Name

glBufferData, glNamedBufferData — creates and initializes a buffer object's data store

C Specification

```
void glBufferData( GLenum target,
                   GLsizeiptr size,
                   const void * data,
                   GLenum usage );

void glNamedBufferData( GLuint buffer,
                        GLsizeiptr size,
                        const void *data,
                        GLenum usage );
```

Parameters

`target`

Specifies the target to which the buffer object is bound for **glBufferData**, which must be one of the buffer binding targets in the following table:

Buffer Binding Target	Purpose
`GL_ARRAY_BUFFER`	Vertex attributes
`GL_ATOMIC_COUNTER_BUFFER`	Atomic counter storage
`GL_COPY_READ_BUFFER`	Buffer copy source
`GL_COPY_WRITE_BUFFER`	Buffer copy destination
`GL_DISPATCH_INDIRECT_BUFFER`	Indirect compute dispatch commands
`GL_DRAW_INDIRECT_BUFFER`	Indirect command arguments
`GL_ELEMENT_ARRAY_BUFFER`	Vertex array indices
`GL_PIXEL_PACK_BUFFER`	Pixel read target
`GL_PIXEL_UNPACK_BUFFER`	Texture data source
`GL_QUERY_BUFFER`	Query result buffer
`GL_SHADER_STORAGE_BUFFER`	Read-write storage for shaders
`GL_TEXTURE_BUFFER`	Texture data buffer
`GL_TRANSFORM_FEEDBACK_BUFFER`	Transform feedback buffer
`GL_UNIFORM_BUFFER`	Uniform block storage

`buffer`

Specifies the name of the buffer object for **glNamedBufferData** function.

`size`

Specifies the size in bytes of the buffer object's new data store.

`data`

Specifies a pointer to data that will be copied into the data store for initialization, or `NULL` if no data is to be copied.

`usage`

Specifies the expected usage pattern of the data store. The symbolic constant must be `GL_STREAM_DRAW`, `GL_STREAM_READ`, `GL_STREAM_COPY`, `GL_STATIC_DRAW`, `GL_STATIC_READ`, `GL_STATIC_COPY`, `GL_DYNAMIC_DRAW`, `GL_DYNAMIC_READ`, or `GL_DYNAMIC_COPY`.

Description

glBufferData and **glNamedBufferData** create a new data store for a buffer object. In case of **glBufferData**, the buffer object currently bound to *`target`* is used. For **glNamedBufferData**, a buffer object associated with ID specified by the caller in *`buffer`* will be used instead.

While creating the new storage, any pre-existing data store is deleted. The new data store is created with the specified *`size`* in bytes and *`usage`*. If *`data`* is not `NULL`, the data store is initialized with data from this pointer. In its initial state, the new data store is not mapped, it has a `NULL` mapped pointer, and its mapped access is `GL_READ_WRITE`.

`usage` is a hint to the GL implementation as to how a buffer object's data store will be accessed. This enables the GL implementation to make more intelligent decisions that may significantly impact buffer object performance. It does not, however, constrain the actual usage of the data store. *`usage`* can be broken down into two parts: first, the frequency of access (modification and usage), and second, the nature of that access. The frequency of access 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 access 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.

Notes

If *`data`* is `NULL`, a data store of the specified size is still created, but its contents remain uninitialized and thus undefined.

Clients must align data elements consistently with the requirements of the client platform, with an additional base-level requirement that an offset within a buffer to a datum comprising N bytes be a multiple of N.

The `GL_ATOMIC_COUNTER_BUFFER` target is available only if the GL version is 4.2 or greater.

The `GL_DISPATCH_INDIRECT_BUFFER` and `GL_SHADER_STORAGE_BUFFER` targets are available only if the GL version is 4.3 or greater.

The `GL_QUERY_BUFFER` target is available only if the GL version is 4.4 or greater.

Errors

`GL_INVALID_ENUM` is generated by **glBufferData** if *`target`* is not one of the accepted buffer targets.

`GL_INVALID_ENUM` is generated if *`usage`* is not `GL_STREAM_DRAW`, `GL_STREAM_READ`, `GL_STREAM_COPY`, `GL_STATIC_DRAW`, `GL_STATIC_READ`, `GL_STATIC_COPY`, `GL_DYNAMIC_DRAW`, `GL_DYNAMIC_READ`, or `GL_DYNAMIC_COPY`.

`GL_INVALID_VALUE` is generated if *`size`* is negative.

`GL_INVALID_OPERATION` is generated by **glBufferData** if the reserved buffer object name 0 is bound to *`target`*.

`GL_INVALID_OPERATION` is generated by **glNamedBufferData** if buffer is not the name of an existing buffer object.

`GL_INVALID_OPERATION` is generated if the `GL_BUFFER_IMMUTABLE_STORAGE` flag of the buffer object is `GL_TRUE`.

`GL_OUT_OF_MEMORY` is generated if the GL is unable to create a data store with the specified *`size`*.

Associated Gets

glGetBufferSubData

glGetBufferParameter with argument `GL_BUFFER_SIZE` or `GL_BUFFER_USAGE`

Version Support

	OpenGL Version											
Function / Feature Name	**2.0**	**2.1**	**3.0**	**3.1**	**3.2**	**3.3**	**4.0**	**4.1**	**4.2**	**4.3**	**4.4**	**4.5**
glBufferData	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
glNamedBufferData	-	-	-	-	-	-	-	-	-	-	-	✔

See Also

glBindBuffer, glBufferSubData, glMapBuffer, glUnmapBuffer

Copyright