

Easy OpenGL

Generated by Doxygen 1.16.1



<b>1 Directory Hierarchy</b>	<b>1</b>
1.1 Directories	1
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Data Structure Index</b>	<b>7</b>
4.1 Data Structures	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Directory Documentation</b>	<b>11</b>
6.1 inc/EGL Directory Reference	11
6.2 inc Directory Reference	11
<b>7 Namespace Documentation</b>	<b>13</b>
7.1 egl Namespace Reference	13
7.1.1 Enumeration Type Documentation	14
7.1.1.1 BufferFlag	14
7.1.1.2 BufferType	15
7.1.1.3 BufferUsage	15
7.1.1.4 MapUsage	17
7.1.1.5 ShaderType	17
7.1.2 Function Documentation	17
7.1.2.1 glCheckError()	17
7.1.2.2 glErrorString()	18
7.1.2.3 operator&() [1/2]	18
7.1.2.4 operator&() [2/2]	18
7.1.2.5 operator"   () [1/2]	19
7.1.2.6 operator"   () [2/2]	19
7.1.2.7 operator"   =() [1/2]	19
7.1.2.8 operator"   =() [2/2]	19
7.1.2.9 shaderTypeToString()	19
7.1.2.10 toGLenum() [1/5]	19
7.1.2.11 toGLenum() [2/5]	20
7.1.2.12 toGLenum() [3/5]	20
7.1.2.13 toGLenum() [4/5]	20
7.1.2.14 toGLenum() [5/5]	20
7.1.2.15 validateBufferFlag()	20
7.1.2.16 validateMapUsage()	20

<b>8 Data Structure Documentation</b>	<b>21</b>
8.1 egl::Buffer Class Reference	21
8.1.1 Constructor & Destructor Documentation	22
8.1.1.1 Buffer() [1/4]	22
8.1.1.2 Buffer() [2/4]	22
8.1.1.3 Buffer() [3/4]	22
8.1.1.4 Buffer() [4/4]	22
8.1.1.5 ~Buffer()	22
8.1.2 Member Function Documentation	22
8.1.2.1 _check()	22
8.1.2.2 _delete()	22
8.1.2.3 bind()	22
8.1.2.4 getSubData()	23
8.1.2.5 getType()	23
8.1.2.6 map()	23
8.1.2.7 operator=() [1/2]	23
8.1.2.8 operator=() [2/2]	23
8.1.2.9 setData()	23
8.1.2.10 setStorage()	23
8.1.2.11 setSubData()	23
8.1.2.12 size()	24
8.1.2.13 unmap()	24
8.1.3 Field Documentation	24
8.1.3.1 _flags	24
8.1.3.2 _id	24
8.1.3.3 _mapped	24
8.1.3.4 _mapUsage	24
8.1.3.5 _type	24
8.2 egl::Program Class Reference	24
8.2.1 Detailed Description	25
8.2.2 Constructor & Destructor Documentation	26
8.2.2.1 Program() [1/3]	26
8.2.2.2 Program() [2/3]	26
8.2.2.3 Program() [3/3]	26
8.2.2.4 ~Program()	26
8.2.3 Member Function Documentation	26
8.2.3.1 _check()	26
8.2.3.2 _delete()	26
8.2.3.3 _ensure()	26
8.2.3.4 _getError()	26
8.2.3.5 attach()	27
8.2.3.6 attached()	27

8.2.3.7 bind()	28
8.2.3.8 detach()	28
8.2.3.9 link()	28
8.2.3.10 linked()	29
8.2.3.11 operator=() [1/2]	29
8.2.3.12 operator=() [2/2]	29
8.2.3.13 reset()	29
8.2.3.14 unbind()	29
8.2.4 Field Documentation	29
8.2.4.1 _id	29
8.2.4.2 _linked	30
8.3 egl::ProgramLinkError Class Reference	30
8.3.1 Detailed Description	30
8.3.2 Constructor & Destructor Documentation	30
8.3.2.1 ProgramLinkError()	30
8.4 egl::ProgramValidateError Class Reference	31
8.4.1 Detailed Description	31
8.4.2 Constructor & Destructor Documentation	31
8.4.2.1 ProgramValidateError()	31
8.5 egl::Shader Class Reference	31
8.5.1 Detailed Description	33
8.5.2 Constructor & Destructor Documentation	33
8.5.2.1 Shader() [1/7]	33
8.5.2.2 Shader() [2/7]	33
8.5.2.3 Shader() [3/7]	33
8.5.2.4 Shader() [4/7]	34
8.5.2.5 Shader() [5/7]	34
8.5.2.6 Shader() [6/7]	34
8.5.2.7 Shader() [7/7]	34
8.5.2.8 ~Shader()	34
8.5.3 Member Function Documentation	35
8.5.3.1 _check()	35
8.5.3.2 _delete()	35
8.5.3.3 _ensure()	35
8.5.3.4 _getError()	35
8.5.3.5 compile() [1/3]	35
8.5.3.6 compile() [2/3]	36
8.5.3.7 compile() [3/3]	36
8.5.3.8 compiled()	36
8.5.3.9 getType()	37
8.5.3.10 operator=() [1/2]	37
8.5.3.11 operator=() [2/2]	37

---

8.5.3.12 reset()	37
8.5.4 Field Documentation	37
8.5.4.1 _compiled	37
8.5.4.2 _id	37
8.5.4.3 _type	37
8.5.4.4 Program	38
8.6 egl::ShaderCompileError Class Reference	38
8.6.1 Detailed Description	38
8.6.2 Constructor & Destructor Documentation	38
8.6.2.1 ShaderCompileError()	38
8.7 egl::VertexBuffer Class Reference	39
<b>9 File Documentation</b>	<b>41</b>
9.1 inc/EGL/buffer.h File Reference	41
9.2 buffer.h	42
9.3 inc/EGL/debug.h File Reference	44
9.3.1 Macro Definition Documentation	44
9.3.1.1 DEBUG_ONLY	44
9.3.1.2 GL_CALL	44
9.4 debug.h	45
9.5 inc/EGL/program.h File Reference	45
9.6 program.h	46
9.7 inc/EGL/shader.h File Reference	46
9.8 shader.h	47
9.9 inc/EGL/vertexBuffer.h File Reference	48
9.10 vertexBuffer.h	48
<b>Index</b>	<b>49</b>

# Chapter 1

## Directory Hierarchy

### 1.1 Directories

EGL . . . . .	11
buffer.h . . . . .	41
debug.h . . . . .	44
program.h . . . . .	45
shader.h . . . . .	46
vertexBuffer.h . . . . .	48
inc . . . . .	11
EGL . . . . .	11
buffer.h . . . . .	41
debug.h . . . . .	44
program.h . . . . .	45
shader.h . . . . .	46
vertexBuffer.h . . . . .	48





## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">egl</a> . . . . .	13
-------------------------------	----



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

egl::Buffer . . . . .	21
egl::Program . . . . .	24
std::runtime_error	
egl::ProgramLinkError . . . . .	30
egl::ProgramValidateError . . . . .	31
egl::ShaderCompileError . . . . .	38
egl::Shader . . . . .	31
egl::VertexBuffer . . . . .	39



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">egl::Buffer</a> . . . . .	21
<a href="#">egl::Program</a> <a href="#">Program</a> class to abstract OpenGL <a href="#">Shader</a> Programs . . . . .	24
<a href="#">egl::ProgramLinkError</a> Exception thrown when <a href="#">Program</a> linking fails . . . . .	30
<a href="#">egl::ProgramValidateError</a> Exception thrown when <a href="#">Program</a> validation fails, mainly used in debug builds . . . . .	31
<a href="#">egl::Shader</a> <a href="#">Shader</a> class to abstract OpenGL Shaders . . . . .	31
<a href="#">egl::ShaderCompileError</a> Exception thrown when <a href="#">Shader</a> compilation fails . . . . .	38
<a href="#">egl::VertexBuffer</a> . . . . .	39



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

inc/EGL/ <a href="#">buffer.h</a> . . . . .	41
inc/EGL/ <a href="#">debug.h</a> . . . . .	44
inc/EGL/ <a href="#">program.h</a> . . . . .	45
inc/EGL/ <a href="#">shader.h</a> . . . . .	46
inc/EGL/ <a href="#">vertexBuffer.h</a> . . . . .	48





## Chapter 6

# Directory Documentation

### 6.1 inc/EGL Directory Reference

#### Files

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

### 6.2 inc Directory Reference

#### Directories

- directory [EGL](#)



# Chapter 7

## Namespace Documentation

### 7.1 egl Namespace Reference

#### Data Structures

- class [Buffer](#)
- class [ProgramLinkError](#)  
*Exception thrown when [Program](#) linking fails.*
- class [ProgramValidateError](#)  
*Exception thrown when [Program](#) validation fails, mainly used in debug builds.*
- 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](#)

#### 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.*

## 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 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.*

## 7.1.1 Enumeration Type Documentation

### 7.1.1.1 BufferFlag

```
enum class egl::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 <a href="#">glBufferSubData</a> .
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.

### 7.1.1.2 BufferType

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

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

#### Enumerator

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

### 7.1.1.3 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	

#### 7.1.1.4 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 fassion and that the client intends to hold and use the returned pointer during subsequent GL operation.
Coherent	Indicates that a persistent mapping is also to be coherent. Coherent maps guarantee that the effect of writes to a buffer's data store by either the client or server will eventually become visible to the other without further intervention from the application.
InvalidRange	Indicates that the previous contents of the specified range may be discarded. Data within this range are undefined with the exception of subsequently written data.
InvalidateBuffer	Indicates that the previous contents of the entire buffer may be discarded. Data within the entire buffer are undefined with the exception of subsequently written data.
FlushExplicit	Indicates that one or more discrete subranges of the mapping may be modified. When this flag is set, modifications to each subrange must be explicitly flushed by calling <code>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> .

#### 7.1.1.5 ShaderType

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

[ShaderType](#) enum to indicate usage.

##### Enumerator

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

## 7.1.2 Function Documentation

### 7.1.2.1 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

#### 7.1.2.2 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

#### 7.1.2.3 operator&() [1/2]

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

#### 7.1.2.4 operator&() [2/2]

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



**7.1.2.5 operator" | () [1/2]**

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

**7.1.2.6 operator" | () [2/2]**

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

**7.1.2.7 operator" |= () [1/2]**

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

**7.1.2.8 operator" |= () [2/2]**

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

**7.1.2.9 shaderTypeToString()**

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

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

**Parameters**

<i>type</i>	The <a href="#">ShaderType</a> to convert to a std::string.
-------------	---

**Returns**

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

**7.1.2.10 toGLenum() [1/5]**

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

**7.1.2.11 toGLenum() [2/5]**

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

**7.1.2.12 toGLenum() [3/5]**

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

**7.1.2.13 toGLenum() [4/5]**

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

**7.1.2.14 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 <a href="#">ShaderType</a> is invalid.
-------------------------------	---

**Parameters**

<code>type</code>	The <a href="#">ShaderType</a> to convert to a <code>std::string</code> .
-------------------	---

**Returns**

The resulting OpenGL enum.

**7.1.2.15 validateBufferFlag()**

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

**7.1.2.16 validateMapUsage()**

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

## Chapter 8

# Data Structure Documentation

### 8.1 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)
- void [setStorage](#) ([int64\\_t size](#), const void \*data, [BufferFlag](#) 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

#### 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](#)

## 8.1.1 Constructor & Destructor Documentation

### 8.1.1.1 Buffer() [1/4]

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

### 8.1.1.2 Buffer() [2/4]

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

### 8.1.1.3 Buffer() [3/4]

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

### 8.1.1.4 Buffer() [4/4]

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

### 8.1.1.5 ~Buffer()

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

## 8.1.2 Member Function Documentation

### 8.1.2.1 \_check()

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

### 8.1.2.2 \_delete()

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

### 8.1.2.3 bind()

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

#### 8.1.2.4 getSubData()

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

#### 8.1.2.5 getType()

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

#### 8.1.2.6 map()

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

#### 8.1.2.7 operator=() [1/2]

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

#### 8.1.2.8 operator=() [2/2]

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

#### 8.1.2.9 setData()

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

#### 8.1.2.10 setStorage()

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

#### 8.1.2.11 setSubData()

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

#### 8.1.2.12 size()

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

#### 8.1.2.13 unmap()

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

### 8.1.3 Field Documentation

#### 8.1.3.1 \_flags

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

#### 8.1.3.2 \_id

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

#### 8.1.3.3 \_mapped

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

#### 8.1.3.4 \_mapUsage

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

#### 8.1.3.5 \_type

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

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

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

## 8.2 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

## 8.2.1 Detailed Description

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

#### Warning

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\(\)](#).

## 8.2.2 Constructor & Destructor Documentation

### 8.2.2.1 Program() [1/3]

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

Construct an empty [Program](#) object.

### 8.2.2.2 Program() [2/3]

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

### 8.2.2.3 Program() [3/3]

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

### 8.2.2.4 ~Program()

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

## 8.2.3 Member Function Documentation

### 8.2.3.1 \_check()

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

### 8.2.3.2 \_delete()

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

### 8.2.3.3 \_ensure()

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

### 8.2.3.4 \_getError()

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



### 8.2.3.5 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

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

#### Parameters

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

### 8.2.3.6 attached()

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

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

#### Parameters

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

#### Exceptions

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

#### Returns

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

### 8.2.3.7 bind()

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

Binds this [Program](#).

#### Exceptions

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

### 8.2.3.8 detach()

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

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

#### Note

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

#### Exceptions

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

#### Parameters

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

### 8.2.3.9 link()

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

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

#### Exceptions

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

**8.2.3.10 linked()**

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

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

**Returns**

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

**8.2.3.11 operator=() [1/2]**

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

**8.2.3.12 operator=() [2/2]**

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

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

**8.2.3.14 unbind()**

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

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

**8.2.4 Field Documentation****8.2.4.1 \_id**

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

### 8.2.4.2 `_linked`

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

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

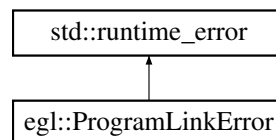
- [inc/EGL/program.h](#)

## 8.3 `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.*

### 8.3.1 Detailed Description

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

### 8.3.2 Constructor & Destructor Documentation

#### 8.3.2.1 ProgramLinkError()

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

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

#### Parameters

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

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

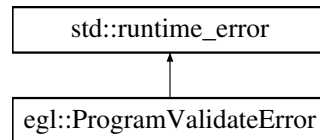
- [inc/EGL/program.h](#)

## 8.4 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.

### 8.4.1 Detailed Description

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

### 8.4.2 Constructor & Destructor Documentation

#### 8.4.2.1 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](#)

## 8.5 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](#) ([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](#) (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

### 8.5.1 Detailed Description

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

#### Warning

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\(\)](#).

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 Shader() [1/7]

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

#### 8.5.2.2 Shader() [2/7]

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

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

#### Exceptions

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

#### 8.5.2.3 Shader() [3/7]

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

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

#### Exceptions

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

**8.5.2.4 Shader()** [4/7]

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

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

**Exceptions**

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

**8.5.2.5 Shader()** [5/7]

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

**8.5.2.6 Shader()** [6/7]

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

**8.5.2.7 Shader()** [7/7]

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

**8.5.2.8 ~Shader()**

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



## 8.5.3 Member Function Documentation

### 8.5.3.1 \_check()

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

### 8.5.3.2 \_delete()

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

### 8.5.3.3 \_ensure()

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

### 8.5.3.4 \_getError()

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

### 8.5.3.5 compile() [1/3]

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

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

#### Exceptions

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

#### Parameters

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

### 8.5.3.6 compile() [2/3]

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

#### Parameters

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

### 8.5.3.7 compile() [3/3]

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

#### Parameters

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

### 8.5.3.8 compiled()

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

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

### 8.5.3.9 getType()

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

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

### 8.5.3.10 operator=() [1/2]

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

### 8.5.3.11 operator=() [2/2]

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

### 8.5.3.12 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 <a href="#">Shader</a> object.
---------------------------------	---

## 8.5.4 Field Documentation

### 8.5.4.1 \_compiled

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

### 8.5.4.2 \_id

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

### 8.5.4.3 \_type

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

#### 8.5.4.4 Program

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

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

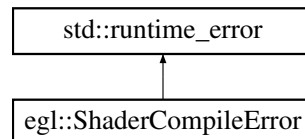
- inc/EGL/[shader.h](#)

## 8.6 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.*

### 8.6.1 Detailed Description

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

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 ShaderCompileError()

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

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

#### Parameters

<a href="#">ShaderType</a>	The type of <a href="#">Shader</a> that failed to compile.
<i>infoLog</i>	InfoLog buffer from glGetShaderInfoLog.

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

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

## 8.7 egl::VertexBuffer Class Reference

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

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

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



## Chapter 9

# File Documentation

### 9.1 inc/EGL/buffer.h File Reference

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

#### 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 falgs 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)

## 9.2 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
00008 namespace egl {
00009
00013 enum class BufferType {
00014     Array,
00015     AtomicCounter,
00016     CopyRead,
00017     CopyWrite,
00018     DispatchIndirect,
00019     DrawIndirect,
00020     ElementArray,
00021     PixelPack,
00022     PixelUnpack,
00023     Query,
00024     ShaderStorage,
00025     Texture,
00026     TransformFeedback,
00027     Uniform
00028 };
00029
00058 enum class BufferUsage {
00059     StreamDraw,
00060     StreamRead,
00061     StreamCopy,
00062     StaticDraw,
00063     StaticRead,
00064     StaticCopy,
00065     DynamicDraw,
00066     DynamicRead,
00067     DynamicCopy
00068 };
00069
00073 enum class MapUsage : uint32_t {
00074     None = 0,
00075     Read = 1 << 0,
00076     Write = 1 << 1,
00077     Persistent = 1 << 2,
00078     Coherent = 1 << 3,
00079     InvalidRange = 1 << 4,
00080     InvalidateBuffer = 1 << 5,
00081     FlushExplicit = 1 << 6,
00082     Unsynchronized = 1 << 7
00083 };
00084
00085 inline MapUsage operator| (MapUsage a, MapUsage b) {
00086     return static_cast<MapUsage>(
00087         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00088     );
00089 }

```



```

00090
00091 inline MapUsage operator&(MapUsage a, MapUsage b) {
00092     return static_cast<MapUsage>(
00093         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00094     );
00095 }
00096
00097 inline MapUsage& operator|=(MapUsage& a, MapUsage b) {
00098     a = a | b;
00099     return a;
00100 }
00101
00105 enum class BufferFlag : uint32_t {
00106     None = 0,
00107     DynamicStorage = 1 << 0,
00108     MapRead = 1 << 1,
00109     MapWrite = 1 << 2,
00110     MapPersistent = 1 << 3,
00111     MapCoherent = 1 << 4,
00112     ClientStorage = 1 << 5
00113 };
00114
00115 inline BufferFlag operator|(BufferFlag a, BufferFlag b) {
00116     return static_cast<BufferFlag>(
00117         static_cast<uint32_t>(a) | static_cast<uint32_t>(b)
00118     );
00119 }
00120
00121 inline BufferFlag operator&(BufferFlag a, BufferFlag b) {
00122     return static_cast<BufferFlag>(
00123         static_cast<uint32_t>(a) & static_cast<uint32_t>(b)
00124     );
00125 }
00126
00127 inline BufferFlag& operator|=(BufferFlag& a, BufferFlag b) {
00128     a = a | b;
00129     return a;
00130 }
00131
00132 unsigned int toGLenum(BufferType type);
00133 unsigned int toGLenum(BufferUsage usage);
00134 unsigned int toGLenum(MapUsage usage);
00135 unsigned int toGLenum(BufferFlag flag);
00136 bool validateMapUsage(MapUsage usage, std::string& error);
00137 bool validateBufferFlag(BufferFlag flag, std::string& error);
00138
00139 class Buffer {
00140 protected:
00141     unsigned int _id = 0;
00142     bool _mapped = false;
00143     MapUsage _mapUsage = MapUsage::None;
00144     BufferFlag _flags = BufferFlag::None;
00145     BufferType _type;
00146
00147     void _delete();
00148     void _check();
00149
00150 public:
00151     Buffer() = delete;
00152     Buffer(BufferType type);
00153     Buffer(Buffer&& other);
00154     Buffer(const Buffer& other) = delete;
00155     ~Buffer() noexcept;
00156
00157     void bind() const;
00158     int64_t size() const;
00159
00160     BufferType getType() const;
00161
00162     void setData(int64_t size, const void* data, BufferUsage usage); // size in bytes
00163     void setStorage(int64_t size, const void* data, BufferFlag flags); // size in bytes
00164
00165     void setSubData(int64_t offset, int64_t size, const void* data); // offset and size in bytes
00166
00167     void getSubData(int64_t offset, int64_t size, void* data); // data must be at least be as big as
size
00168
00169     void* map(int64_t offset, int64_t length, MapUsage access); // get a pointer to the buffer data
for a specific usage
00170     void unmap();
00171
00172     Buffer& operator=(Buffer&& other);
00173     Buffer& operator=(const Buffer& other) = delete;
00174 };
00175
00176 }
00177

```

```
00178 #endif
```

## 9.3 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 occured.*

## 9.3.1 Macro Definition Documentation

### 9.3.1.1 `DEBUG_ONLY`

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

#### Value:

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

Runs given code only if `DEBUG_BUILD` is defined.

### 9.3.1.2 `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
---	--

## 9.4 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 #ifndef DEBUG_BUILD
00046     #define GL_CALL(x) do { x; egl::glCheckError(#x, __FILE__, __LINE__); } while(0)
00048     #define DEBUG_ONLY(x) x
00050 #else
00051     #define GL_CALL(x) x
00053     #define DEBUG_ONLY(x) ((void)0)
00055 #endif
00057 #endif
00058
00059 #endif

```

## 9.5 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`

## 9.6 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
00049     class Program {
00050     protected:
00051         unsigned int _id = 0;
00052         bool _linked = false;
00053
00054         void _delete();
00055         void _check();
00056         void _ensure();
00057         std::string _getError();
00058     public:
00063         Program();
00064         Program(Program&& other);
00065         Program(const Program&) = delete; // OpenGL Programs are not copy safe
00066         ~Program();
00067
00073         void reset();
00074
00084         bool attached(const Shader& shader);
00085
00099         void attach(const Shader& shader);
00100
00111         void detach(const Shader& shader);
00112
00118         bool linked() const { return _linked; }
00119
00127         void link();
00128
00135         void bind();
00136
00140         static void unbind();
00141
00142         Program& operator=(Program&& other);
00143         Program& operator=(const Program&) = delete; // OpenGL Programs are not copy safe
00144     };
00145
00146 }
00147
00148 #endif

```

## 9.7 inc/EGL/shader.h File Reference

```

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

```

## 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.*

## 9.8 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 <iosfwd> // std::istream forward-declared
00007
00008 namespace egl {
00009
00010 class Program;
00011
00012 enum class ShaderType {
00013     Fragment,
00014     Vertex,
00015     Geometry,
00016     TessEvaluation,
00017     TessControl,
00018     Compute
00019 };
00020
00021 constexpr std::string shaderTypeToString(ShaderType type);
00022
00023 unsigned int toGLenum(ShaderType type);
00024
00025 class ShaderCompileError : public std::runtime_error {
00026 public:
00027     ShaderCompileError(ShaderType ShaderType,
00028                       const std::string& infoLog)
00029         : std::runtime_error(
00030             "Shader compile failed (" + shaderTypeToString(ShaderType) + "):\n" + infoLog) {}
00031 };
00032
00033 class Shader {
00034 protected:
```

```

00068     unsigned int _id = 0;
00069     ShaderType _type;
00070     bool _compiled = false;
00071
00072     void _delete();
00073     void _check();
00074     void _ensure();
00075     std::string _getError();
00076
00077 public:
00078     Shader() = delete;
00085     Shader(ShaderType type);
00094     Shader(ShaderType type, const char* src);
00103     Shader(ShaderType type, const std::string& src);
00113     Shader(ShaderType type, std::istream& in);
00114
00115     Shader(Shader&& other);
00116     Shader(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00117     ~Shader() noexcept;
00118
00127     void reset();
00128
00132     bool compiled() const { return _compiled; }
00133
00137     ShaderType getType() const { return _type; }
00138
00148     void compile(const char* src);
00149
00161     void compile(std::istream& in);
00162
00173     void compile(const std::string& str);
00174
00175     friend Program;
00176
00177     Shader& operator=(Shader&& other);
00178     Shader& operator=(const Shader& other) = delete; // OpenGL Shaders are not copy safe
00179 };
00180
00181 }
00182
00183 #endif

```

## 9.9 inc/EGL/vertexBuffer.h File Reference

### Data Structures

- class [egl::VertexBuffer](#)

### Namespaces

- namespace [egl](#)

## 9.10 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

```

# Index

- [egl::Buffer](#), [22](#)
  - [egl::Program](#), [26](#)
  - [egl::Shader](#), [35](#)
- [\\_compiled](#)
  - [egl::Shader](#), [37](#)
- [\\_delete](#)
  - [egl::Buffer](#), [22](#)
  - [egl::Program](#), [26](#)
  - [egl::Shader](#), [35](#)
- [\\_ensure](#)
  - [egl::Program](#), [26](#)
  - [egl::Shader](#), [35](#)
- [\\_flags](#)
  - [egl::Buffer](#), [24](#)
- [\\_getError](#)
  - [egl::Program](#), [26](#)
  - [egl::Shader](#), [35](#)
- [\\_id](#)
  - [egl::Buffer](#), [24](#)
  - [egl::Program](#), [29](#)
  - [egl::Shader](#), [37](#)
- [\\_linked](#)
  - [egl::Program](#), [29](#)
- [\\_mapUsage](#)
  - [egl::Buffer](#), [24](#)
- [\\_mapped](#)
  - [egl::Buffer](#), [24](#)
- [\\_type](#)
  - [egl::Buffer](#), [24](#)
  - [egl::Shader](#), [37](#)
- [~Buffer](#)
  - [egl::Buffer](#), [22](#)
- [~Program](#)
  - [egl::Program](#), [26](#)
- [~Shader](#)
  - [egl::Shader](#), [34](#)
- [Array](#)
  - [egl](#), [15](#)
- [AtomicCounter](#)
  - [egl](#), [15](#)
- [attach](#)
  - [egl::Program](#), [26](#)
- [attached](#)
  - [egl::Program](#), [27](#)
- [bind](#)
  - [egl::Buffer](#), [22](#)
  - [egl::Program](#), [27](#)
- [Buffer](#)
  - [egl::Buffer](#), [22](#)
- [BufferFlag](#)
  - [egl](#), [14](#)
- [BufferType](#)
  - [egl](#), [15](#)
- [BufferUsage](#)
  - [egl](#), [15](#)
- [ClientStorage](#)
  - [egl](#), [15](#)
- [Coherent](#)
  - [egl](#), [17](#)
- [compile](#)
  - [egl::Shader](#), [35](#), [36](#)
- [compiled](#)
  - [egl::Shader](#), [36](#)
- [Compute](#)
  - [egl](#), [17](#)
- [CopyRead](#)
  - [egl](#), [15](#)
- [CopyWrite](#)
  - [egl](#), [15](#)
- [debug.h](#)
  - [DEBUG\\_ONLY](#), [44](#)
  - [GL\\_CALL](#), [44](#)
- [DEBUG\\_ONLY](#)
  - [debug.h](#), [44](#)
- [detach](#)
  - [egl::Program](#), [28](#)
- [DispatchIndirect](#)
  - [egl](#), [15](#)
- [DrawIndirect](#)
  - [egl](#), [15](#)
- [DynamicCopy](#)
  - [egl](#), [16](#)
- [DynamicDraw](#)
  - [egl](#), [16](#)
- [DynamicRead](#)
  - [egl](#), [16](#)
- [DynamicStorage](#)
  - [egl](#), [14](#)
- [egl](#), [13](#)
  - [Array](#), [15](#)
  - [AtomicCounter](#), [15](#)
  - [BufferFlag](#), [14](#)
  - [BufferType](#), [15](#)
  - [BufferUsage](#), [15](#)

- ClientStorage, 15
- Coherent, 17
- Compute, 17
- CopyRead, 15
- CopyWrite, 15
- DispatchIndirect, 15
- DrawIndirect, 15
- DynamicCopy, 16
- DynamicDraw, 16
- DynamicRead, 16
- DynamicStorage, 14
- ElementArray, 15
- FlushExplicit, 17
- Fragment, 17
- Geometry, 17
- glCheckError, 17
- glErrorString, 18
- InvalidateBuffer, 17
- InvalidRange, 17
- MapCoherent, 15
- MapPersistent, 15
- MapRead, 14
- MapUsage, 16
- MapWrite, 14
- None, 14, 17
- operator&, 18
- operator|, 18, 19
- operator|=, 19
- Persistent, 17
- PixelPack, 15
- PixelUnpack, 15
- Query, 15
- Read, 17
- ShaderStorage, 15
- ShaderType, 17
- shaderTypeToString, 19
- StaticCopy, 16
- StaticDraw, 16
- StaticRead, 16
- StreamCopy, 16
- StreamDraw, 16
- StreamRead, 16
- TessControl, 17
- TessEvaluation, 17
- Texture, 15
- toGLenum, 19, 20
- TransformFeedback, 15
- Uniform, 15
- Unsynchronized, 17
- validateBufferFlag, 20
- validateMapUsage, 20
- Vertex, 17
- Write, 17
- egl::Buffer, 21
  - \_check, 22
  - \_delete, 22
  - \_flags, 24
  - \_id, 24
  - \_mapUsage, 24
  - \_mapped, 24
  - \_type, 24
  - ~Buffer, 22
  - bind, 22
  - Buffer, 22
  - getSubData, 22
  - getType, 23
  - map, 23
  - operator=, 23
  - setData, 23
  - setStorage, 23
  - setSubData, 23
  - size, 23
  - unmap, 24
- egl::Program, 24
  - \_check, 26
  - \_delete, 26
  - \_ensure, 26
  - \_getError, 26
  - \_id, 29
  - \_linked, 29
  - ~Program, 26
  - attach, 26
  - attached, 27
  - bind, 27
  - detach, 28
  - link, 28
  - linked, 28
  - operator=, 29
  - Program, 26
  - reset, 29
  - unbind, 29
- egl::ProgramLinkError, 30
  - ProgramLinkError, 30
- egl::ProgramValidateError, 31
  - ProgramValidateError, 31
- egl::Shader, 31
  - \_check, 35
  - \_compiled, 37
  - \_delete, 35
  - \_ensure, 35
  - \_getError, 35
  - \_id, 37
  - \_type, 37
  - ~Shader, 34
  - compile, 35, 36
  - compiled, 36
  - getType, 36
  - operator=, 37
  - Program, 37
  - reset, 37
  - Shader, 33, 34
- egl::ShaderCompileError, 38
  - ShaderCompileError, 38
- egl::VertexBuffer, 39
- ElementArray
  - egl, 15



FlushExplicit  
  egl, 17

Fragment  
  egl, 17

Geometry  
  egl, 17

getSubData  
  egl::Buffer, 22

getType  
  egl::Buffer, 23  
  egl::Shader, 36

GL\_CALL  
  debug.h, 44

glCheckError  
  egl, 17

glErrorString  
  egl, 18

inc Directory Reference, 11

inc/EGL Directory Reference, 11

inc/EGL/buffer.h, 41, 42

inc/EGL/debug.h, 44, 45

inc/EGL/program.h, 45, 46

inc/EGL/shader.h, 46, 47

inc/EGL/vertexBuffer.h, 48

InvalidateBuffer  
  egl, 17

InvalidRange  
  egl, 17

link  
  egl::Program, 28

linked  
  egl::Program, 28

map  
  egl::Buffer, 23

MapCoherent  
  egl, 15

MapPersistent  
  egl, 15

MapRead  
  egl, 14

MapUsage  
  egl, 16

MapWrite  
  egl, 14

None  
  egl, 14, 17

operator=  
  egl::Buffer, 23  
  egl::Program, 29  
  egl::Shader, 37

operator&  
  egl, 18

operator |  
  egl, 18, 19

operator |=  
  egl, 19

Persistent  
  egl, 17

PixelPack  
  egl, 15

PixelUnpack  
  egl, 15

Program  
  egl::Program, 26  
  egl::Shader, 37

ProgramLinkError  
  egl::ProgramLinkError, 30

ProgramValidateError  
  egl::ProgramValidateError, 31

Query  
  egl, 15

Read  
  egl, 17

reset  
  egl::Program, 29  
  egl::Shader, 37

setData  
  egl::Buffer, 23

setStorage  
  egl::Buffer, 23

setSubData  
  egl::Buffer, 23

Shader  
  egl::Shader, 33, 34

ShaderCompileError  
  egl::ShaderCompileError, 38

ShaderStorage  
  egl, 15

ShaderType  
  egl, 17

shaderTypeToString  
  egl, 19

size  
  egl::Buffer, 23

StaticCopy  
  egl, 16

StaticDraw  
  egl, 16

StaticRead  
  egl, 16

StreamCopy  
  egl, 16

StreamDraw  
  egl, 16

StreamRead  
  egl, 16

TessControl  
  egl, 17

TessEvaluation  
    egl, [17](#)  
Texture  
    egl, [15](#)  
toGLenum  
    egl, [19](#), [20](#)  
TransformFeedback  
    egl, [15](#)  
  
unbind  
    egl::Program, [29](#)  
Uniform  
    egl, [15](#)  
unmap  
    egl::Buffer, [24](#)  
Unsynchronized  
    egl, [17](#)  
  
validateBufferFlag  
    egl, [20](#)  
validateMapUsage  
    egl, [20](#)  
Vertex  
    egl, [17](#)  
  
Write  
    egl, [17](#)