- CopyWrite
 - egl, 17
- debug.h
 - DEBUG_ONLY, 52
 - GL_CALL, 52
- DEBUG_ONLY
 - debug.h, 52
- detach
 - egl::Program, 32
- DispatchIndirect
 - egl, 17
- DrawIndirect
 - egl, 17
- DynamicCopy
 - egl, 18
- DynamicDraw
 - egl, 18
- DynamicRead
 - egl, 18
- DynamicStorage
 - egl, 17
- egl, 15
 - Array, 17
 - AtomicCounter, 17
 - BufferFlag, 16

BufferType, 17
BufferUsage, 17
ClientStorage, 17
Coherent, 19
Compute, 19
CopyRead, 17
CopyWrite, 17
DispatchIndirect, 17
DrawIndirect, 17
DynamicCopy, 18
DynamicDraw, 18
DynamicRead, 18
DynamicStorage, 17
ElementArray, 17
FlushExplicit, 19
Fragment, 19
Geometry, 19
glCheckError, 19
glErrorString, 20
initGLFW, 20
InvalidateBuffer, 19
InvalidRange, 19
MapCoherent, 17
MapPersistent, 17
MapRead, 17
MapUsage, 18
MapWrite, 17
None, 17, 19
operator&, 20, 21
operator | , 21
operator |=, 21
Persistent, 19
PixelPack, 17
PixelUnpack, 17
pollEvents, 21
Query, 17
Read, 19
ShaderStorage, 17
ShaderType, 19
shaderTypeToString, 21
StaticCopy, 18
StaticDraw, 18
StaticRead, 18
StreamCopy, 18
StreamDraw, 18
StreamRead, 18
terminateGLFW, 22
TessControl, 19
TessEvaluation, 19
Texture, 17
toGLenum, 22
TransformFeedback, 17
Uniform, 17
Unsynchronized, 19
validateBufferFlag, 23
validateMapUsage, 23
Vertex, 19
Write, 19

egl::Buffer, 25
 _check, 26
 _delete, 26
 _flags, 28
 _id, 28
 _mapUsage, 28
 _mapped, 28
 _type, 29
 ~Buffer, 26
bind, 26
Buffer, 26
getSubData, 27
getType, 27
map, 27
operator=, 27
setData, 27
setStorage, 27, 28
setSubData, 28
size, 28
unmap, 28

egl::Program, 29
 _check, 31
 _delete, 31
 _ensure, 31
 _getError, 31
 _id, 34
 _linked, 34
 ~Program, 30
attach, 31
attached, 31
bind, 32
detach, 32
link, 33
linked, 33
operator=, 33
Program, 30
reset, 33
unbind, 34

egl::ProgramLinkError, 34
 ProgramLinkError, 35

egl::ProgramValidateError, 35
 ProgramValidateError, 36

egl::Shader, 36
 _check, 40
 _compiled, 43
 _delete, 40
 _ensure, 40
 _getError, 40
 _id, 43
 _type, 43
 ~Shader, 39
compile, 40, 41
compiled, 42
getType, 42
operator=, 42
Program, 43
reset, 42
Shader, 37–39

egl::ShaderCompileError, 43
 ShaderCompileError, 44
egl::VertexBuffer, 44
egl::WindowContext, 44
 ~WindowContext, 46
 operator=, 46
 run, 47
 shouldClose, 47
 swapBuffers, 47
 useContext, 47
 window, 48
 WindowContext, 46
ElementArray
 egl, 17
FlushExplicit
 egl, 19
Fragment
 egl, 19
Geometry
 egl, 19
getSubData
 egl::Buffer, 27
getType
 egl::Buffer, 27
 egl::Shader, 42
GL_CALL
 debug.h, 52
glCheckError
 egl, 19
glErrorString
 egl, 20

inc Directory Reference, 13
inc/EGL Directory Reference, 13
inc/EGL/buffer.h, 49, 50
inc/EGL/debug.h, 52, 53
inc/EGL/mainpage.md, 53
inc/EGL/program.h, 53, 54
inc/EGL/shader.h, 55, 56
inc/EGL/vertexBuffer.h, 57
inc/EGL/windowContext.h, 57, 58
initGLFW
 egl, 20
Introduction, 1
InvalidateBuffer
 egl, 19
InvalidRange
 egl, 19

link
 egl::Program, 33
linked
 egl::Program, 33

map
 egl::Buffer, 27
MapCoherent

egl, 17
MapPersistent
 egl, 17
MapRead
 egl, 17
MapUsage
 egl, 18
MapWrite
 egl, 17
None
 egl, 17, 19
operator=
 egl::Buffer, 27
 egl::Program, 33
 egl::Shader, 42
 egl::WindowContext, 46
operator&
 egl, 20, 21
operator |
 egl, 21
operator |=
 egl, 21
Persistent
 egl, 19
PixelPack
 egl, 17
PixelUnpack
 egl, 17
pollEvents
 egl, 21
Program
 egl::Program, 30
 egl::Shader, 43
ProgramLinkError
 egl::ProgramLinkError, 35
ProgramValidateError
 egl::ProgramValidateError, 36
Query
 egl, 17
Read
 egl, 19
reset
 egl::Program, 33
 egl::Shader, 42
run
 egl::WindowContext, 47
setData
 egl::Buffer, 27
setStorage
 egl::Buffer, 27, 28
setSubData
 egl::Buffer, 28
Shader
 egl::Shader, 37–39

ShaderCompileError
 egl::ShaderCompileError, 44

ShaderStorage
 egl, 17

ShaderType
 egl, 19

shaderTypeToString
 egl, 21

shouldClose
 egl::WindowContext, 47

size
 egl::Buffer, 28

StaticCopy
 egl, 18

StaticDraw
 egl, 18

StaticRead
 egl, 18

StreamCopy
 egl, 18

StreamDraw
 egl, 18

StreamRead
 egl, 18

swapBuffers
 egl::WindowContext, 47

terminateGLFW
 egl, 22

TessControl
 egl, 19

TessEvaluation
 egl, 19

Texture
 egl, 17

toGLenum
 egl, 22

TransformFeedback
 egl, 17

unbind
 egl::Program, 34

Uniform
 egl, 17

unmap
 egl::Buffer, 28

Unsynchronized
 egl, 19

useContext
 egl::WindowContext, 47

validateBufferFlag
 egl, 23

validateMapUsage
 egl, 23

Vertex
 egl, 19

window