GLFW

Reference Manual

Document release 1.4

SUMMARY

This document is a function reference manual for the GLFW 2.4 API. For a description of how to use GLFW you should refer to the *GLFW Users Guide*.

TRADEMARKS

TABLE OF CONTENTS

1. INTRODUCTION

GLFW is a portable API (Application Program Interface) that handles operating system specific tasks related to OpenGL programming. While OpenGL in general is portable, easy to use and often results in tidy and compact code, the operating system specific mechanisms that are required to set up and manage an OpenGL window are quite the opposite. GLFW tries to remedy this by providing the following functionality:

- Opening and maintaining an OpenGL window.
- Keyboard, mouse and joystick input.
- A high precision timer.
- Multi threading support.
- Support for querying and using OpenGL extensions.
- Image file loading support.

All this functionality is implemented as a set of easy-to-use functions, which makes it possible to write an OpenGL application framework in just a few lines of code.

The GLFW API is completely operating system and platform independent, which makes it very simple to port GLFW based OpenGL applications to a variety of platforms. Currently supported platforms are:

- Microsoft Windows® 95/98/ME/NT/2000/XP/.NET Server.
- Unix® or Unix-like systems[1] running the X Window System™, e.g. Linux®, IRIX® and Solaris™.

[1] For threading support GLFW uses the POSIX™ threading package (pthread), which is supported by most modern Unix-like systems.

2. FUNCTION REFERENCE

2.1 GLFW Initialization and Termination

Before any GLFW functions can be used, GLFW must be initialized to ensure proper functionality, and before a program terminates, GLFW has to be terminated in order to free up resources etc.

2.1.1 glfwInit

C language syntax

```
     int glfwInit( void )
```

Parameters
none

Return values
If the function succeeds, *GL_TRUE* is returned.

If the function fails, *GL_FALSE* is returned.

Description
The **glfwInit** function initializes GLFW. No other GLFW functions may be used before this function has been called.

Notes
This function may take several seconds to complete on some systems, while on other systems it may take only a fraction of a second to complete.

2.1.2 glfwTerminate

C language syntax

```
     void glfwTerminate( void )
```

Parameters
none

Return values
none

Description
The function terminates GLFW. Among other things it closes the window, if it is opened, and kills any running threads. This function must be called before a program exits.

Notes
none

2.1.3 glfwGetVersion

C language syntax

```
    void glfwGetVersion( int *major, int *minor, int *rev )
```

Parameters

major

Pointer to an integer that will hold the major version number.

minor

Pointer to an integer that will hold the minor version number.

rev

Pointer to an integer that will hold the revision.

Return values

The function returns the major and minor version numbers and the revision for the currently linked GLFW library.

Description

The function returns the GLFW library version.

Notes

none

2.2 Window Handling

The main functionality of GLFW is to provide a simple interface to OpenGL window management. GLFW can open one window, which can be either a normal desktop window or a fullscreen window.

2.2.1 glfwOpenWindow

C language syntax

```
    int glfwOpenWindow( int width, int height,
                        int redbits, int greenbits, int bluebits,
                        int alphabits, int depthbits, int stencilbits,
                        int mode )
```

Parameters

width

The width of the window. If *width* is zero, it will be calculated as $\frac{4}{3}height$ if *height* is not zero. If both *width* and *height* are zero, *width* will be set to 640.

height

The height of the window. If *height* is zero, it will be calculated as $\frac{3}{4}width$ if *width* is not zero. If both *width* and *height* are zero, *height* will be set to 480.

redbits, greenbits, bluebits

The number of bits to use for each color component of the color buffer. For instance, setting *redbits* = 5, *greenbits* = 6, and *bluebits* = 5 will generate a 16-bit color buffer, if possible.

alphabits

The number of bits to use for the alpha buffer (0 means no alpha buffer).

depthbits

The number of bits to use for the depth buffer (0 means no depth buffer).

stencilbits

The number of bits to use for the stencil buffer (0 means no stencil buffer).

mode

Selects which type of OpenGL window to use. *mode* can be either *GLFW_WINDOW*, which will generate a normal desktop window, or *GLFW_FULLSCREEN*, which will generate a window which covers the entire screen. When *GLFW_FULLSCREEN* is selected, the video mode will be changed to the resolution that closest matches the *width* and *height* parameters.

Return values
If the function succeeds, *GL_TRUE* is returned. If the function fails, *GL_FALSE* is returned.

Description
The function opens a window that best matches the parameters given to the function. How well the resulting window matches the desired window depends mostly on the available hardware and OpenGL drivers. In general, selecting a fullscreen mode has better chances of generating a close match than does a normal desktop window, since GLFW can freely select from all the available video modes. A desktop window is normally restricted to the video mode of the desktop.

Notes
In fullscreen mode the mouse cursor is hidden by default, and any system screensavers are prohibited from starting. In windowed mode the mouse cursor is not hidden, and screensavers are allowed to start. To change the visibility of the mouse cursor, use **glfwEnable** or **glfwDisable** with the argument *GLFW_MOUSE_CURSOR*.

In order to determine the actual properties of an opened window, use **glfwGetWindowParam** and **glfwGetWindowSize** (or **glfwSetWindowSizeCallback**).

2.2.2 glfwOpenWindowHint

C language syntax
```
    void glfwOpenWindowHint( int target, int hint )
```

Parameters
target
Can be any of the constants in the table below:

Name	**Description**
GLFW_REFRESH_RATE	Vertical monitor refresh rate in Hz (only used for fullscreen windows).
GLFW_ACCUM_RED_BITS	Number of bits for the red channel of the accumulator buffer.
GLFW_ACCUM_GREEN_BITS	Number of bits for the green channel of the accumulator buffer.
GLFW_ACCUM_BLUE_BITS	Number of bits for the blue channel of the accumulator buffer.
GLFW_ACCUM_ALPHA_BITS	Number of bits for the alpha channel of the accumulator buffer.
GLFW_AUX_BUFFERS	Number of auxiliary buffers.
GLFW_STEREO	Specify if stereo rendering should be supported (can be GL_TRUE or GL_FALSE).

hint

An integer giving the value of the corresponding target (see table above).

Return values

none

Description

The function sets additional properties for a window that is to be opened. For a hint to be registered, the function must be called before calling **glfwOpenWindow**. When the **glfwOpenWindow** function is called, any hints that were registered with the **glfwOpenWindowHint** function are used for setting the corresponding window properties, and then all hints are reset to "do not care".

Notes

In order to determine the actual properties of an opened window, use **glfwGetWindowParam** (after the window has been opened).

GLFW_STEREO is a hard constraint. If stereo rendering is requested, but no stereo rendering capable pixel formats / visuals are available, **glfwOpenWindow** will fail.

GLFW_REFRESH_RATE is only supported under Windows.

The *GLFW_REFRESH_RATE* property should be used with caution. Most systems have default values for monitor refresh rates that are optimal for the specific system. Specifying the refresh rate can override these settings, which can result in suboptimal operation. The monitor may be unable to display the resulting video signal, or in the worst case it may even be damaged!

2.2.3 glfwCloseWindow

C language syntax

```
void glfwCloseWindow( void )
```

Parameters

none

Return values

none

Description

The function closes an opened window.

Notes

none

2.2.4 glfwSetWindowTitle

C language syntax

```
    void glfwSetWindowTitle( const char *title )
```

Parameters

title

Pointer to a null terminated ISO 8859-1 (8-bit Latin 1) string that holds the title of the window.

Return values

none

Description

The function changes the title of the opened window.

Notes

The title property of a window is often used in situations other than for the window title, such as the title of an application icon when it is in iconified state.

2.2.5 glfwSetWindowSize

C language syntax

```
    void glfwSetWindowSize( int width, int height )
```

Parameters

width

Width of the window.

height

Height of the window.

Return values

none

Description

The function changes the size of an opened window, and calls the window size callback function (if one is registered). If the window is in fullscreen mode, the video mode will be changed to a resolution that closest matches the *width* and *height* parameters (the number of color bits will not be changed).

Notes

none

2.2.6 glfwSetWindowPos

C language syntax

```
     void glfwSetWindowPos( int x, int y )
```

Parameters

x

Horizontal position of the window, relative to the upper left corner of the desktop.

y

Vertical position of the window, relative to the upper left corner of the desktop.

Return values

none

Description

The function changes the position of an opened window. It does not have any effect on a fullscreen window.

Notes

none

2.2.7 glfwGetWindowSize

C language syntax

```
     void glfwGetWindowSize( int *width, int *height )
```

Parameters

width

Pointer to an integer that will hold the width of the window.

height

Pointer to an integer that will hold the height of the window.

Return values

The current width and height of the opened window is returned in the *width* and *height* parameters, respectively.

Description

The function is used for determining the size of an opened window. The returned values are dimensions of the drawing area of the window (i.e. excluding any window borders).

Notes

Even if the size of a fullscreen window does not change once the window has been opened, it does not necessarily have to be the same as the size that was requested using **glfwOpenWindow**. Therefor it is wise to use this function to determine the true size of the window once it has been opened.

2.2.8 glfwSetWindowSizeCallback

C language syntax

```
    void glfwSetWindowSizeCallback( GLFWwindowsizefun cbfun )
```

Parameters

cbfun

Pointer to a callback function which will be called every time the window size changes. The function should have the following C language prototype:

```
void GLFWCALL functionname( int width, int height );
```

Where *functionname* is the name of the callback function, and *width* and *height* are the window dimension passed to the function.

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values

none

Description

The function selects which function to be called upon a window size change event.

Notes

Window size changes are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

2.2.9 glfwGetWindowParam

C language syntax

```
    int glfwGetWindowParam( int param )
```

Parameters

param

A token selecting which parameter the function should return (see below).

Return values

The function returns different parameters depending on the value of *param*. Below is a table of what valid *param* values, and their corresponding return values:

Name	**Return value**
GLFW_OPENED	*GL_TRUE* if window is opened, else *GL_FALSE*.
GLFW_ACTIVE	*GL_TRUE* if window has focus and is active, else *GL_FALSE*.
GLFW_ICONIFIED	*GL_TRUE* if window is iconified, else *GL_FALSE*.
GLFW_ACCELERATED	*GL_TRUE* if window is hardware accelerated.
GLFW_RED_BITS	Number of bits for the red color component.
GLFW_GREEN_BITS	Number of bits for the green color component.
GLFW_BLUE_BITS	Number of bits for the blue color component.
GLFW_ALPHA_BITS	Number of bits for the alpha buffer.
GLFW_DEPTH_BITS	Number of bits for the depth buffer.
GLFW_STENCIL_BITS	Number of bits for the stencil buffer.
GLFW_REFRESH_RATE	Vertical monitor refresh rate in Hz. Zero indicates an unknown or a default refresh rate.
GLFW_ACCUM_RED_BITS	Number of bits for the red channel of the accumulator buffer.
GLFW_ACCUM_GREEN_BITS	Number of bits for the green channel of the accumulator buffer.
GLFW_ACCUM_BLUE_BITS	Number of bits for the blue channel of the accumulator buffer.
GLFW_ACCUM_ALPHA_BITS	Number of bits for the alpha channel of the accumulator buffer.
GLFW_AUX_BUFFERS	Number of auxiliary buffers.
GLFW_STEREO	*GL_TRUE* if stereo rendering is supported, else *GL_FALSE*.

Description
The function is used for acquiring various properties of an opened window.

Notes
GLFW_ACCELERATED is only supported under Windows. Other systems will always generate *GL_TRUE*.

GLFW_REFRESH_RATE is only supported under Windows and XFree86. Other systems will always generate zero (0). With some Windows drivers, zero (0) may be returned, indicating a default refresh rate.

2.2.10 glfwSwapBuffers

C language syntax

```
     void glfwSwapBuffers( void )
```

Parameters

none

Return values

none

Description

The function swaps the back and front color buffers of the window. If *GLFW_AUTO_POLL_EVENTS* is enabled (which is the default), **glfwPollEvents** is called before swapping the front and back buffers.

Notes

none

2.2.11 glfwSwapInterval

C language syntax

```
     void glfwSwapInterval( int interval )
```

Parameters

interval

Minimum number of monitor vertical retraces between each buffer swap performed by **glfwSwapBuffers**. If *interval* is zero, buffer swaps will not be synchronized to the vertical refresh of the monitor (also known as 'VSync off').

Return values

none

Description

The function selects the minimum number of monitor vertical retraces that should occur between two buffer swaps. If the selected swap interval is one, the rate of buffer swaps will never be higher than the vertical refresh rate of the monitor. If the selected swap interval is zero, the rate of buffer swaps is only limited by the speed of the software and the hardware.

Notes

This function will only have an effect on hardware and drivers that support user selection of the swap interval.

2.3 Video Modes

Since GLFW supports video mode changes when using a fullscreen window, it also provides functionality for querying which video modes are supported on a system.

2.3.1 glfwGetVideoModes

C language syntax

```
     int glfwGetVideoModes( GLFWvidmode *list, int maxcount )
```

Parameters

list

A vector of *GLFWvidmode* structures, which will be filled out by the function.

maxcount

Maximum number of video modes that the *list* vector can hold.

Return values

The function returns the number of detected video modes (this number will never exceed *maxcount*). The *list* vector is filled out with the video modes that are supported by the system.

Description

The function returns a list of supported video modes. Each video mode is represented by a *GLFWvidmode* structure, which has the following definition:

```
typedef struct {
    int Width, Height; // Video resolution
    int RedBits;       // Number of bits for the red color component
    int GreenBits;     // Number of bits for the green color component
    int BlueBits;      // Number of bits for the blue color component
} GLFWvidmode;
```

Width and *Height* give the resolution of the video mode, and *RedBits*, *GreenBits* and *BlueBits* give the number of color bits per pixel.

Notes

The returned list is sorted, first by color depth (*RedBits* + *GreenBits* + *BlueBits*), and then by resolution (*Width**Height*), with the lowest resolution, fewest bits per pixel mode first.

2.3.2 glfwGetDesktopMode

C language syntax

```
     void glfwGetDesktopMode( GLFWvidmode *mode )
```

Parameters

mode

Pointer to a *GLFWvidmode* structures, which will be filled out by the function.

Return values
The *GLFWvidmode* structure pointed to by *mode* is filled out with the desktop video mode.

Description
The function returns the desktop video mode in a *GLFWvidmode* structure. See the description of **glfwGetVideoModes** for a description of the *GLFWvidmode* structure.

Notes
The color depth of the desktop display is always reported as the number of bits for each individual color component (red, green and blue), even if the desktop is not using an RGB or RGBA color format. For instance, an indexed 256 color display will report RedBits = 3, GreenBits = 3 and BlueBits = 2, which adds up to 8 bits in total.

2.4 Input Handling

GLFW supports three channels of user input: keyboard input, mouse input and joystick input.

Keyboard and mouse input can be treated either as events, using callback functions, or as state, using functions for polling specific keyboard and mouse states. Regardless of which method is used, all keyboard and mouse input is collected using window event polling.

Joystick input is asynchronous to the keyboard and mouse input, and does not require event polling for keeping up to date joystick information. Also, joystick input is independent of any window, so a window does not have to be opened for joystick input to be used.

2.4.1 glfwPollEvents

C language syntax

```
    void glfwPollEvents( void )
```

Parameters
none

Return values
none

Description
The function is used for polling for events, such as user input and window resize events. Upon calling this function, all window state and keyboard and mouse input state is updated. If any related callback functions are registered, these are called during the call to **glfwPollEvents**.

Notes
glfwPollEvents is called implicitly from **glfwSwapBuffers** if *GLFW_AUTO_POLL_EVENTS* is enabled (default). Thus, if **glfwSwapBuffers** is called frequently (which is normally the case), there is no need to call **glfwPollEvents**.

2.4.2 glfwGetKey

C language syntax

```
    int glfwGetKey( int key )
```

Parameters
key
A keyboard key identifier, which can be either an uppercase printable ISO 8859-1 (Latin 1) character (e.g. 'A', '3' or '.'), or a special key identifier. The following tokens are valid special key identifiers:

Name	**Description**
GLFW_KEY_SPACE	Space
GLFW_KEY_ESC	Escape
GLFW_KEY_Fn	Function key *n* (*n* can be in the range 1..25).
GLFW_KEY_UP	Cursor up
GLFW_KEY_DOWN	Cursor down
GLFW_KEY_LEFT	Cursor left
GLFW_KEY_RIGHT	Cursor right
GLFW_KEY_LSHIFT	Left shift key
GLFW_KEY_RSHIFT	Right shift key
GLFW_KEY_LCTRL	Left control key
GLFW_KEY_RCTRL	Right control key
GLFW_KEY_LALT	Left alternate function key
GLFW_KEY_RALT	Right alternate function key
GLFW_KEY_TAB	Tabulator
GLFW_KEY_ENTER	Enter
GLFW_KEY_BACKSPACE	Backspace
GLFW_KEY_INSERT	Insert
GLFW_KEY_DEL	Delete
GLFW_KEY_PAGEUP	Page up
GLFW_KEY_PAGEDOWN	Page down
GLFW_KEY_HOME	Home
GLFW_KEY_END	End
GLFW_KEY_KP_n	Keypad numeric key *n* (*n* can be in the range 0..9).
GLFW_KEY_KP_DIVIDE	Keypad divide (“/”)
GLFW_KEY_KP_MULTIPLY	Keypad multiply (“*”)
GLFW_KEY_KP_SUBTRACT	Keypad subtract (“-”)
GLFW_KEY_KP_ADD	Keypad add (“+”)
GLFW_KEY_KP_DECIMAL	Keypad decimal (“.” or “,” for some languages)
GLFW_KEY_KP_EQUAL	Keypad equal (“=”)
GLFW_KEY_KP_ENTER	Keypad enter

Return values
The function returns *GLFW_PRESS* if the key is held down, or *GLFW_RELEASE* if the key is not held down.

Description
The function queries the current state of a specific keyboard key. The physical location of each key depends on the system keyboard layout setting.

Notes
The constant *GLFW_KEY_SPACE* is equal to 32, which is the ISO 8859-1 code for space.

Not all key codes are supported on all systems. Also, while some keys are available on some keyboard layouts, they may not be available on other keyboard layouts.

For systems that do not distinguish between left and right versions of modifier keys (shift, alt and control), the left version is used (e.g. *GLFW_KEY_LSHIFT*).

A window must be opened for the function to have any effect, and **glfwPollEvents** or **glfwSwapBuffers** must be called before any keyboard events are recorded and reported by **glfwGetKey**.

2.4.3 glfwGetMouseButton

C language syntax
```
    int glfwGetMouseButton( int button )
```

Parameters
button
A mouse button identifier, which can be one of *GLFW_MOUSE_BUTTON_LEFT*, *GLFW_MOUSE_BUTTON_RIGHT* or *GLFW_MOUSE_BUTTON_MIDDLE*.

Return values
The function returns *GLFW_PRESS* if the mouse button is held down, or *GLFW_RELEASE* if the mouse button is not held down.

Description
The function queries the current state of a specific mouse button.

Notes
A window must be opened for the function to have any effect, and **glfwPollEvents** or **glfwSwapBuffers** must be called before any mouse button events are recorded and reported by **glfwGetMouseButton**.

2.4.4 glfwGetMousePos

C language syntax
```
    void glfwGetMousePos( int *xpos, int *ypos )
```

Parameters
xpos
Pointer to an integer that will be filled out with the horizontal position of the mouse.

ypos
Pointer to an integer that will be filled out with the vertical position of the mouse.

Return values
The function returns the current mouse position in *xpos* and *ypos*.

Description
The function returns the current mouse position. If the cursor is not hidden, the mouse position is the cursor position, relative to the upper left corner of the window and limited to the client area of the window. If the cursor is hidden, the mouse position is a virtual absolute position, not limited to any boundaries except to those implied by the maximum number that can be represented by a signed integer (normally -2147483648 to +2147483647).

Notes
A window must be opened for the function to have any effect, and **glfwPollEvents** or **glfwSwapBuffers** must be called before any mouse movements are recorded and reported by **glfwGetMousePos**.

2.4.5 glfwSetMousePos

C language syntax
```
    void glfwSetMousePos( int xpos, int ypos )
```

Parameters
xpos
Horizontal position of the mouse.

ypos
Vertical position of the mouse.

Return values
none

Description
The function changes the position of the mouse, and calls the mouse position callback function (if one is registered). If the cursor is visible (not disabled), the cursor will be moved to the specified position, relative to the upper left corner of the window client area. If the cursor is hidden (disabled), only the mouse position that is reported by GLFW is changed.

Notes
none

2.4.6 glfwGetMouseWheel

C language syntax

```
     int glfwGetMouseWheel( void )
```

Parameters
none

Return values
The function returns the current mouse wheel position.

Description
The function returns the current mouse wheel position. The mouse wheel can be thought of as a third mouse axis, which is available as a separate wheel or up/down stick on some mice.

Notes
A window must be opened for the function to have any effect, and **glfwPollEvents** or **glfwSwapBuffers** must be called before any mouse wheel movements are recorded and reported by **glfwGetMouseWheel**.

2.4.7 glfwSetMouseWheel

C language syntax

```
     void glfwSetMousePos( int pos )
```

Parameters
pos
    Position of the mouse wheel.

Return values
none

Description
The function changes the position of the mouse wheel, and calls the mouse wheel callback function (if one is registered)

Notes
none

2.4.8 glfwSetKeyCallback

C language syntax

```
     void glfwSetKeyCallback( GLFWkeyfun cbfun )
```

Parameters
cbfun

Pointer to a callback function which will be called every time a key is pressed or released. The function should have the following C language prototype:

```
void GLFWCALL functionname( int key, int action );
```

Where *functionname* is the name of the callback function, *key* is a key identifier, which is an uppercase printable ISO 8859-1 character or a special key identifier (see **glfwGetKey** for more information), and *action* is either *GLFW_PRESS* or *GLFW_RELEASE*.

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values
none

Description
The function selects which function to be called upon a keyboard key event. The callback function is called every time the state of a single key is changed (from released to pressed or vice versa). The reported keys are unaffected by any modifiers (such as shift or alt).

Notes
Keyboard events are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

2.4.9 glfwSetCharCallback

C language syntax
```
void glfwSetCharCallback( GLFWcharfun cbfun )
```

Parameters
cbfun
Pointer to a callback function which will be called every time a printable character is generated by the keyboard. The function should have the following C language prototype:

```
void GLFWCALL functionname( int character, int action );
```

Where *functionname* is the name of the callback function, *character* is a Unicode (ISO 10646) character, and *action* is either *GLFW_PRESS* or *GLFW_RELEASE*.

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values
none

Description

The function selects which function to be called upon a keyboard character event. The callback function is called every time a key that results in a printable Unicode character is pressed or released. Characters are affected by modifiers (such as shift or alt).

Notes
Character events are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

Control characters, such as tab and carriage return, are not reported to the character callback function, since they are not part of the Unicode character set. Use the key callback function for such events (see **glfwSetKeyCallback**).

The Unicode character set supports character codes above 255, so never cast a Unicode character to an eight bit data type (e.g. the C language 'char' type) without first checking that the character code is less than 256. Also note that Unicode character codes 0 to 255 are equal to ISO 8859-1 (Latin 1).

2.4.10 glfwSetMouseButtonCallback

C language syntax
```
    void glfwSetMouseButtonCallback( GLFWmousebuttonfun cbfun )
```

Parameters
cbfun

Pointer to a callback function which will be called every time a mouse button is pressed or released. The function should have the following C language prototype:

```
        void GLFWCALL functionname( int button, int action );
```

Where *functionname* is the name of the callback function, *button* is a mouse button identifier (*GLFW_MOUSE_BUTTON_LEFT*, *GLFW_MOUSE_BUTTON_RIGHT*, or *GLFW_MOUSE_BUTTON_MIDDLE*), and *action* is either *GLFW_PRESS* or *GLFW_RELEASE*.

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values
none

Description
The function selects which function to be called upon a mouse button event.

Notes
Mouse button events are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

2.4.11 glfwSetMousePosCallback

C language syntax

```
    void glfwSetMousePosCallback( GLFWmouseposfun cbfun )
```

Parameters

cbfun

Pointer to a callback function which will be called every time the mouse is moved. The function should have the following C language prototype:

```
void GLFWCALL functionname( int x, int y );
```

Where *functionname* is the name of the callback function, and *x* and *y* are the mouse coordinates (see **glfwGetMousePos** for more information on mouse coordinates).

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values

none

Description

The function selects which function to be called upon a mouse motion event.

Notes

Mouse motion events are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

2.4.12 glfwSetMouseWheelCallback

C language syntax

```
    void glfwSetMouseWheelCallback( GLFWmousewheelfun cbfun )
```

Parameters

cbfun

Pointer to a callback function which will be called every time the mouse wheel is moved. The function should have the following C language prototype:

```
void GLFWCALL functionname( int pos );
```

Where *functionname* is the name of the callback function, and *pos* is the mouse wheel position.

If *cbfun* is *NULL*, any previously selected callback function will be deselected.

Return values

none

Description

The function selects which function to be called upon a mouse wheel event.

Notes
Mouse wheel events are recorded continuously, but only reported when **glfwPollEvents** or **glfwSwapBuffers** is called.

2.4.13 glfwGetJoystickParam

C language syntax
```
    int glfwGetJoystickParam( int joy, int param )
```

Parameters
joy
A joystick identifier, which should be *GLFW_JOYSTICK_n*, where *n* is in the range 1 to 16.

param
A token selecting which parameter the function should return (see below).

Return values
The function returns different parameters depending on the value of *param*. Below is a table of what valid *param* values, and their corresponding return values:

Name	**Return value**
GLFW_PRESENT	*GL_TRUE* if the joystick is connected, else *GL_FALSE*.
GLFW_AXES	Number of axes supported by the joystick.
GLFW_BUTTONS	Number of buttons supported by the joystick.

Description
The function is used for acquiring various properties of a joystick.

Notes
The joystick information is updated every time the function is called.

No window has to be opened for joystick information to be valid.

2.4.14 glfwGetJoystickPos

C language syntax

```
    int glfwGetJoystickPos( int joy, float *pos, int numaxes )
```

Parameters

joy

A joystick identifier, which should be *GLFW_JOYSTICK_n*, where *n* is in the range 1 to 16.

pos

An array that will hold the positional values for all requested axes.

numaxes

Specifies how many axes should be returned.

Return values

The function returns the number of actually returned axes. This is the minimum of *numaxes* and the number of axes supported by the joystick. If the joystick is not supported or connected, the function will return 0 (zero).

Description

The function queries the current position of one or more axes of a joystick. The positional values are returned in an array, where the first element represents the first axis of the joystick (normally the X axis). Each position is in the range -1.0 to 1.0. Where applicable, the positive direction of an axis is right, forward or up, and the negative direction is left, back or down.

If *numaxes* exceeds the number of axes supported by the joystick, or if the joystick is not available, the unused elements in the *pos* array will be set to 0.0 (zero).

Notes

The joystick state is updated every time the function is called, so there is no need to call **glfwPollEvents** for joystick state to be updated.

Use **glfwGetJoystickParam** to retrieve joystick capabilities, such as joystick availability and number of supported axes.

No window has to be opened for joystick input to be valid.

2.4.15 glfwGetJoystickButtons

C language syntax

```
    int glfwGetJoystickButtons( int joy, unsigned char *buttons,
                                int numbuttons )
```

Parameters

joy

A joystick identifier, which should be *GLFW_JOYSTICK_n*, where *n* is in the range 1 to 16.

buttons

An array that will hold the button states for all requested buttons.

numbuttons

Specifies how many buttons should be returned.

Return values

The function returns the number of actually returned buttons. This is the minimum of *numbuttons* and the number of buttons supported by the joystick. If the joystick is not supported or connected, the function will return 0 (zero).

Description

The function queries the current state of one or more buttons of a joystick. The button states are returned in an array, where the first element represents the first button of the joystick. Each state can be either *GLFW_PRESS* or *GLFW_RELEASE*.

If *numbuttons* exceeds the number of buttons supported by the joystick, or if the joystick is not available, the unused elements in the *buttons* array will be set to *GLFW_RELEASE*.

Notes

The joystick state is updated every time the function is called, so there is no need to call **glfwPollEvents** for joystick state to be updated.

Use **glfwGetJoystickParam** to retrieve joystick capabilities, such as joystick availability and number of supported buttons.

No window has to be opened for joystick input to be valid.

2.5 Timing

2.5.1 glfwGetTime

C language syntax

```
    double glfwGetTime( void )
```

Parameters

none

Return values

The function returns the value of the high precision timer. The time is measured in seconds, and is returned as a double precision floating point value.

Description

The function returns the state of a high precision timer. Unless the timer has been set by the **glfwSetTime** function, the time is measured as the number of seconds that have passed since **glfwInit** was called.

Notes

The resolution of the timer depends on which system the program is running on. The worst case resolution is somewhere in the order of 10 ms, while for most systems the resolution should be better than 1 μs.

2.5.2 glfwSetTime

C language syntax

```
    void glfwSetTime( double time )
```

Parameters

time

Time (in seconds) that the timer should be set to.

Return values

none

Description

The function sets the current time of the high precision timer to the specified time. Subsequent calls to **glfwGetTime** will be relative to this time. The time is given in seconds.

Notes

none

2.5.3 glfwSleep

C language syntax

```
     void glfwSleep( double time )
```

Parameters

time

Time, in seconds, to sleep.

Return values

none

Description

The function puts the calling thread to sleep for the requested period of time. Only the calling thread is put to sleep. Other threads within the same process can still execute.

Notes

There is a system dependent minimum time for which it is possible to sleep. This time is generally in the range 1 ms to 20 ms. Using a shorter time as a parameter to **glfwSleep** can give one of two results: either the thread will sleep for the minimum possible sleep time, or the thread will not sleep at all (**glfwSleep** returns immediately). The latter should only happen when very short sleep times are specified.

2.6 OpenGL Extension Support

One of the great features of OpenGL is its support for extensions, which allow independent vendors to supply non-standard functionality in their OpenGL implementations. Using extensions is different under different systems, which is why GLFW has provided an operating system independent interface to querying and using OpenGL extensions.

2.6.1 glfwExtensionSupported

C language syntax

```
    int glfwExtensionSupported( const char *extension )
```

Parameters

ectension

A null terminated string containing the name of an OpenGL extension.

Return values

The function returns *GL_TRUE* if the extension is supported. Otherwise it returns *GL_FALSE*.

Description

The function does a string search in the list of supported OpenGL extensions to find if the specified extension is listed.

Notes

none

2.6.2 glfwGetProcAddress

C language syntax

```
    void * glfwGetProcAddress( const char *procname )
```

Parameters

procname

A null terminated string containing the name of an OpenGL extension function.

Return values

The function returns the address to the specified OpenGL function if it is supported, otherwise *NULL* is returned.

Description

The function acquires the address of an OpenGL extension function. Some (but not all) OpenGL extensions define new API functions, which are usually not available through normal linking. It is therefore necessary to get access to those API functions at runtime.

Notes
glfwGetProcAddress is equivalent to **wglGetProcAddress** under Windows. Under the X Window System it can be implemented by one of the following functions:**glXGetProcAddressARB**, **glXGetProcAddress**, **glXGetProcAddressEXT**, or **dlsym** (if none of them is supported, **glfwGetProcAddress** will always return *NULL*).

2.6.3 glfwGetGLVersion

C language syntax

```
    void glfwGetGLVersion( int *major, int *minor, int *rev )
```

Parameters

major
Pointer to an integer that will hold the major version number.

minor
Pointer to an integer that will hold the minor version number.

rev
Pointer to an integer that will hold the revision.

Return values
The function returns the major and minor version numbers and the revision for the currently used OpenGL implementation.

Description
The function returns the OpenGL implementation version. This is a convenient function that parses the version number information from the string returned by calling **glGetString(GL_VERSION)**. The OpenGL version information can be used to determine what functionality is supported by the used OpenGL implementation.

Notes
none

2.7 Threads

A thread is a separate execution path within a process. All threads within a process share the same address space and resources. Threads execute in parallel, either virtually by means of time-sharing on a single processor, or truly in parallel on several processors. Even on a multi-processor system, time-sharing is employed in order to maximize processor utilization and to ensure fair scheduling. GLFW provides an operating system independent interface to thread management.

2.7.1 glfwCreateThread

C language syntax

```
    GLFWthread glfwCreateThread( GLFWthreadfun fun, void *arg )
```

Parameters

fun

A pointer to a function which acts as the entry point for the new thread. The function should have the following C language prototype:

```
void GLFWCALL functionname( void *arg );
```

Where *functionname* is the name of the callback function, and *arg* is the user supplied argument (see below).

arg

An arbitrary argument for the thread. *arg* will be passed as the argument to the thread function pointed to by *fun*. For instance, *arg* can point to data which is to be processed by the thread.

Return values

The function returns a thread identification number if the thread was created successfully. This number is always positive. If the function fails, a negative number is returned.

Description

The function creates a new thread, which executes within the same address space as the calling process. The thread entry point is specified with the *fun* argument.

Once the thread function *fun* returns, the thread dies.

Notes

Even if the function returns a positive thread ID, indicating that the thread was created successfully, the thread may be unable to execute, for instance if the thread start address is not a valid thread entry point.

2.7.2 glfwDestroyThread

C language syntax

```
     void glfwDestroyThread( GLFWthread ID )
```

Parameters

ID

A thread identification handle, which is returned by **glfwCreateThread** or **glfwGetThreadID**.

Return values

none

Description

The function kills a running thread and removes it from the thread list.

Notes

This function is a very dangerous operation, which may interrupt a thread in the middle of an important operation, and its use is discouraged. You should always try to end a thread in a graceful way using thread communication, and use **glfwWaitThread** in order to wait for the thread to die.

2.7.3 glfwWaitThread

C language syntax

```
     int glfwWaitThread( GLFWthread ID, int waitmode )
```

Parameters

ID

A thread identification handle, which is returned by **glfwCreateThread** or **glfwGetThreadID**.

waitmode

Can be either *GLFW_WAIT* or *GLFW_NOWAIT*.

Return values

The function returns *GL_TRUE* if the specified thread died after the function was called, or the thread did not exist, in which case **glfwWaitThread** will return immediately regardless of *waitmode*. The function returns *GL_FALSE* if *waitmode* is *GLFW_NOWAIT*, and the specified thread exists and is still running.

Description

If *waitmode* is *GLFW_WAIT*, the function waits for a thread to die. If *waitmode* is *GLFW_NOWAIT*, the function checks if a thread exists.

Notes

none

2.7.4 glfwGetThreadID

C language syntax

```
    GLFWthread glfwGetThreadID( void )
```

Parameters

none

Return values

The function returns a thread identifier for the calling thread.

Description

The function determines the thread ID for the calling thread. The ID is the same value as was returned by **glfwCreateThread** when the thread was created.

Notes

none

2.8 Mutexes

Mutexes are used to securely share data between threads. A mutex object can only be owned by one thread at a time. If more than one thread requires access to a mutex object, all but one thread will be put to sleep until they get access to it.

2.8.1 glfwCreateMutex

C language syntax

```
    GLFWmutex glfwCreateMutex( void )
```

Parameters

none

Return values

The function returns a mutex handle, or NULL if the mutex could not be created.

Description

The function creates a mutex object, which can be used to control access to certain data.

Notes

none

2.8.2 glfwDestroyMutex

C language syntax

```
    void glfwDestroyMutex( GLFWmutex mutex )
```

Parameters

mutex

A mutex object handle.

Return values

none

Description

The function destroys a mutex object. After a mutex object has been destroyed, it may no longer be used by any thread.

Notes

none

2.8.3 glfwLockMutex

C language syntax

```
     void glfwLockMutex( GLFWmutex mutex )
```

Parameters

mutex

A mutex object handle.

Return values

none

Description

The function will acquire a lock on the selected mutex object. If the mutex is already locked by another thread, the function will block the calling thread until it is released by the locking thread. Once the function returns, the calling thread has an exclusive lock on the mutex. To release the mutex, call **glfwUnlockMutex**.

Notes

none

2.8.4 glfwUnlockMutex

C language syntax

```
     void glfwUnlockMutex( GLFWmutex mutex )
```

Parameters

mutex

A mutex object handle.

Return values

none

Description

The function releases the lock of a locked mutex object.

Notes

none

2.9 Condition Variables

Condition variables are used to synchronize threads. A thread can wait for a condition variable to be signaled by another thread.

2.9.1 glfwCreateCond

C language syntax

```
    GLFWcond glfwCreateCond( void )
```

Parameters
none

Return values
The function returns a condition variable handle, or NULL if the condition variable could not be created.

Description
The function creates a condition variable object, which can be used to synchronize threads.

Notes
none

2.9.2 glfwDestroyCond

C language syntax

```
    void glfwDestroyCond( GLFWcond cond )
```

Parameters
cond
A condition variable object handle.

Return values
none

Description
The function destroys a condition variable object. After a condition variable object has been destroyed, it may no longer be used by any thread.

Notes
none

2.9.3 glfwWaitCond

C language syntax

```
    void glfwWaitCond( GLFWcond cond, GLFWmutex mutex,
                       double timeout )
```

Parameters

cond
: A condition variable object handle.

mutex
: A mutex object handle.

timeout
: Maximum time to wait for the condition variable. The parameter can either be a positive time (in seconds), or *GLFW_INFINITY*.

Return values

none

Description

The function atomically unlocks the mutex specified by *mutex*, and waits for the condition variable *cond* to be signaled. The thread execution is suspended and does not consume any CPU time until the condition variable is signaled or the amount of time specified by *timeout* has passed. If *timeout* is *GLFW_INFINITY*, **glfwWaitCond** will wait forever for *cond* to be signaled. Before returning to the calling thread, **glfwWaitCond** automatically re-acquires the mutex.

Notes

The mutex specified by *mutex* must be locked by the calling thread on entrance to **glfwWaitCond**.

A condition variable must always be associated with a mutex, to avoid the race condition where a thread prepares to wait on a condition variable and another thread signals the condition just before the first thread actually waits on it.

2.9.4 glfwSignalCond

C language syntax

```
    void glfwSignalCond( GLFWcond cond )
```

Parameters

cond
: A condition variable object handle.

Return values

none

Description
The function restarts one of the threads that are waiting on the condition variable *cond*. If no threads are waiting on *cond*, nothing happens. If several threads are waiting on *cond*, exactly one is restarted, but it is not specified which.

Notes
When several threads are waiting for the condition variable, which thread is started depends on operating system scheduling rules, and may vary from system to system and from time to time.

2.9.5 glfwBroadcastCond

C language syntax
```
    void glfwBroadcastCond( GLFWcond cond )
```

Parameters
cond
    A condition variable object handle.

Return values
none

Description
The function restarts all the threads that are waiting on the condition variable *cond*. Nothing happens if no threads are waiting on *cond*.

Notes
When several threads are waiting for the condition variable, the order in which threads are started depends on operating system scheduling rules, and may vary from system to system and from time to time.

2.10 Image and Texture Loading

In order to aid texture file loading, GLFW has support for loading images from files.

2.10.1 glfwReadImage

C language syntax

```
    int glfwReadImage( const char *name, GLFWimage *img, int flags )
```

Parameters

name

A null terminated ISO 8859-1 (8-bit Latin 1) string holding the name of the file that should be read.

img

Pointer to a *GLFWimage* struct, which will hold the information about the loaded image (if the read was successful).

flags

Flags for controlling the image reading process. Valid flags are:

Name	**Description**
GLFW_NO_RESCALE_BIT	Do not rescale image to closest 2^m x 2^n resolution
GLFW_ORIGIN_UL_BIT	Specifies that the origin of the *loaded* image should be in the upper left corner (default is the lower left corner)

Return values

The function returns *GL_TRUE* if the image was loaded successfully. Otherwise *GL_FALSE* is returned.

Description

The function reads an image from the file specified by the parameter *name* and returns the image information and data in a *GLFWimage* structure, which has the following definition:

```
typedef struct {
    int Width, Height;    // Image dimensions
    int Format;           // OpenGL pixel format
    int BytesPerPixel;    // Number of bytes per pixel
    unsigned char *Data;  // Pointer to pixel data
} GLFWimage;
```

Width and *Height* give the dimensions of the image. *Format* specifies an OpenGL pixel format, which can be *GL_LUMINANCE* (for gray scale images), *GL_RGB* or *GL_RGBA*. *BytesPerPixel* specifies the number of bytes per pixel. *Data* is a pointer to the actual pixel data.

By default the read image is rescaled to the nearest larger 2^m x 2^n resolution using bilinear interpolation, if necessary, which is useful if the image is to be used as an OpenGL texture. This behavior can be disabled by setting the *GLFW_NO_RESCALE_BIT* flag.

Unless the flag *GLFW_ORIGIN_UL_BIT* is set, the first pixel in *img->Data* is the lower left corner of the image. If the flag *GLFW_ORIGIN_UL_BIT* is set, however, the first pixel is the upper left corner.

Notes
glfwReadImage supports the Truevision Targa version 1 file format (.TGA). Supported pixel formats are: 8-bit gray scale, 8-bit paletted (24/32-bit color), 24-bit true color and 32-bit true color + alpha.

Paletted images are translated into true color or true color + alpha pixel formats.

2.10.2 glfwFreeImage

C language syntax
```
    void glfwFreeImage( GLFWimage *img )
```

Parameters
img
    Pointer to a *GLFWimage* struct, which will hold the information about the loaded image (if the read was successful).

Return values
none

Description
The function frees any memory occupied by a loaded image, and clears all the fields of the *GLFWimage* struct. Any image that has been loaded by the **glfwReadImage** or **glfwReadImageF** functions should be deallocated using this function, once the image is not needed anymore.

Notes
none

2.10.3 glfwLoadTexture2D

C language syntax

```
    int glfwLoadTexture2D( const char *name, int flags )
```

Parameters

name

An ISO 8859-1 (8-bit Latin 1) string holding the name of the file that should be loaded.

flags

Flags for controlling the texture loading process. Valid flags are:

Name	**Description**
GLFW_ORIGIN_UL_BIT	Specifies that the origin of the *loaded* image should be in the upper left corner (default is the lower left corner)
GLFW_BUILD_MIPMAPS_BIT	Automatically build and upload all mipmap levels

Return values

The function returns *GL_TRUE* if the texture was loaded successfully. Otherwise *GL_FALSE* is returned.

Description

The function reads an image from the file specified by the parameter *name* and uploads the image to OpenGL texture memory (using the **glTexImage2D** function).

If the *GLFW_BUILD_MIPMAPS_BIT* flag is set, all mipmap levels for the loaded texture are generated and uploaded to texture memory.

Unless the flag *GLFW_ORIGIN_UL_BIT* is set, the first pixel in *img->Data* is the lower left corner of the image. If the flag *GLFW_ORIGIN_UL_BIT* is set, however, the first pixel is the upper left corner.

Notes

glfwLoadTexture2D supports the Truevision Targa version 1 file format (.TGA). Supported pixel formats are: 8-bit gray scale, 8-bit paletted (24/32-bit color), 24-bit true color and 32-bit true color + alpha.

Paletted images are translated into true color or true color + alpha pixel formats.

The read texture is always rescaled to the nearest larger 2^m x 2^n resolution using bilinear interpolation, if necessary, since OpenGL requires textures to have a 2^m x 2^n resolution.

If the *GL_SGIS_generate_mipmap* extension, which is usually hardware accelerated, is supported by the OpenGL implementation it will be used for mipmap generation. Otherwise the mipmaps will be generated by GLFW in software.

2.11 Miscellaneous

2.11.1 glfwEnable/glfwDisable

C language syntax

```
    void glfwEnable( int token )
    void glfwDisable( int token )
```

Parameters

token

A value specifying a feature to enable or disable. The following values are valid tokens:

Name	**Controls**	**Default**
GLFW_AUTO_POLL_EVENTS	Automatic event polling when **glfwSwapBuffers** is called	Enabled
GLFW_KEY_REPEAT	Keyboard key repeat	Disabled
GLFW_MOUSE_CURSOR	Mouse cursor visibility	Enabled in windowed mode. Disabled in fullscreen mode.
GLFW_STICKY_KEYS	Keyboard key “stickiness”	Disabled
GLFW_STICKY_MOUSE_BUTTONS	Mouse button “stickiness”	Disabled
GLFW_SYSTEM_KEYS	Special system key actions	Enabled

Return values

none

Description

glfwEnable is used to enable a certain feature, while **glfwDisable** is used to disable it. For a description of each feature, see the following sub-sections.

Notes

none

2.11.1.1 GLFW_AUTO_POLL_EVENTS

When *GLFW_AUTO_POLL_EVENTS* is enabled, **glfwPollEvents** is automatically called each time that **glfwSwapBuffers** is called.

When *GLFW_AUTO_POLL_EVENTS* is disabled, calling **glfwSwapBuffers** will not result in a call to **glfwPollEvents**. This can be useful if **glfwSwapBuffers** needs to be called from within a callback function, since calling **glfwPollEvents** from a callback function is not allowed.

2.11.1.2 GLFW_KEY_REPEAT

When *GLFW_KEY_REPEAT* is enabled, the key and character callback functions are called repeatedly when a key is held down long enough (according to the system key repeat configuration).

When *GLFW_KEY_REPEAT* is disabled, the key and character callback functions are only called once when a key is pressed (and once when it is released).

2.11.1.3 GLFW_MOUSE_CURSOR

When *GLFW_MOUSE_CURSOR* is enabled, the mouse cursor is visible, and mouse coordinates are relative to the upper left corner of the window and are limited to the drawing area of the window.

When *GLFW_MOUSE_CURSOR* is disabled, the mouse cursor is invisible, and mouse coordinates are not limited to the drawing area of the window.

2.11.1.4 GLFW_STICKY_KEYS

When *GLFW_STICKY_KEYS* is enabled, keys which are pressed will not be released until they are physically released *and* checked with **glfwGetKey**. This behavior makes it possible to catch keys which were pressed and then released again between two calls to **glfwPollEvents** or **glfwSwapBuffers**, which would otherwise have been reported as released. Care should be taken when using this mode, since keys that are not checked with **glfwGetKey** will never be released. Note also that enabling *GLFW_STICKY_KEYS* does not affect the behavior of the keyboard callback functionality.

When *GLFW_STICKY_KEYS* is disabled, the status of a key that is reported by **glfwGetKey** is always the physical state of the key.

2.11.1.5 GLFW_STICKY_MOUSE_BUTTONS

When *GLFW_STICKY_MOUSE_BUTTONS* is enabled, mouse buttons which are pressed will not be released until they are physically released *and* checked with **glfwGetMouseButton**. This behavior makes it possible to catch mouse buttons which were pressed and then released again between two calls to **glfwPollEvents** or **glfwSwapBuffers**, which would otherwise have been reported as released. Care should be taken when using this mode, since mouse buttons that are not checked with **glfwGetMouseButton** will never be released. Note also that enabling *GLFW_STICKY_MOUSE_BUTTONS* does not affect the behavior of the mouse button callback functionality.

When *GLFW_STICKY_MOUSE_BUTTONS* is disabled, the status of a mouse button that is reported by **glfwGetMouseButton** is always the physical state of the mouse button.

2.11.1.6 GLFW_SYSTEM_KEYS

When *GLFW_SYSTEM_KEYS* is enabled, pressing standard system key combinations, such as ALT+TAB under Windows, will give the normal behavior. Note that when ALT+TAB is issued under Windows in this mode so that the GLFW application is deselected when GLFW is operating in fullscreen mode, the GLFW application window will be minimized and the video mode will be set to the original desktop mode. When the GLFW application is re-selected, the video mode will be set to the GLFW video mode again.

When *GLFW_SYSTEM_KEYS* is disabled, pressing standard system key combinations will have no effect, since those key combinations are blocked by GLFW. This mode can be useful in situations when the GLFW program must not be interrupted (normally for games in fullscreen mode).

2.11.2 glfwGetNumberOfProcessors

C language syntax

```
    int glfwGetNumberOfProcessors( void )
```

Parameters

none

Return values

The function returns the number of active processors in the system.

Description

The function determines the number of active processors in the system.

Notes

none