octoDrone

0.1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Bug List

Member glfwSetMonitorCallback (GLFWmonitorfun cbfun)

X11: This callback is not yet called on monitor configuration changes.

X11: This callback is not yet called on monitor configuration changes.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Module Documentation

5.1 Context handling

Typedefs

- typedef void(∗ GLFWglproc) (void)

    Client API function pointer type.
- typedef void(∗ GLFWglproc) (void)

    Client API function pointer type.

Functions

- GLFWAPI void glfwMakeContextCurrent (GLFWwindow ∗window)

    Makes the context of the specified window current for the calling thread.
- GLFWAPI GLFWwindow ∗ glfwGetCurrentContext (void)

    Returns the window whose context is current on the calling thread.
- GLFWAPI void glfwSwapInterval (int interval)

    Sets the swap interval for the current context.
- GLFWAPI int glfwExtensionSupported (const char ∗extension)

    Returns whether the specified extension is available.
- GLFWAPI GLFWglproc glfwGetProcAddress (const char ∗procname)

    Returns the address of the specified function for the current context.

5.1.1 Detailed Description

This is the reference documentation for context related functions. For more information, see the Context handling.

5.1.2 Typedef Documentation

5.1.2.1 typedef void(∗ GLFWglproc) (void)

Client API function pointer type.

Generic function pointer used for returning client API function pointers without forcing a cast from a regular pointer.

5.1.2.2 typedef void(∗ GLFWglproc) (void)

Client API function pointer type.

Generic function pointer used for returning client API function pointers without forcing a cast from a regular pointer.

5.1.3 Function Documentation

5.1.3.1 GLFWAPI int glfwExtensionSupported (const char ∗ *extension*)

Returns whether the specified extension is available.

This function returns whether the specified client API extension is supported by the current OpenGL or OpenGL ES context. It searches both for OpenGL and OpenGL ES extension and platform-specific context creation API extensions.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↩_NO_CURRENT_CONTEXT error.

As this functions retrieves and searches one or more extension strings each call, it is recommended that you cache its results if it is going to be used frequently. The extension strings will not change during the lifetime of a context, so there is no danger in doing this.

Parameters

`in`	*extension*	The ASCII encoded name of the extension.

Returns

`GL_TRUE` if the extension is available, or `GL_FALSE` otherwise.

Thread Safety

This function may be called from any thread.

See also

context_glext
glfwGetProcAddress

Since

Added in GLFW 1.0.

5.1.3.2 GLFWAPI GLFWwindow ∗ glfwGetCurrentContext (void)

Returns the window whose context is current on the calling thread.

This function returns the window whose OpenGL or OpenGL ES context is current on the calling thread.

Returns

The window whose context is current, or `NULL` if no window's context is current.

Thread Safety

This function may be called from any thread.

See also

context_current
glfwMakeContextCurrent

Since

Added in GLFW 3.0.

5.1.3.3 GLFWAPI GLFWglproc glfwGetProcAddress (const char * *procname*)

Returns the address of the specified function for the current context.

This function returns the address of the specified core or extension function, if it is supported by the current context.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↩
_NO_CURRENT_CONTEXT error.

Parameters

`in`	*procname*	The ASCII encoded name of the function.

Returns

The address of the function, or `NULL` if an [error](error_handling) occurred.

Remarks

The address of a given function is not guaranteed to be the same between contexts.
This function may return a non-`NULL` address despite the associated version or extension not being available. Always check the context version or extension string first.

Pointer Lifetime

The returned function pointer is valid until the context is destroyed or the library is terminated.

Thread Safety

This function may be called from any thread.

See also

context_glext
glfwExtensionSupported

Since

Added in GLFW 1.0.

5.1.3.4 GLFWAPI void glfwMakeContextCurrent (GLFWwindow ∗ *window*)

Makes the context of the specified window current for the calling thread.

This function makes the OpenGL or OpenGL ES context of the specified window current on the calling thread. A context can only be made current on a single thread at a time and each thread can have only a single current context at a time.

By default, making a context non-current implicitly forces a pipeline flush. On machines that support `GL_KHR_↩ context_flush_control`, you can control whether a context performs this flush by setting the GLFW_CO↩ NTEXT_RELEASE_BEHAVIOR window hint.

Parameters

`in`	*window*	The window whose context to make current, or `NULL` to detach the current context.

Thread Safety

This function may be called from any thread.

See also

context_current
glfwGetCurrentContext

Since

Added in GLFW 3.0.

5.1.3.5 GLFWAPI void glfwSwapInterval (int *interval*)

Sets the swap interval for the current context.

This function sets the swap interval for the current context, i.e. the number of screen updates to wait from the time glfwSwapBuffers was called before swapping the buffers and returning. This is sometimes called *vertical synchronization*, *vertical retrace synchronization* or just *vsync*.

Contexts that support either of the `WGL_EXT_swap_control_tear` and `GLX_EXT_swap_control_tear` extensions also accept negative swap intervals, which allow the driver to swap even if a frame arrives a little bit late. You can check for the presence of these extensions using glfwExtensionSupported. For more information about swap tearing, see the extension specifications.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↩ _NO_CURRENT_CONTEXT error.

Parameters

`in`	*interval*	The minimum number of screen updates to wait for until the buffers are swapped by glfwSwapBuffers.

Remarks

This function is not called during context creation, leaving the swap interval set to whatever is the default on that platform. This is done because some swap interval extensions used by GLFW do not allow the swap interval to be reset to zero once it has been set to a non-zero value.
Some GPU drivers do not honor the requested swap interval, either because of a user setting that overrides the application's request or due to bugs in the driver.

Thread Safety

This function may be called from any thread.

See also

buffer_swap
glfwSwapBuffers

Since

Added in GLFW 1.0.

5.2 Initialization, version and errors

Modules

- Error codes

Typedefs

- typedef void(∗ GLFWerrorfun) (int, const char ∗)

  The function signature for error callbacks.

- typedef void(∗ GLFWerrorfun) (int, const char ∗)

  The function signature for error callbacks.

Functions

- GLFWAPI int glfwInit (void)

  Initializes the GLFW library.
- GLFWAPI void glfwTerminate (void)

  Terminates the GLFW library.
- GLFWAPI void glfwGetVersion (int ∗major, int ∗minor, int ∗rev)

  Retrieves the version of the GLFW library.
- GLFWAPI const char ∗ glfwGetVersionString (void)

  Returns a string describing the compile-time configuration.
- GLFWAPI GLFWerrorfun glfwSetErrorCallback (GLFWerrorfun cbfun)

  Sets the error callback.

GLFW version macros

- #define GLFW_VERSION_MAJOR 3

  The major version number of the GLFW library.
- #define GLFW_VERSION_MINOR 1

  The minor version number of the GLFW library.
- #define GLFW_VERSION_REVISION 2

  The revision number of the GLFW library.

GLFW version macros

- #define GLFW_VERSION_MAJOR 3

  The major version number of the GLFW library.
- #define GLFW_VERSION_MINOR 1

  The minor version number of the GLFW library.
- #define GLFW_VERSION_REVISION 2

  The revision number of the GLFW library.

5.2.1 Detailed Description

This is the reference documentation for initialization and termination of the library, version management and error handling. For more information, see the intro.

5.2.2 Macro Definition Documentation

5.2.2.1 #define GLFW_VERSION_MAJOR 3

The major version number of the GLFW library.

This is incremented when the API is changed in non-compatible ways.

5.2.2.2 #define GLFW_VERSION_MAJOR 3

The major version number of the GLFW library.

This is incremented when the API is changed in non-compatible ways.

5.2.2.3 #define GLFW_VERSION_MINOR 1

The minor version number of the GLFW library.

This is incremented when features are added to the API but it remains backward-compatible.

5.2.2.4 #define GLFW_VERSION_MINOR 1

The minor version number of the GLFW library.

This is incremented when features are added to the API but it remains backward-compatible.

5.2.2.5 #define GLFW_VERSION_REVISION 2

The revision number of the GLFW library.

This is incremented when a bug fix release is made that does not contain any API changes.

5.2.2.6 #define GLFW_VERSION_REVISION 2

The revision number of the GLFW library.

This is incremented when a bug fix release is made that does not contain any API changes.

5.2.3 Typedef Documentation

5.2.3.1 typedef void(∗ GLFWerrorfun) (int, const char ∗)

The function signature for error callbacks.

This is the function signature for error callback functions.

Parameters

in	*error*	An error code.
in	*description*	A UTF-8 encoded string describing the error.

See also

glfwSetErrorCallback

5.2.3.2 typedef void(∗ GLFWerrorfun) (int, const char ∗)

The function signature for error callbacks.

This is the function signature for error callback functions.

Parameters

in	*error*	An error code.
in	*description*	A UTF-8 encoded string describing the error.

See also

glfwSetErrorCallback

5.2.4 Function Documentation

5.2.4.1 GLFWAPI void glfwGetVersion (int ∗ *major,* int ∗ *minor,* int ∗ *rev*)

Retrieves the version of the GLFW library.

This function retrieves the major, minor and revision numbers of the GLFW library. It is intended for when you are using GLFW as a shared library and want to ensure that you are using the minimum required version.

Any or all of the version arguments may be `NULL`. This function always succeeds.

Parameters

out	*major*	Where to store the major version number, or `NULL`.
out	*minor*	Where to store the minor version number, or `NULL`.
out	*rev*	Where to store the revision number, or `NULL`.

Remarks

This function may be called before glfwInit.

Thread Safety

This function may be called from any thread.

See also

intro_version
glfwGetVersionString

Since

Added in GLFW 1.0.

5.2.4.2 GLFWAPI const char ∗ glfwGetVersionString (void)

Returns a string describing the compile-time configuration.

This function returns the compile-time generated version string of the GLFW library binary. It describes the version, platform, compiler and any platform-specific compile-time options.

Do not use the version string to parse the GLFW library version. The glfwGetVersion function already provides the version of the running library binary.

This function always succeeds.

Returns

The GLFW version string.

Remarks

This function may be called before glfwInit.

Pointer Lifetime

The returned string is static and compile-time generated.

Thread Safety

This function may be called from any thread.

See also

intro_version
glfwGetVersion

Since

Added in GLFW 3.0.

5.2.4.3 GLFWAPI int glfwInit (void)

Initializes the GLFW library.

This function initializes the GLFW library. Before most GLFW functions can be used, GLFW must be initialized, and before an application terminates GLFW should be terminated in order to free any resources allocated during or after initialization.

If this function fails, it calls glfwTerminate before returning. If it succeeds, you should call glfwTerminate before the application exits.

Additional calls to this function after successful initialization but before termination will return `GL_TRUE` immediately.

Returns

`GL_TRUE` if successful, or `GL_FALSE` if an error occurred.

Remarks

OS X: This function will change the current directory of the application to the `Contents/Resources` subdirectory of the application's bundle, if present. This can be disabled with a compile-time option.

Thread Safety

This function may only be called from the main thread.

See also

intro_init
glfwTerminate

Since

Added in GLFW 1.0.

5.2.4.4 GLFWAPI GLFWerrorfun glfwSetErrorCallback (GLFWerrorfun *cbfun*)

Sets the error callback.

This function sets the error callback, which is called with an error code and a human-readable description each time a GLFW error occurs.

The error callback is called on the thread where the error occurred. If you are using GLFW from multiple threads, your error callback needs to be written accordingly.

Because the description string may have been generated specifically for that error, it is not guaranteed to be valid after the callback has returned. If you wish to use it after the callback returns, you need to make a copy.

Once set, the error callback remains set even after the library has been terminated.

Parameters

in	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set.

Remarks

This function may be called before glfwInit.

Thread Safety

This function may only be called from the main thread.

See also

error_handling

Since

Added in GLFW 3.0.

5.2.4.5 GLFWAPI void glfwTerminate (void)

Terminates the GLFW library.

This function destroys all remaining windows and cursors, restores any modified gamma ramps and frees any other allocated resources. Once this function is called, you must again call glfwInit successfully before you will be able to use most GLFW functions.

If GLFW has been successfully initialized, this function should be called before the application exits. If initialization fails, there is no need to call this function, as it is called by glfwInit before it returns failure.

Remarks

This function may be called before glfwInit.

Warning

No window's context may be current on another thread when this function is called.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

intro_init
glfwInit

Since

Added in GLFW 1.0.

5.3 Input handling

Modules

- Keyboard keys
- Modifier key flags
- Mouse buttons
- Joysticks
- Standard cursor shapes

Typedefs

- typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

  The function signature for mouse button callbacks.
- typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

  The function signature for cursor position callbacks.
- typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

  The function signature for cursor enter/leave callbacks.
- typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

  The function signature for scroll callbacks.
- typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

  The function signature for keyboard key callbacks.
- typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

  The function signature for Unicode character callbacks.
- typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

  The function signature for Unicode character with modifiers callbacks.
- typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

  The function signature for file drop callbacks.

- typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

  The function signature for mouse button callbacks.
- typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

  The function signature for cursor position callbacks.
- typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

  The function signature for cursor enter/leave callbacks.
- typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

  The function signature for scroll callbacks.
- typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

  The function signature for keyboard key callbacks.
- typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

  The function signature for Unicode character callbacks.
- typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

  The function signature for Unicode character with modifiers callbacks.
- typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

  The function signature for file drop callbacks.

Functions

- GLFWAPI int glfwGetInputMode (GLFWwindow ∗window, int mode)

  Returns the value of an input option for the specified window.
- GLFWAPI void glfwSetInputMode (GLFWwindow ∗window, int mode, int value)

  Sets an input option for the specified window.
- GLFWAPI int glfwGetKey (GLFWwindow ∗window, int key)

  Returns the last reported state of a keyboard key for the specified window.
- GLFWAPI int glfwGetMouseButton (GLFWwindow ∗window, int button)

  Returns the last reported state of a mouse button for the specified window.
- GLFWAPI void glfwGetCursorPos (GLFWwindow ∗window, double ∗xpos, double ∗ypos)

  Retrieves the position of the cursor relative to the client area of the window.
- GLFWAPI void glfwSetCursorPos (GLFWwindow ∗window, double xpos, double ypos)

  Sets the position of the cursor, relative to the client area of the window.
- GLFWAPI GLFWcursor ∗ glfwCreateCursor (const GLFWimage ∗image, int xhot, int yhot)

  Creates a custom cursor.
- GLFWAPI GLFWcursor ∗ glfwCreateStandardCursor (int shape)

  Creates a cursor with a standard shape.
- GLFWAPI void glfwDestroyCursor (GLFWcursor ∗cursor)

  Destroys a cursor.
- GLFWAPI void glfwSetCursor (GLFWwindow ∗window, GLFWcursor ∗cursor)

  Sets the cursor for the window.
- GLFWAPI GLFWkeyfun glfwSetKeyCallback (GLFWwindow ∗window, GLFWkeyfun cbfun)

  Sets the key callback.
- GLFWAPI GLFWcharfun glfwSetCharCallback (GLFWwindow ∗window, GLFWcharfun cbfun)

  Sets the Unicode character callback.
- GLFWAPI GLFWcharmodsfun glfwSetCharModsCallback (GLFWwindow ∗window, GLFWcharmodsfun cbfun)

  Sets the Unicode character with modifiers callback.
- GLFWAPI GLFWmousebuttonfun glfwSetMouseButtonCallback (GLFWwindow ∗window, GLFWmousebuttonfun cbfun)

  Sets the mouse button callback.
- GLFWAPI GLFWcursorposfun glfwSetCursorPosCallback (GLFWwindow ∗window, GLFWcursorposfun cbfun)

  Sets the cursor position callback.
- GLFWAPI GLFWcursorenterfun glfwSetCursorEnterCallback (GLFWwindow ∗window, GLFWcursorenterfun cbfun)

  Sets the cursor enter/exit callback.
- GLFWAPI GLFWscrollfun glfwSetScrollCallback (GLFWwindow ∗window, GLFWscrollfun cbfun)

  Sets the scroll callback.
- GLFWAPI GLFWdropfun glfwSetDropCallback (GLFWwindow ∗window, GLFWdropfun cbfun)

  Sets the file drop callback.
- GLFWAPI int glfwJoystickPresent (int joy)

  Returns whether the specified joystick is present.
- GLFWAPI const float ∗ glfwGetJoystickAxes (int joy, int ∗count)

  Returns the values of all axes of the specified joystick.
- GLFWAPI const unsigned char ∗ glfwGetJoystickButtons (int joy, int ∗count)

  Returns the state of all buttons of the specified joystick.
- GLFWAPI const char ∗ glfwGetJoystickName (int joy)

  Returns the name of the specified joystick.
- GLFWAPI void glfwSetClipboardString (GLFWwindow ∗window, const char ∗string)

Sets the clipboard to the specified string.

- GLFWAPI const char ∗ glfwGetClipboardString (GLFWwindow ∗window)

  Returns the contents of the clipboard as a string.
- GLFWAPI double glfwGetTime (void)

  Returns the value of the GLFW timer.
- GLFWAPI void glfwSetTime (double time)

  Sets the GLFW timer.

Key and button actions

- #define GLFW_RELEASE 0

  The key or mouse button was released.
- #define GLFW_PRESS 1

  The key or mouse button was pressed.
- #define GLFW_REPEAT 2

  The key was held down until it repeated.

Key and button actions

- #define GLFW_RELEASE 0

  The key or mouse button was released.
- #define GLFW_PRESS 1

  The key or mouse button was pressed.
- #define GLFW_REPEAT 2

  The key was held down until it repeated.

5.3.1 Detailed Description

This is the reference documentation for input related functions and types. For more information, see the Input handling.

5.3.2 Macro Definition Documentation

5.3.2.1 #define GLFW_PRESS 1

The key or mouse button was pressed.

The key or mouse button was pressed.

5.3.2.2 #define GLFW_PRESS 1

The key or mouse button was pressed.

The key or mouse button was pressed.

5.3.2.3 #define GLFW_RELEASE 0

The key or mouse button was released.

The key or mouse button was released.

5.3.2.4 #define GLFW_RELEASE 0

The key or mouse button was released.

The key or mouse button was released.

5.3.2.5 #define GLFW_REPEAT 2

The key was held down until it repeated.

The key was held down until it repeated.

5.3.2.6 #define GLFW_REPEAT 2

The key was held down until it repeated.

The key was held down until it repeated.

5.3.3 Typedef Documentation

5.3.3.1 typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

The function signature for Unicode character callbacks.

This is the function signature for Unicode character callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*codepoint*	The Unicode code point of the character.

See also

glfwSetCharCallback

5.3.3.2 typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

The function signature for Unicode character callbacks.

This is the function signature for Unicode character callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*codepoint*	The Unicode code point of the character.

See also

glfwSetCharCallback

5.3.3.3 typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

The function signature for Unicode character with modifiers callbacks.

This is the function signature for Unicode character with modifiers callback functions. It is called for each input character, regardless of what modifier keys are held down.

Parameters

`in`	*window*	The window that received the event.
`in`	*codepoint*	The Unicode code point of the character.
`in`	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetCharModsCallback

5.3.3.4 typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

The function signature for Unicode character with modifiers callbacks.

This is the function signature for Unicode character with modifiers callback functions. It is called for each input character, regardless of what modifier keys are held down.

Parameters

`in`	*window*	The window that received the event.
`in`	*codepoint*	The Unicode code point of the character.
`in`	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetCharModsCallback

5.3.3.5 typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

The function signature for cursor enter/leave callbacks.

This is the function signature for cursor enter/leave callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*entered*	`GL_TRUE` if the cursor entered the window's client area, or `GL_FALSE` if it left it.

See also

glfwSetCursorEnterCallback

5.3.3.6 typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

The function signature for cursor enter/leave callbacks.

This is the function signature for cursor enter/leave callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*entered*	`GL_TRUE` if the cursor entered the window's client area, or `GL_FALSE` if it left it.

See also

glfwSetCursorEnterCallback

5.3.3.7 typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

The function signature for cursor position callbacks.

This is the function signature for cursor position callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*xpos*	The new x-coordinate, in screen coordinates, of the cursor.
`in`	*ypos*	The new y-coordinate, in screen coordinates, of the cursor.

See also

glfwSetCursorPosCallback

5.3.3.8 typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

The function signature for cursor position callbacks.

This is the function signature for cursor position callback functions.

Parameters

in	*window*	The window that received the event.
in	*xpos*	The new x-coordinate, in screen coordinates, of the cursor.
in	*ypos*	The new y-coordinate, in screen coordinates, of the cursor.

See also

glfwSetCursorPosCallback

5.3.3.9 typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

The function signature for file drop callbacks.

This is the function signature for file drop callbacks.

Parameters

in	*window*	The window that received the event.
in	*count*	The number of dropped files.
in	*paths*	The UTF-8 encoded file and/or directory path names.

See also

glfwSetDropCallback

5.3.3.10 typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

The function signature for file drop callbacks.

This is the function signature for file drop callbacks.

Parameters

in	*window*	The window that received the event.
in	*count*	The number of dropped files.
in	*paths*	The UTF-8 encoded file and/or directory path names.

See also

glfwSetDropCallback

5.3.3.11 typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

The function signature for keyboard key callbacks.

This is the function signature for keyboard key callback functions.

Parameters

in	*window*	The window that received the event.
in	*key*	The keyboard key that was pressed or released.
in	*scancode*	The system-specific scancode of the key.
in	*action*	`GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`.
in	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetKeyCallback

5.3.3.12 typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

The function signature for keyboard key callbacks.

This is the function signature for keyboard key callback functions.

Parameters

in	*window*	The window that received the event.
in	*key*	The keyboard key that was pressed or released.
in	*scancode*	The system-specific scancode of the key.
in	*action*	`GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`.
in	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetKeyCallback

5.3.3.13 typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

The function signature for mouse button callbacks.

This is the function signature for mouse button callback functions.

Parameters

in	*window*	The window that received the event.
in	*button*	The mouse button that was pressed or released.
in	*action*	One of `GLFW_PRESS` or `GLFW_RELEASE`.
in	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetMouseButtonCallback

5.3.3.14 typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

The function signature for mouse button callbacks.

This is the function signature for mouse button callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*button*	The mouse button that was pressed or released.
`in`	*action*	One of `GLFW_PRESS` or `GLFW_RELEASE`.
`in`	*mods*	Bit field describing which modifier keys were held down.

See also

glfwSetMouseButtonCallback

5.3.3.15 typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

The function signature for scroll callbacks.

This is the function signature for scroll callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*xoffset*	The scroll offset along the x-axis.
`in`	*yoffset*	The scroll offset along the y-axis.

See also

glfwSetScrollCallback

5.3.3.16 typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

The function signature for scroll callbacks.

This is the function signature for scroll callback functions.

Parameters

`in`	*window*	The window that received the event.
`in`	*xoffset*	The scroll offset along the x-axis.
`in`	*yoffset*	The scroll offset along the y-axis.

See also

glfwSetScrollCallback

5.3.4 Function Documentation

5.3.4.1 GLFWAPI GLFWcursor ∗ glfwCreateCursor (const GLFWimage ∗ *image,* int *xhot,* int *yhot*)

Creates a custom cursor.

Creates a new custom cursor image that can be set for a window with glfwSetCursor. The cursor can be destroyed with glfwDestroyCursor. Any remaining cursors are destroyed by glfwTerminate.

The pixels are 32-bit, little-endian, non-premultiplied RGBA, i.e. eight bits per channel. They are arranged canonically as packed sequential rows, starting from the top-left corner.

The cursor hotspot is specified in pixels, relative to the upper-left corner of the cursor image. Like all other coordinate systems in GLFW, the X-axis points to the right and the Y-axis points down.

Parameters

`in`	*image*	The desired cursor image.
`in`	*xhot*	The desired x-coordinate, in pixels, of the cursor hotspot.
`in`	*yhot*	The desired y-coordinate, in pixels, of the cursor hotspot.

Returns

The handle of the created cursor, or `NULL` if an error occurred.

Pointer Lifetime

The specified image data is copied before this function returns.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

cursor_object
glfwDestroyCursor
glfwCreateStandardCursor

Since

Added in GLFW 3.1.

5.3.4.2 GLFWAPI GLFWcursor ∗ glfwCreateStandardCursor (int *shape*)

Creates a cursor with a standard shape.

Returns a cursor with a standard shape, that can be set for a window with glfwSetCursor.

Parameters

`in`	*shape*	One of the standard shapes.

Returns

A new cursor ready to use or `NULL` if an error occurred.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

cursor_object
glfwCreateCursor

Since

Added in GLFW 3.1.

5.3.4.3 GLFWAPI void glfwDestroyCursor (GLFWcursor ∗ *cursor*)

Destroys a cursor.

This function destroys a cursor previously created with glfwCreateCursor. Any remaining cursors will be destroyed by glfwTerminate.

Parameters

`in`	*cursor*	The cursor object to destroy.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

cursor_object
glfwCreateCursor

Since

Added in GLFW 3.1.

5.3.4.4 GLFWAPI const char ∗ glfwGetClipboardString (GLFWwindow ∗ *window*)

Returns the contents of the clipboard as a string.

This function returns the contents of the system clipboard, if it contains or is convertible to a UTF-8 encoded string. If the clipboard is empty or if its contents cannot be converted, `NULL` is returned and a GLFW_FORMAT_UNAV↩ AILABLE error is generated.

Parameters

`in`	*window*	The window that will request the clipboard contents.

Returns

The contents of the clipboard as a UTF-8 encoded string, or `NULL` if an error occurred.

Pointer Lifetime

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the next call to glfwGetClipboardString or glfwSetClipboardString, or until the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

clipboard
glfwSetClipboardString

Since

Added in GLFW 3.0.

5.3.4.5 GLFWAPI void glfwGetCursorPos (GLFWwindow ∗ *window,* double ∗ *xpos,* double ∗ *ypos*)

Retrieves the position of the cursor relative to the client area of the window.

This function returns the position of the cursor, in screen coordinates, relative to the upper-left corner of the client area of the specified window.

If the cursor is disabled (with `GLFW_CURSOR_DISABLED`) then the cursor position is unbounded and limited only by the minimum and maximum values of a `double`.

The coordinate can be converted to their integer equivalents with the `floor` function. Casting directly to an integer type works for positive coordinates, but fails for negative ones.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

Parameters

`in`	*window*	The desired window.
`out`	*xpos*	Where to store the cursor x-coordinate, relative to the left edge of the client area, or `NULL`.
`out`	*ypos*	Where to store the cursor y-coordinate, relative to the to top edge of the client area, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

cursor_pos
glfwSetCursorPos

Since

Added in GLFW 3.0. Replaces `glfwGetMousePos`.

5.3.4.6 GLFWAPI int glfwGetInputMode (GLFWwindow ∗ *window*, int *mode*)

Returns the value of an input option for the specified window.

This function returns the value of an input option for the specified window. The mode must be one of `GLFW_CU←RSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.

Parameters

`in`	*window*	The window to query.
`in`	*mode*	One of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.

Thread Safety

This function may only be called from the main thread.

See also

glfwSetInputMode

Since

Added in GLFW 3.0.

5.3.4.7 GLFWAPI const float ∗ glfwGetJoystickAxes (int *joy*, int ∗ *count*)

Returns the values of all axes of the specified joystick.

This function returns the values of all axes of the specified joystick. Each element in the array is a value between -1.0 and 1.0.

Parameters

`in`	*joy*	The joystick to query.
`out`	*count*	Where to store the number of axis values in the returned array. This is set to zero if an error occurred.

Returns

An array of axis values, or `NULL` if the joystick is not present.

Pointer Lifetime

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

joystick_axis

Since

Added in GLFW 3.0. Replaces `glfwGetJoystickPos`.

5.3.4.8 GLFWAPI const unsigned char ∗ glfwGetJoystickButtons (int *joy,* int ∗ *count*)

Returns the state of all buttons of the specified joystick.

This function returns the state of all buttons of the specified joystick. Each element in the array is either `GLFW_P`↩
`RESS` or `GLFW_RELEASE`.

Parameters

`in`	*joy*	The joystick to query.
`out`	*count*	Where to store the number of button states in the returned array. This is set to zero if an error occurred.

Returns

An array of button states, or `NULL` if the joystick is not present.

Pointer Lifetime

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

joystick_button

Since

Added in GLFW 2.2.

GLFW 3: Changed to return a dynamic array.

5.3.4.9 GLFWAPI const char ∗ glfwGetJoystickName (int *joy*)

Returns the name of the specified joystick.

This function returns the name, encoded as UTF-8, of the specified joystick. The returned string is allocated and freed by GLFW. You should not free it yourself.

Parameters

`in`	*joy*	The joystick to query.

Returns

The UTF-8 encoded name of the joystick, or `NULL` if the joystick is not present.

Pointer Lifetime

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

joystick_name

Since

Added in GLFW 3.0.

5.3.4.10 GLFWAPI int glfwGetKey (GLFWwindow ∗ *window,* int *key*)

Returns the last reported state of a keyboard key for the specified window.

This function returns the last state reported for the specified key to the specified window. The returned state is one of `GLFW_PRESS` or `GLFW_RELEASE`. The higher-level action `GLFW_REPEAT` is only reported to the key callback.

If the `GLFW_STICKY_KEYS` input mode is enabled, this function returns `GLFW_PRESS` the first time you call it for a key that was pressed, even if that key has already been released.

The key functions deal with physical keys, with key tokens named after their use on the standard US keyboard layout. If you want to input text, use the Unicode character callback instead.

The modifier key bit masks are not key tokens and cannot be used with this function.

Parameters

`in`	*window*	The desired window.
`in`	*key*	The desired keyboard key. `GLFW_KEY_UNKNOWN` is not a valid key for this function.

Returns

One of `GLFW_PRESS` or `GLFW_RELEASE`.

Thread Safety

This function may only be called from the main thread.

See also

input_key

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.3.4.11 GLFWAPI int glfwGetMouseButton (GLFWwindow ∗ *window,* int *button*)

Returns the last reported state of a mouse button for the specified window.

This function returns the last state reported for the specified mouse button to the specified window. The returned state is one of `GLFW_PRESS` or `GLFW_RELEASE`.

If the `GLFW_STICKY_MOUSE_BUTTONS` input mode is enabled, this function `GLFW_PRESS` the first time you call it for a mouse button that was pressed, even if that mouse button has already been released.

Parameters

`in`	*window*	The desired window.
`in`	*button*	The desired mouse button.

Returns

One of `GLFW_PRESS` or `GLFW_RELEASE`.

Thread Safety

This function may only be called from the main thread.

See also

input_mouse_button

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.3.4.12 GLFWAPI double glfwGetTime (void)

Returns the value of the GLFW timer.

This function returns the value of the GLFW timer. Unless the timer has been set using glfwSetTime, the timer measures time elapsed since GLFW was initialized.

The resolution of the timer is system dependent, but is usually on the order of a few micro- or nanoseconds. It uses the highest-resolution monotonic time source on each supported platform.

Returns

The current value, in seconds, or zero if an error occurred.

Thread Safety

This function may be called from any thread. Access is not synchronized.

See also

time

Since

Added in GLFW 1.0.

5.3.4.13 GLFWAPI int glfwJoystickPresent (int *joy*)

Returns whether the specified joystick is present.

This function returns whether the specified joystick is present.

Parameters

`in`	*joy*	The joystick to query.

Returns

`GL_TRUE` if the joystick is present, or `GL_FALSE` otherwise.

Thread Safety

This function may only be called from the main thread.

See also

joystick

Since

Added in GLFW 3.0. Replaces `glfwGetJoystickParam`.

5.3.4.14 GLFWAPI GLFWcharfun glfwSetCharCallback (GLFWwindow ∗ *window,* GLFWcharfun *cbfun*)

Sets the Unicode character callback.

This function sets the character callback of the specified window, which is called when a Unicode character is input.

The character callback is intended for Unicode text input. As it deals with characters, it is keyboard layout dependent, whereas the key callback is not. Characters do not map 1:1 to physical keys, as a key may produce zero, one or more characters. If you want to know whether a specific physical key was pressed or released, see the key callback instead.

The character callback behaves as system text input normally does and will not be called if modifier keys are held down that would prevent normal text input on that platform, for example a Super (Command) key on OS X or Alt key on Windows. There is a character with modifiers callback that receives these events.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

input_char

Since

Added in GLFW 2.4.

GLFW 3: Added window handle parameter. Updated callback signature.

5.3.4.15 GLFWAPI **GLFWcharmodsfun** glfwSetCharModsCallback (**GLFWwindow** ∗ *window,* **GLFWcharmodsfun** *cbfun*)

Sets the Unicode character with modifiers callback.

This function sets the character with modifiers callback of the specified window, which is called when a Unicode character is input regardless of what modifier keys are used.

The character with modifiers callback is intended for implementing custom Unicode character input. For regular Unicode text input, see the character callback. Like the character callback, the character with modifiers callback deals with characters and is keyboard layout dependent. Characters do not map 1:1 to physical keys, as a key may produce zero, one or more characters. If you want to know whether a specific physical key was pressed or released, see the key callback instead.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or an error occurred.

Thread Safety

This function may only be called from the main thread.

See also

input_char

Since

Added in GLFW 3.1.

5.3.4.16 GLFWAPI void glfwSetClipboardString (**GLFWwindow** ∗ *window,* const char ∗ *string*)

Sets the clipboard to the specified string.

This function sets the system clipboard to the specified, UTF-8 encoded string.

Parameters

`in`	*window*	The window that will own the clipboard contents.
`in`	*string*	A UTF-8 encoded string.

Pointer Lifetime

The specified string is copied before this function returns.

Thread Safety

This function may only be called from the main thread.

See also

clipboard
glfwGetClipboardString

Since

Added in GLFW 3.0.

5.3.4.17 GLFWAPI void glfwSetCursor (GLFWwindow ∗ *window*, GLFWcursor ∗ *cursor*)

Sets the cursor for the window.

This function sets the cursor image to be used when the cursor is over the client area of the specified window. The set cursor will only be visible when the cursor mode of the window is `GLFW_CURSOR_NORMAL`.

On some platforms, the set cursor may not be visible unless the window also has input focus.

Parameters

in	*window*	The window to set the cursor for.
in	*cursor*	The cursor to set, or `NULL` to switch back to the default arrow cursor.

Thread Safety

This function may only be called from the main thread.

See also

cursor_object

Since

Added in GLFW 3.1.

5.3.4.18 GLFWAPI GLFWcursorenterfun glfwSetCursorEnterCallback (GLFWwindow ∗ *window*, GLFWcursorenterfun *cbfun*)

Sets the cursor enter/exit callback.

This function sets the cursor boundary crossing callback of the specified window, which is called when the cursor enters or leaves the client area of the window.

Parameters

in	*window*	The window whose callback to set.
in	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

cursor_enter

Since

Added in GLFW 3.0.

5.3.4.19 GLFWAPI void glfwSetCursorPos (**GLFWwindow** ∗ *window,* double *xpos,* double *ypos*)

Sets the position of the cursor, relative to the client area of the window.

This function sets the position, in screen coordinates, of the cursor relative to the upper-left corner of the client area of the specified window. The window must have input focus. If the window does not have input focus when this function is called, it fails silently.

Do not use this function to implement things like camera controls. GLFW already provides the `GLFW_CURS↩ OR_DISABLED` cursor mode that hides the cursor, transparently re-centers it and provides unconstrained cursor motion. See glfwSetInputMode for more information.

If the cursor mode is `GLFW_CURSOR_DISABLED` then the cursor position is unconstrained and limited only by the minimum and maximum values of a `double`.

Parameters

in	*window*	The desired window.
in	*xpos*	The desired x-coordinate, relative to the left edge of the client area.
in	*ypos*	The desired y-coordinate, relative to the top edge of the client area.

Remarks

X11: Due to the asynchronous nature of X11, it may take a moment for the window focus event to arrive. This means you may not be able to set the cursor position directly after window creation.

Thread Safety

This function may only be called from the main thread.

See also

cursor_pos
glfwGetCursorPos

Since

Added in GLFW 3.0. Replaces `glfwSetMousePos`.

5.3.4.20 GLFWAPI GLFWcursorposfun glfwSetCursorPosCallback (GLFWwindow ∗ *window,* GLFWcursorposfun *cbfun*)

Sets the cursor position callback.

This function sets the cursor position callback of the specified window, which is called when the cursor is moved. The callback is provided with the position, in screen coordinates, relative to the upper-left corner of the client area of the window.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

cursor_pos

Since

Added in GLFW 3.0. Replaces `glfwSetMousePosCallback`.

5.3.4.21 GLFWAPI GLFWdropfun glfwSetDropCallback (GLFWwindow ∗ *window,* GLFWdropfun *cbfun*)

Sets the file drop callback.

This function sets the file drop callback of the specified window, which is called when one or more dragged files are dropped on the window.

Because the path array and its strings may have been generated specifically for that event, they are not guaranteed to be valid after the callback has returned. If you wish to use them after the callback returns, you need to make a deep copy.

Parameters

in	*window*	The window whose callback to set.
in	*cbfun*	The new file drop callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

path_drop

Since

Added in GLFW 3.1.

5.3.4.22 GLFWAPI void glfwSetInputMode (GLFWwindow ∗ *window,* int *mode,* int *value*)

Sets an input option for the specified window.

This function sets an input mode option for the specified window. The mode must be one of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.

If the mode is `GLFW_CURSOR`, the value must be one of the following cursor modes:

- `GLFW_CURSOR_NORMAL` makes the cursor visible and behaving normally.
- `GLFW_CURSOR_HIDDEN` makes the cursor invisible when it is over the client area of the window but does not restrict the cursor from leaving.
- `GLFW_CURSOR_DISABLED` hides and grabs the cursor, providing virtual and unlimited cursor movement. This is useful for implementing for example 3D camera controls.

If the mode is `GLFW_STICKY_KEYS`, the value must be either `GL_TRUE` to enable sticky keys, or `GL_FALSE` to disable it. If sticky keys are enabled, a key press will ensure that glfwGetKey returns `GLFW_PRESS` the next time it is called even if the key had been released before the call. This is useful when you are only interested in whether keys have been pressed but not when or in which order.

If the mode is `GLFW_STICKY_MOUSE_BUTTONS`, the value must be either `GL_TRUE` to enable sticky mouse buttons, or `GL_FALSE` to disable it. If sticky mouse buttons are enabled, a mouse button press will ensure that glfwGetMouseButton returns `GLFW_PRESS` the next time it is called even if the mouse button had been released before the call. This is useful when you are only interested in whether mouse buttons have been pressed but not when or in which order.

Parameters

in	*window*	The window whose input mode to set.
in	*mode*	One of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.
in	*value*	The new value of the specified input mode.

Thread Safety

This function may only be called from the main thread.

See also

glfwGetInputMode

Since

Added in GLFW 3.0. Replaces `glfwEnable` and `glfwDisable`.

5.3.4.23 GLFWAPI GLFWkeyfun glfwSetKeyCallback (GLFWwindow * *window*, GLFWkeyfun *cbfun*)

Sets the key callback.

This function sets the key callback of the specified window, which is called when a key is pressed, repeated or released.

The key functions deal with physical keys, with layout independent key tokens named after their values in the standard US keyboard layout. If you want to input text, use the character callback instead.

When a window loses input focus, it will generate synthetic key release events for all pressed keys. You can tell these events from user-generated events by the fact that the synthetic ones are generated after the focus loss event has been processed, i.e. after the window focus callback has been called.

The scancode of a key is specific to that platform or sometimes even to that machine. Scancodes are intended to allow users to bind keys that don't have a GLFW key token. Such keys have `key` set to `GLFW_KEY_UNKNOWN`, their state is not saved and so it cannot be queried with glfwGetKey.

Sometimes GLFW needs to generate synthetic key events, in which case the scancode may be zero.

Parameters

in	*window*	The window whose callback to set.
in	*cbfun*	The new key callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

input_key

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter. Updated callback signature.

5.3.4.24 GLFWAPI GLFWmousebuttonfun glfwSetMouseButtonCallback (GLFWwindow * *window*, GLFWmousebuttonfun *cbfun*)

Sets the mouse button callback.

This function sets the mouse button callback of the specified window, which is called when a mouse button is pressed or released.

When a window loses input focus, it will generate synthetic mouse button release events for all pressed mouse buttons. You can tell these events from user-generated events by the fact that the synthetic ones are generated after the focus loss event has been processed, i.e. after the window focus callback has been called.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

input_mouse_button

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter. Updated callback signature.

5.3.4.25 GLFWAPI GLFWscrollfun glfwSetScrollCallback (GLFWwindow * *window*, GLFWscrollfun *cbfun*)

Sets the scroll callback.

This function sets the scroll callback of the specified window, which is called when a scrolling device is used, such as a mouse wheel or scrolling area of a touchpad.

The scroll callback receives all scrolling input, like that from a mouse wheel or a touchpad scrolling area.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new scroll callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

scrolling

Since

Added in GLFW 3.0. Replaces `glfwSetMouseWheelCallback`.

5.3.4.26 GLFWAPI void glfwSetTime (double *time*)

Sets the GLFW timer.

This function sets the value of the GLFW timer. It then continues to count up from that value. The value must be a positive finite number less than or equal to 18446744073.0, which is approximately 584.5 years.

Parameters

`in`	*time*	The new value, in seconds.

Remarks

The upper limit of the timer is calculated as floor((2^{64} - 1) / 10^9) and is due to implementations storing nanoseconds in 64 bits. The limit may be increased in the future.

Thread Safety

This function may only be called from the main thread.

See also

time

Since

Added in GLFW 2.2.

5.4 Monitor handling

Classes

- struct GLFWvidmode

  Video mode type.
- struct GLFWgammaramp

  Gamma ramp.

Typedefs

- typedef struct GLFWmonitor GLFWmonitor

  Opaque monitor object.
- typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

  The function signature for monitor configuration callbacks.
- typedef struct GLFWvidmode GLFWvidmode

  Video mode type.
- typedef struct GLFWgammaramp GLFWgammaramp

  Gamma ramp.
- typedef struct GLFWmonitor GLFWmonitor

  Opaque monitor object.
- typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

  The function signature for monitor configuration callbacks.
- typedef struct GLFWvidmode GLFWvidmode

  Video mode type.
- typedef struct GLFWgammaramp GLFWgammaramp

  Gamma ramp.

Functions

- GLFWAPI GLFWmonitor ∗∗ glfwGetMonitors (int ∗count)

  Returns the currently connected monitors.
- GLFWAPI GLFWmonitor ∗ glfwGetPrimaryMonitor (void)

  Returns the primary monitor.
- GLFWAPI void glfwGetMonitorPos (GLFWmonitor ∗monitor, int ∗xpos, int ∗ypos)

  Returns the position of the monitor's viewport on the virtual screen.
- GLFWAPI void glfwGetMonitorPhysicalSize (GLFWmonitor ∗monitor, int ∗widthMM, int ∗heightMM)

  Returns the physical size of the monitor.
- GLFWAPI const char ∗ glfwGetMonitorName (GLFWmonitor ∗monitor)

  Returns the name of the specified monitor.
- GLFWAPI GLFWmonitorfun glfwSetMonitorCallback (GLFWmonitorfun cbfun)

  Sets the monitor configuration callback.
- GLFWAPI const GLFWvidmode ∗ glfwGetVideoModes (GLFWmonitor ∗monitor, int ∗count)

  Returns the available video modes for the specified monitor.
- GLFWAPI const GLFWvidmode ∗ glfwGetVideoMode (GLFWmonitor ∗monitor)

  Returns the current mode of the specified monitor.
- GLFWAPI void glfwSetGamma (GLFWmonitor ∗monitor, float gamma)

  Generates a gamma ramp and sets it for the specified monitor.
- GLFWAPI const GLFWgammaramp ∗ glfwGetGammaRamp (GLFWmonitor ∗monitor)

  Returns the current gamma ramp for the specified monitor.
- GLFWAPI void glfwSetGammaRamp (GLFWmonitor ∗monitor, const GLFWgammaramp ∗ramp)

  Sets the current gamma ramp for the specified monitor.

5.4.1 Detailed Description

This is the reference documentation for monitor related functions and types. For more information, see the Monitor handling.

5.4.2 Typedef Documentation

5.4.2.1 typedef struct **GLFWgammaramp GLFWgammaramp**

Gamma ramp.

This describes the gamma ramp for a monitor.

See also

glfwGetGammaRamp glfwSetGammaRamp

5.4.2.2 typedef struct **GLFWgammaramp GLFWgammaramp**

Gamma ramp.

This describes the gamma ramp for a monitor.

See also

glfwGetGammaRamp glfwSetGammaRamp

5.4.2.3 typedef struct **GLFWmonitor GLFWmonitor**

Opaque monitor object.

Opaque monitor object.

5.4.2.4 typedef struct **GLFWmonitor GLFWmonitor**

Opaque monitor object.

Opaque monitor object.

5.4.2.5 typedef void(∗ GLFWmonitorfun) (**GLFWmonitor** ∗, int)

The function signature for monitor configuration callbacks.

This is the function signature for monitor configuration callback functions.

Parameters

`in`	*monitor*	The monitor that was connected or disconnected.
`in`	*event*	One of `GLFW_CONNECTED` or `GLFW_DISCONNECTED`.

See also

glfwSetMonitorCallback

5.4.2.6 typedef void(∗ GLFWmonitorfun) (**GLFWmonitor** ∗, int)

The function signature for monitor configuration callbacks.

This is the function signature for monitor configuration callback functions.

Parameters

`in`	*monitor*	The monitor that was connected or disconnected.
`in`	*event*	One of `GLFW_CONNECTED` or `GLFW_DISCONNECTED`.

See also

glfwSetMonitorCallback

5.4.2.7 typedef struct **GLFWvidmode GLFWvidmode**

Video mode type.

This describes a single video mode.

5.4.2.8 typedef struct **GLFWvidmode GLFWvidmode**

Video mode type.

This describes a single video mode.

5.4.3 Function Documentation

5.4.3.1 GLFWAPI const **GLFWgammaramp** ∗ glfwGetGammaRamp (**GLFWmonitor** ∗ *monitor*)

Returns the current gamma ramp for the specified monitor.

This function returns the current gamma ramp of the specified monitor.

Parameters

`in`	*monitor*	The monitor to query.

Returns

The current gamma ramp, or `NULL` if an error occurred.

Pointer Lifetime

The returned structure and its arrays are allocated and freed by GLFW. You should not free them yourself. They are valid until the specified monitor is disconnected, this function is called again for that monitor or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

monitor_gamma

Since

Added in GLFW 3.0.

5.4.3.2 GLFWAPI const char ∗ glfwGetMonitorName (GLFWmonitor ∗ *monitor*)

Returns the name of the specified monitor.

This function returns a human-readable name, encoded as UTF-8, of the specified monitor. The name typically reflects the make and model of the monitor and is not guaranteed to be unique among the connected monitors.

Parameters

`in`	*monitor*	The monitor to query.

Returns

The UTF-8 encoded name of the monitor, or `NULL` if an error occurred.

Pointer Lifetime

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

monitor_properties

Since

Added in GLFW 3.0.

5.4.3.3 GLFWAPI void glfwGetMonitorPhysicalSize (GLFWmonitor ∗ *monitor,* int ∗ *widthMM,* int ∗ *heightMM*)

Returns the physical size of the monitor.

This function returns the size, in millimetres, of the display area of the specified monitor.

Some systems do not provide accurate monitor size information, either because the monitor `EDID` data is incorrect or because the driver does not report it accurately.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

Parameters

`in`	*monitor*	The monitor to query.
`out`	*widthMM*	Where to store the width, in millimetres, of the monitor's display area, or `NULL`.
`out`	*heightMM*	Where to store the height, in millimetres, of the monitor's display area, or `NULL`.

Remarks

Windows: The OS calculates the returned physical size from the current resolution and system DPI instead of querying the monitor EDID data.

Thread Safety

This function may only be called from the main thread.

See also

monitor_properties

Since

Added in GLFW 3.0.

5.4.3.4 GLFWAPI void glfwGetMonitorPos (GLFWmonitor ∗ *monitor,* int ∗ *xpos,* int ∗ *ypos*)

Returns the position of the monitor's viewport on the virtual screen.

This function returns the position, in screen coordinates, of the upper-left corner of the specified monitor.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

Parameters

`in`	*monitor*	The monitor to query.
`out`	*xpos*	Where to store the monitor x-coordinate, or `NULL`.
`out`	*ypos*	Where to store the monitor y-coordinate, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

monitor_properties

Since

Added in GLFW 3.0.

5.4.3.5 GLFWAPI GLFWmonitor ** glfwGetMonitors (int * *count*)

Returns the currently connected monitors.

This function returns an array of handles for all currently connected monitors. The primary monitor is always first in the returned array. If no monitors were found, this function returns `NULL`.

Parameters

`out`	*count*	Where to store the number of monitors in the returned array. This is set to zero if an error occurred.

Returns

An array of monitor handles, or `NULL` if no monitors were found or if an error occurred.

Pointer Lifetime

The returned array is allocated and freed by GLFW. You should not free it yourself. It is guaranteed to be valid only until the monitor configuration changes or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

monitor_monitors
monitor_event
glfwGetPrimaryMonitor

Since

Added in GLFW 3.0.

5.4.3.6 GLFWAPI **GLFWmonitor** ∗ glfwGetPrimaryMonitor (void)

Returns the primary monitor.

This function returns the primary monitor. This is usually the monitor where elements like the task bar or global menu bar are located.

Returns

The primary monitor, or `NULL` if no monitors were found or if an error occurred.

Thread Safety

This function may only be called from the main thread.

Remarks

The primary monitor is always first in the array returned by glfwGetMonitors.

See also

monitor_monitors
glfwGetMonitors

Since

Added in GLFW 3.0.

5.4.3.7 GLFWAPI const **GLFWvidmode** ∗ glfwGetVideoMode (**GLFWmonitor** ∗ *monitor*)

Returns the current mode of the specified monitor.

This function returns the current video mode of the specified monitor. If you have created a full screen window for that monitor, the return value will depend on whether that window is iconified.

Parameters

`in`	*monitor*	The monitor to query.

Returns

The current mode of the monitor, or `NULL` if an error occurred.

Pointer Lifetime

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

monitor_modes
glfwGetVideoModes

Since

Added in GLFW 3.0. Replaces `glfwGetDesktopMode`.

5.4.3.8 GLFWAPI const GLFWvidmode ∗ glfwGetVideoModes (GLFWmonitor ∗ *monitor*, int ∗ *count*)

Returns the available video modes for the specified monitor.

This function returns an array of all video modes supported by the specified monitor. The returned array is sorted in ascending order, first by color bit depth (the sum of all channel depths) and then by resolution area (the product of width and height).

Parameters

`in`	*monitor*	The monitor to query.
`out`	*count*	Where to store the number of video modes in the returned array. This is set to zero if an error occurred.

Returns

An array of video modes, or `NULL` if an error occurred.

Pointer Lifetime

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected, this function is called again for that monitor or the library is terminated.

Thread Safety

This function may only be called from the main thread.

See also

monitor_modes
glfwGetVideoMode

Since

Added in GLFW 1.0.

GLFW 3: Changed to return an array of modes for a specific monitor.

5.4.3.9 GLFWAPI void glfwSetGamma (GLFWmonitor ∗ *monitor*, float *gamma*)

Generates a gamma ramp and sets it for the specified monitor.

This function generates a 256-element gamma ramp from the specified exponent and then calls glfwSetGamma↩ Ramp with it. The value must be a finite number greater than zero.

Parameters

`in`	*monitor*	The monitor whose gamma ramp to set.
`in`	*gamma*	The desired exponent.

Thread Safety

This function may only be called from the main thread.

See also

monitor_gamma

Since

Added in GLFW 3.0.

5.4.3.10 GLFWAPI void glfwSetGammaRamp (**GLFWmonitor** ∗ *monitor,* const **GLFWgammaramp** ∗ *ramp*)

Sets the current gamma ramp for the specified monitor.

This function sets the current gamma ramp for the specified monitor. The original gamma ramp for that monitor is saved by GLFW the first time this function is called and is restored by glfwTerminate.

Parameters

`in`	*monitor*	The monitor whose gamma ramp to set.
`in`	*ramp*	The gamma ramp to use.

Remarks

Gamma ramp sizes other than 256 are not supported by all platforms or graphics hardware.
Windows: The gamma ramp size must be 256.

Pointer Lifetime

The specified gamma ramp is copied before this function returns.

Thread Safety

This function may only be called from the main thread.

See also

monitor_gamma

Since

Added in GLFW 3.0.

5.4.3.11 GLFWAPI GLFWmonitorfun glfwSetMonitorCallback (GLFWmonitorfun *cbfun*)

Sets the monitor configuration callback.

This function sets the monitor configuration callback, or removes the currently set callback. This is called when a monitor is connected to or disconnected from the system.

Parameters

`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Bug **X11:** This callback is not yet called on monitor configuration changes.

Thread Safety

This function may only be called from the main thread.

See also

monitor_event

Since

Added in GLFW 3.0.

This function sets the monitor configuration callback, or removes the currently set callback. This is called when a monitor is connected to or disconnected from the system.

Parameters

`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Bug **X11:** This callback is not yet called on monitor configuration changes.

Thread Safety

This function may only be called from the main thread.

See also

monitor_event

Since

Added in GLFW 3.0.

5.5 Window handling

Typedefs

- typedef struct GLFWwindow GLFWwindow

  Opaque window object.
- typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

  The function signature for window position callbacks.
- typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

  The function signature for window resize callbacks.
- typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

  The function signature for window close callbacks.
- typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

  The function signature for window content refresh callbacks.
- typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

  The function signature for window focus/defocus callbacks.
- typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

  The function signature for window iconify/restore callbacks.
- typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

  The function signature for framebuffer resize callbacks.

- typedef struct GLFWwindow GLFWwindow

  Opaque window object.
- typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

  The function signature for window position callbacks.
- typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

  The function signature for window resize callbacks.
- typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

  The function signature for window close callbacks.
- typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

  The function signature for window content refresh callbacks.
- typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

  The function signature for window focus/defocus callbacks.
- typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

  The function signature for window iconify/restore callbacks.
- typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

  The function signature for framebuffer resize callbacks.

Functions

- GLFWAPI void glfwDefaultWindowHints (void)

  Resets all window hints to their default values.
- GLFWAPI void glfwWindowHint (int target, int hint)

  Sets the specified window hint to the desired value.
- GLFWAPI GLFWwindow ∗ glfwCreateWindow (int width, int height, const char ∗title, GLFWmonitor ∗monitor, GLFWwindow ∗share)

  Creates a window and its associated context.
- GLFWAPI void glfwDestroyWindow (GLFWwindow ∗window)

  Destroys the specified window and its context.
- GLFWAPI int glfwWindowShouldClose (GLFWwindow ∗window)

  Checks the close flag of the specified window.

- GLFWAPI void glfwSetWindowShouldClose (GLFWwindow ∗window, int value)

  Sets the close flag of the specified window.
- GLFWAPI void glfwSetWindowTitle (GLFWwindow ∗window, const char ∗title)

  Sets the title of the specified window.
- GLFWAPI void glfwGetWindowPos (GLFWwindow ∗window, int ∗xpos, int ∗ypos)

  Retrieves the position of the client area of the specified window.
- GLFWAPI void glfwSetWindowPos (GLFWwindow ∗window, int xpos, int ypos)

  Sets the position of the client area of the specified window.
- GLFWAPI void glfwGetWindowSize (GLFWwindow ∗window, int ∗width, int ∗height)

  Retrieves the size of the client area of the specified window.
- GLFWAPI void glfwSetWindowSize (GLFWwindow ∗window, int width, int height)

  Sets the size of the client area of the specified window.
- GLFWAPI void glfwGetFramebufferSize (GLFWwindow ∗window, int ∗width, int ∗height)

  Retrieves the size of the framebuffer of the specified window.
- GLFWAPI void glfwGetWindowFrameSize (GLFWwindow ∗window, int ∗left, int ∗top, int ∗right, int ∗bottom)

  Retrieves the size of the frame of the window.
- GLFWAPI void glfwIconifyWindow (GLFWwindow ∗window)

  Iconifies the specified window.
- GLFWAPI void glfwRestoreWindow (GLFWwindow ∗window)

  Restores the specified window.
- GLFWAPI void glfwShowWindow (GLFWwindow ∗window)

  Makes the specified window visible.
- GLFWAPI void glfwHideWindow (GLFWwindow ∗window)

  Hides the specified window.
- GLFWAPI GLFWmonitor ∗ glfwGetWindowMonitor (GLFWwindow ∗window)

  Returns the monitor that the window uses for full screen mode.
- GLFWAPI int glfwGetWindowAttrib (GLFWwindow ∗window, int attrib)

  Returns an attribute of the specified window.
- GLFWAPI void glfwSetWindowUserPointer (GLFWwindow ∗window, void ∗pointer)

  Sets the user pointer of the specified window.
- GLFWAPI void ∗ glfwGetWindowUserPointer (GLFWwindow ∗window)

  Returns the user pointer of the specified window.
- GLFWAPI GLFWwindowposfun glfwSetWindowPosCallback (GLFWwindow ∗window, GLFWwindowposfun cbfun)

  Sets the position callback for the specified window.
- GLFWAPI GLFWwindowsizefun glfwSetWindowSizeCallback (GLFWwindow ∗window, GLFWwindowsizefun cbfun)

  Sets the size callback for the specified window.
- GLFWAPI GLFWwindowclosefun glfwSetWindowCloseCallback (GLFWwindow ∗window, GLFWwindowclosefun cbfun)

  Sets the close callback for the specified window.
- GLFWAPI GLFWwindowrefreshfun glfwSetWindowRefreshCallback (GLFWwindow ∗window, GLF↩ Wwindowrefreshfun cbfun)

  Sets the refresh callback for the specified window.
- GLFWAPI GLFWwindowfocusfun glfwSetWindowFocusCallback (GLFWwindow ∗window, GLFWwindowfocusfun cbfun)

  Sets the focus callback for the specified window.
- GLFWAPI GLFWwindowiconifyfun glfwSetWindowIconifyCallback (GLFWwindow ∗window, GLF↩ Wwindowiconifyfun cbfun)

  Sets the iconify callback for the specified window.
- GLFWAPI GLFWframebuffersizefun glfwSetFramebufferSizeCallback (GLFWwindow ∗window, GLF↩ Wframebuffersizefun cbfun)

Sets the framebuffer resize callback for the specified window.
- GLFWAPI void glfwPollEvents (void)

  Processes all pending events.
- GLFWAPI void glfwWaitEvents (void)

  Waits until events are queued and processes them.
- GLFWAPI void glfwPostEmptyEvent (void)

  Posts an empty event to the event queue.
- GLFWAPI void glfwSwapBuffers (GLFWwindow ∗window)

  Swaps the front and back buffers of the specified window.

5.5.1 Detailed Description

This is the reference documentation for window related functions and types, including creation, deletion and event polling. For more information, see the Window handling.

5.5.2 Typedef Documentation

5.5.2.1 typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

The function signature for framebuffer resize callbacks.

This is the function signature for framebuffer resize callback functions.

Parameters

`in`	*window*	The window whose framebuffer was resized.
`in`	*width*	The new width, in pixels, of the framebuffer.
`in`	*height*	The new height, in pixels, of the framebuffer.

See also

glfwSetFramebufferSizeCallback

5.5.2.2 typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

The function signature for framebuffer resize callbacks.

This is the function signature for framebuffer resize callback functions.

Parameters

`in`	*window*	The window whose framebuffer was resized.
`in`	*width*	The new width, in pixels, of the framebuffer.
`in`	*height*	The new height, in pixels, of the framebuffer.

See also

glfwSetFramebufferSizeCallback

5.5.2.3 typedef struct GLFWwindow GLFWwindow

Opaque window object.

Opaque window object.

5.5.2.4 typedef struct GLFWwindow GLFWwindow

Opaque window object.

Opaque window object.

5.5.2.5 typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

The function signature for window close callbacks.

This is the function signature for window close callback functions.

Parameters

`in`	*window*	The window that the user attempted to close.

See also

glfwSetWindowCloseCallback

5.5.2.6 typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

The function signature for window close callbacks.

This is the function signature for window close callback functions.

Parameters

`in`	*window*	The window that the user attempted to close.

See also

glfwSetWindowCloseCallback

5.5.2.7 typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

The function signature for window focus/defocus callbacks.

This is the function signature for window focus callback functions.

Parameters

`in`	*window*	The window that gained or lost input focus.
`in`	*focused*	`GL_TRUE` if the window was given input focus, or `GL_FALSE` if it lost it.

See also

glfwSetWindowFocusCallback

5.5.2.8 typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

The function signature for window focus/defocus callbacks.

This is the function signature for window focus callback functions.

Parameters

`in`	*window*	The window that gained or lost input focus.
`in`	*focused*	`GL_TRUE` if the window was given input focus, or `GL_FALSE` if it lost it.

See also

glfwSetWindowFocusCallback

5.5.2.9 typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

The function signature for window iconify/restore callbacks.

This is the function signature for window iconify/restore callback functions.

Parameters

`in`	*window*	The window that was iconified or restored.
`in`	*iconified*	`GL_TRUE` if the window was iconified, or `GL_FALSE` if it was restored.

See also

glfwSetWindowIconifyCallback

5.5.2.10 typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

The function signature for window iconify/restore callbacks.

This is the function signature for window iconify/restore callback functions.

Parameters

`in`	*window*	The window that was iconified or restored.
`in`	*iconified*	`GL_TRUE` if the window was iconified, or `GL_FALSE` if it was restored.

See also

glfwSetWindowIconifyCallback

5.5.2.11 typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

The function signature for window position callbacks.

This is the function signature for window position callback functions.

Parameters

`in`	*window*	The window that was moved.
`in`	*xpos*	The new x-coordinate, in screen coordinates, of the upper-left corner of the client area of the window.
`in`	*ypos*	The new y-coordinate, in screen coordinates, of the upper-left corner of the client area of the window.

See also

glfwSetWindowPosCallback

5.5.2.12 typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

The function signature for window position callbacks.

This is the function signature for window position callback functions.

Parameters

`in`	*window*	The window that was moved.
`in`	*xpos*	The new x-coordinate, in screen coordinates, of the upper-left corner of the client area of the window.
`in`	*ypos*	The new y-coordinate, in screen coordinates, of the upper-left corner of the client area of the window.

See also

glfwSetWindowPosCallback

5.5.2.13 typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

The function signature for window content refresh callbacks.

This is the function signature for window refresh callback functions.

Parameters

`in`	*window*	The window whose content needs to be refreshed.

See also

glfwSetWindowRefreshCallback

5.5.2.14 typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

The function signature for window content refresh callbacks.

This is the function signature for window refresh callback functions.

Parameters

`in`	*window*	The window whose content needs to be refreshed.

See also

glfwSetWindowRefreshCallback

5.5.2.15 typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

The function signature for window resize callbacks.

This is the function signature for window size callback functions.

Parameters

`in`	*window*	The window that was resized.
`in`	*width*	The new width, in screen coordinates, of the window.
`in`	*height*	The new height, in screen coordinates, of the window.

See also

glfwSetWindowSizeCallback

5.5.2.16 typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

The function signature for window resize callbacks.

This is the function signature for window size callback functions.

Parameters

`in`	*window*	The window that was resized.
`in`	*width*	The new width, in screen coordinates, of the window.
`in`	*height*	The new height, in screen coordinates, of the window.

See also

glfwSetWindowSizeCallback

5.5.3 Function Documentation

5.5.3.1 GLFWAPI GLFWwindow ∗ glfwCreateWindow (int *width,* int *height,* const char ∗ *title,* GLFWmonitor ∗ *monitor,* GLFWwindow ∗ *share*)

Creates a window and its associated context.

This function creates a window and its associated OpenGL or OpenGL ES context. Most of the options controlling how the window and its context should be created are specified with window hints.

Successful creation does not change which context is current. Before you can use the newly created context, you need to make it current. For information about the `share` parameter, see context_sharing.

The created window, framebuffer and context may differ from what you requested, as not all parameters and hints are hard constraints. This includes the size of the window, especially for full screen windows. To query the actual attributes of the created window, framebuffer and context, see glfwGetWindowAttrib, glfwGetWindowSize and glfw←GetFramebufferSize.

To create a full screen window, you need to specify the monitor the window will cover. If no monitor is specified, windowed mode will be used. Unless you have a way for the user to choose a specific monitor, it is recommended that you pick the primary monitor. For more information on how to query connected monitors, see monitor_monitors.

For full screen windows, the specified size becomes the resolution of the window's *desired video mode*. As long as a full screen window has input focus, the supported video mode most closely matching the desired video mode is set for the specified monitor. For more information about full screen windows, including the creation of so called *windowed full screen* or *borderless full screen* windows, see window_windowed_full_screen.

By default, newly created windows use the placement recommended by the window system. To create the window at a specific position, make it initially invisible using the GLFW_VISIBLE window hint, set its position and then show it.

If a full screen window has input focus, the screensaver is prohibited from starting.

Window systems put limits on window sizes. Very large or very small window dimensions may be overridden by the window system on creation. Check the actual size after creation.

The swap interval is not set during window creation and the initial value may vary depending on driver settings and defaults.

Parameters

`in`	*width*	The desired width, in screen coordinates, of the window. This must be greater than zero.
`in`	*height*	The desired height, in screen coordinates, of the window. This must be greater than zero.
`in`	*title*	The initial, UTF-8 encoded window title.
`in`	*monitor*	The monitor to use for full screen mode, or `NULL` to use windowed mode.
`in`	*share*	The window whose context to share resources with, or `NULL` to not share resources.

Returns

The handle of the created window, or `NULL` if an error occurred.

Remarks

Windows: Window creation will fail if the Microsoft GDI software OpenGL implementation is the only one available.
Windows: If the executable has an icon resource named `GLFW_ICON`, it will be set as the icon for the window. If no such icon is present, the `IDI_WINLOGO` icon will be used instead.
Windows: The context to share resources with may not be current on any other thread.
OS X: The GLFW window has no icon, as it is not a document window, but the dock icon will be the same as the application bundle's icon. For more information on bundles, see the `Bundle Programming Guide` in the Mac Developer Library.
OS X: The first time a window is created the menu bar is populated with common commands like Hide, Quit and About. The About entry opens a minimal about dialog with information from the application's bundle. The menu bar can be disabled with a compile-time option.
OS X: On OS X 10.10 and later the window frame will not be rendered at full resolution on Retina displays unless the `NSHighResolutionCapable` key is enabled in the application bundle's `Info.plist`. For more information, see `High Resolution Guidelines for OS X` in the Mac Developer Library. The GLFW test and example programs use a custom `Info.plist` template for this, which can be found as `C↩ Make/MacOSXBundleInfo.plist.in` in the source tree.
X11: There is no mechanism for setting the window icon yet.
X11: Some window managers will not respect the placement of initially hidden windows.
X11: Due to the asynchronous nature of X11, it may take a moment for a window to reach its requested state. This means you may not be able to query the final size, position or other attributes directly after window creation.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

window_creation
glfwDestroyWindow

Since

Added in GLFW 3.0. Replaces `glfwOpenWindow`.

5.5.3.2 GLFWAPI void glfwDefaultWindowHints (void)

Resets all window hints to their default values.

This function resets all window hints to their default values.

Thread Safety

This function may only be called from the main thread.

See also

window_hints
glfwWindowHint

Since

Added in GLFW 3.0.

5.5.3.3 GLFWAPI void glfwDestroyWindow (GLFWwindow * *window*)

Destroys the specified window and its context.

This function destroys the specified window and its context. On calling this function, no further callbacks will be called for that window.

If the context of the specified window is current on the main thread, it is detached before being destroyed.

Parameters

`in`	*window*	The window to destroy.

Note

The context of the specified window must not be current on any other thread when this function is called.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

window_creation
glfwCreateWindow

Since

Added in GLFW 3.0. Replaces `glfwCloseWindow`.

5.5.3.4 GLFWAPI void glfwGetFramebufferSize (GLFWwindow ∗ *window,* int ∗ *width,* int ∗ *height*)

Retrieves the size of the framebuffer of the specified window.

This function retrieves the size, in pixels, of the framebuffer of the specified window. If you wish to retrieve the size of the window in screen coordinates, see glfwGetWindowSize.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

Parameters

`in`	*window*	The window whose framebuffer to query.
`out`	*width*	Where to store the width, in pixels, of the framebuffer, or `NULL`.
`out`	*height*	Where to store the height, in pixels, of the framebuffer, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

window_fbsize
glfwSetFramebufferSizeCallback

Since

Added in GLFW 3.0.

5.5.3.5 GLFWAPI int glfwGetWindowAttrib (GLFWwindow ∗ *window,* int *attrib*)

Returns an attribute of the specified window.

This function returns the value of an attribute of the specified window or its OpenGL or OpenGL ES context.

Parameters

`in`	*window*	The window to query.
`in`	*attrib*	The window attribute whose value to return.

Returns

The value of the attribute, or zero if an error occurred.

Remarks

Framebuffer related hints are not window attributes. See window_attribs_fb for more information.
Zero is a valid value for many window and context related attributes so you cannot use a return value of zero as an indication of errors. However, this function should not fail as long as it is passed valid arguments and the library has been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_attribs

Since

Added in GLFW 3.0. Replaces `glfwGetWindowParam` and `glfwGetGLVersion`.

5.5.3.6 GLFWAPI void glfwGetWindowFrameSize (GLFWwindow ∗ *window,* int ∗ *left,* int ∗ *top,* int ∗ *right,* int ∗ *bottom*)

Retrieves the size of the frame of the window.

This function retrieves the size, in screen coordinates, of each edge of the frame of the specified window. This size includes the title bar, if the window has one. The size of the frame may vary depending on the window-related hints used to create it.

Because this function retrieves the size of each window frame edge and not the offset along a particular coordinate axis, the retrieved values will always be zero or positive.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

Parameters

`in`	*window*	The window whose frame size to query.
`out`	*left*	Where to store the size, in screen coordinates, of the left edge of the window frame, or `NULL`.
`out`	*top*	Where to store the size, in screen coordinates, of the top edge of the window frame, or `NULL`.
`out`	*right*	Where to store the size, in screen coordinates, of the right edge of the window frame, or `NULL`.
`out`	*bottom*	Where to store the size, in screen coordinates, of the bottom edge of the window frame, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

window_size

Since

Added in GLFW 3.1.

5.5.3.7 GLFWAPI GLFWmonitor ∗ glfwGetWindowMonitor (GLFWwindow ∗ *window*)

Returns the monitor that the window uses for full screen mode.

This function returns the handle of the monitor that the specified window is in full screen on.

Parameters

`in`	*window*	The window to query.

Returns

The monitor, or `NULL` if the window is in windowed mode or an error occurred.

Thread Safety

This function may only be called from the main thread.

See also

window_monitor

Since

Added in GLFW 3.0.

5.5.3.8 GLFWAPI void glfwGetWindowPos (GLFWwindow ∗ *window,* int ∗ *xpos,* int ∗ *ypos*)

Retrieves the position of the client area of the specified window.

This function retrieves the position, in screen coordinates, of the upper-left corner of the client area of the specified window.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

Parameters

`in`	*window*	The window to query.
`out`	*xpos*	Where to store the x-coordinate of the upper-left corner of the client area, or `NULL`.
`out`	*ypos*	Where to store the y-coordinate of the upper-left corner of the client area, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

window_pos
glfwSetWindowPos

Since

Added in GLFW 3.0.

5.5.3.9 GLFWAPI void glfwGetWindowSize (GLFWwindow * *window,* int * *width,* int * *height*)

Retrieves the size of the client area of the specified window.

This function retrieves the size, in screen coordinates, of the client area of the specified window. If you wish to retrieve the size of the framebuffer of the window in pixels, see glfwGetFramebufferSize.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

Parameters

`in`	*window*	The window whose size to retrieve.
`out`	*width*	Where to store the width, in screen coordinates, of the client area, or `NULL`.
`out`	*height*	Where to store the height, in screen coordinates, of the client area, or `NULL`.

Thread Safety

This function may only be called from the main thread.

See also

window_size
glfwSetWindowSize

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.5.3.10 GLFWAPI void * glfwGetWindowUserPointer (GLFWwindow * *window*)

Returns the user pointer of the specified window.

This function returns the current value of the user-defined pointer of the specified window. The initial value is `NULL`.

Parameters

`in`	*window*	The window whose pointer to return.

Thread Safety

This function may be called from any thread. Access is not synchronized.

See also

window_userptr
glfwSetWindowUserPointer

Since

Added in GLFW 3.0.

5.5.3.11 GLFWAPI void glfwHideWindow (GLFWwindow ∗ *window*)

Hides the specified window.

This function hides the specified window if it was previously visible. If the window is already hidden or is in full screen mode, this function does nothing.

Parameters

`in`	*window*	The window to hide.

Thread Safety

This function may only be called from the main thread.

See also

window_hide
glfwShowWindow

Since

Added in GLFW 3.0.

5.5.3.12 GLFWAPI void glfwIconifyWindow (GLFWwindow ∗ *window*)

Iconifies the specified window.

This function iconifies (minimizes) the specified window if it was previously restored. If the window is already iconified, this function does nothing.

If the specified window is a full screen window, the original monitor resolution is restored until the window is restored.

Parameters

`in`	*window*	The window to iconify.

Thread Safety

This function may only be called from the main thread.

See also

window_iconify
glfwRestoreWindow

Since

Added in GLFW 2.1.

GLFW 3: Added window handle parameter.

5.5.3.13 GLFWAPI void glfwPollEvents (void)

Processes all pending events.

This function processes only those events that are already in the event queue and then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.

On some platforms, a window move, resize or menu operation will cause event processing to block. This is due to how event processing is designed on those platforms. You can use the window refresh callback to redraw the contents of your window when necessary during such operations.

On some platforms, certain events are sent directly to the application without going through the event queue, causing callbacks to be called outside of a call to one of the event processing functions.

Event processing is not required for joystick input to work.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

events
glfwWaitEvents

Since

Added in GLFW 1.0.

5.5.3.14 GLFWAPI void glfwPostEmptyEvent (void)

Posts an empty event to the event queue.

This function posts an empty event from the current thread to the event queue, causing glfwWaitEvents to return.

If no windows exist, this function returns immediately. For synchronization of threads in applications that do not create windows, use your threading library of choice.

Thread Safety

This function may be called from any thread.

See also

events
glfwWaitEvents

Since

Added in GLFW 3.1.

5.5.3.15 GLFWAPI void glfwRestoreWindow (GLFWwindow * *window*)

Restores the specified window.

This function restores the specified window if it was previously iconified (minimized). If the window is already restored, this function does nothing.

If the specified window is a full screen window, the resolution chosen for the window is restored on the selected monitor.

Parameters

`in`	*window*	The window to restore.

Thread Safety

This function may only be called from the main thread.

See also

window_iconify
glfwIconifyWindow

Since

Added in GLFW 2.1.

GLFW 3: Added window handle parameter.

5.5.3.16 GLFWAPI GLFWframebuffersizefun glfwSetFramebufferSizeCallback (GLFWwindow * *window*, GLFWframebuffersizefun *cbfun*)

Sets the framebuffer resize callback for the specified window.

This function sets the framebuffer resize callback of the specified window, which is called when the framebuffer of the specified window is resized.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_fbsize

Since

Added in GLFW 3.0.

5.5.3.17 GLFWAPI GLFWwindowclosefun glfwSetWindowCloseCallback (GLFWwindow * *window*, GLFWwindowclosefun *cbfun*)

Sets the close callback for the specified window.

This function sets the close callback of the specified window, which is called when the user attempts to close the window, for example by clicking the close widget in the title bar.

The close flag is set before this callback is called, but you can modify it at any time with glfwSetWindowShouldClose.

The close callback is not triggered by glfwDestroyWindow.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Remarks

OS X: Selecting Quit from the application menu will trigger the close callback for all windows.

Thread Safety

This function may only be called from the main thread.

See also

window_close

Since

Added in GLFW 2.5.

GLFW 3: Added window handle parameter. Updated callback signature.

5.5.3.18 GLFWAPI GLFWwindowfocusfun glfwSetWindowFocusCallback (GLFWwindow ∗ *window*, GLFWwindowfocusfun *cbfun*)

Sets the focus callback for the specified window.

This function sets the focus callback of the specified window, which is called when the window gains or loses input focus.

After the focus callback is called for a window that lost input focus, synthetic key and mouse button release events will be generated for all such that had been pressed. For more information, see glfwSetKeyCallback and glfwSet←MouseButtonCallback.

Parameters

in	*window*	The window whose callback to set.
in	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_focus

Since

Added in GLFW 3.0.

5.5.3.19 GLFWAPI GLFWwindowiconifyfun glfwSetWindowIconifyCallback (GLFWwindow ∗ *window,* GLFWwindowiconifyfun *cbfun*)

Sets the iconify callback for the specified window.

This function sets the iconification callback of the specified window, which is called when the window is iconified or restored.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_iconify

Since

Added in GLFW 3.0.

5.5.3.20 GLFWAPI void glfwSetWindowPos (GLFWwindow ∗ *window,* int *xpos,* int *ypos*)

Sets the position of the client area of the specified window.

This function sets the position, in screen coordinates, of the upper-left corner of the client area of the specified windowed mode window. If the window is a full screen window, this function does nothing.

Do not use this function to move an already visible window unless you have very good reasons for doing so, as it will confuse and annoy the user.

The window manager may put limits on what positions are allowed. GLFW cannot and should not override these limits.

Parameters

`in`	*window*	The window to query.
`in`	*xpos*	The x-coordinate of the upper-left corner of the client area.
`in`	*ypos*	The y-coordinate of the upper-left corner of the client area.

Thread Safety

This function may only be called from the main thread.

See also

window_pos
glfwGetWindowPos

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.5.3.21 GLFWAPI **GLFWwindowposfun** glfwSetWindowPosCallback (**GLFWwindow** $*$ *window*, **GLFWwindowposfun** *cbfun*)

Sets the position callback for the specified window.

This function sets the position callback of the specified window, which is called when the window is moved. The callback is provided with the screen position of the upper-left corner of the client area of the window.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_pos

Since

Added in GLFW 3.0.

5.5.3.22 GLFWAPI **GLFWwindowrefreshfun** glfwSetWindowRefreshCallback (**GLFWwindow** $*$ *window*, **GLFWwindowrefreshfun** *cbfun*)

Sets the refresh callback for the specified window.

This function sets the refresh callback of the specified window, which is called when the client area of the window needs to be redrawn, for example if the window has been exposed after having been covered by another window.

On compositing window systems such as Aero, Compiz or Aqua, where the window contents are saved off-screen, this callback may be called only very infrequently or never at all.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_refresh

Since

Added in GLFW 2.5.

GLFW 3: Added window handle parameter. Updated callback signature.

5.5.3.23 GLFWAPI void glfwSetWindowShouldClose (GLFWwindow ∗ *window,* int *value*)

Sets the close flag of the specified window.

This function sets the value of the close flag of the specified window. This can be used to override the user's attempt to close the window, or to signal that it should be closed.

Parameters

`in`	*window*	The window whose flag to change.
`in`	*value*	The new value.

Thread Safety

This function may be called from any thread. Access is not synchronized.

See also

window_close

Since

Added in GLFW 3.0.

5.5.3.24 GLFWAPI void glfwSetWindowSize (GLFWwindow ∗ *window,* int *width,* int *height*)

Sets the size of the client area of the specified window.

This function sets the size, in screen coordinates, of the client area of the specified window.

For full screen windows, this function selects and switches to the resolution closest to the specified size, without affecting the window's context. As the context is unaffected, the bit depths of the framebuffer remain unchanged.

The window manager may put limits on what sizes are allowed. GLFW cannot and should not override these limits.

Parameters

`in`	*window*	The window to resize.
`in`	*width*	The desired width of the specified window.
`in`	*height*	The desired height of the specified window.

Thread Safety

This function may only be called from the main thread.

See also

window_size
glfwGetWindowSize

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.5.3.25 GLFWAPI GLFWwindowsizefun glfwSetWindowSizeCallback (GLFWwindow ∗ *window,* GLFWwindowsizefun *cbfun*)

Sets the size callback for the specified window.

This function sets the size callback of the specified window, which is called when the window is resized. The callback is provided with the size, in screen coordinates, of the client area of the window.

Parameters

`in`	*window*	The window whose callback to set.
`in`	*cbfun*	The new callback, or `NULL` to remove the currently set callback.

Returns

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

Thread Safety

This function may only be called from the main thread.

See also

window_size

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter. Updated callback signature.

5.5.3.26 GLFWAPI void glfwSetWindowTitle (GLFWwindow * *window,* const char * *title*)

Sets the title of the specified window.

This function sets the window title, encoded as UTF-8, of the specified window.

Parameters

`in`	*window*	The window whose title to change.
`in`	*title*	The UTF-8 encoded window title.

Remarks

OS X: The window title will not be updated until the next time you process events.

Thread Safety

This function may only be called from the main thread.

See also

window_title

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.5.3.27 GLFWAPI void glfwSetWindowUserPointer (GLFWwindow * *window,* void * *pointer*)

Sets the user pointer of the specified window.

This function sets the user-defined pointer of the specified window. The current value is retained until the window is destroyed. The initial value is `NULL`.

Parameters

`in`	*window*	The window whose pointer to set.
`in`	*pointer*	The new value.

Thread Safety

This function may be called from any thread. Access is not synchronized.

See also

window_userptr
glfwGetWindowUserPointer

Since

Added in GLFW 3.0.

5.5.3.28 GLFWAPI void glfwShowWindow (GLFWwindow $*$ *window*)

Makes the specified window visible.

This function makes the specified window visible if it was previously hidden. If the window is already visible or is in full screen mode, this function does nothing.

Parameters

`in`	*window*	The window to make visible.

Thread Safety

This function may only be called from the main thread.

See also

window_hide
glfwHideWindow

Since

Added in GLFW 3.0.

5.5.3.29 GLFWAPI void glfwSwapBuffers (GLFWwindow $*$ *window*)

Swaps the front and back buffers of the specified window.

This function swaps the front and back buffers of the specified window. If the swap interval is greater than zero, the GPU driver waits the specified number of screen updates before swapping the buffers.

Parameters

`in`	*window*	The window whose buffers to swap.

Thread Safety

This function may be called from any thread.

See also

buffer_swap
glfwSwapInterval

Since

Added in GLFW 1.0.

GLFW 3: Added window handle parameter.

5.5.3.30 GLFWAPI void glfwWaitEvents (void)

Waits until events are queued and processes them.

This function puts the calling thread to sleep until at least one event is available in the event queue. Once one or more events are available, it behaves exactly like glfwPollEvents, i.e. the events in the queue are processed and the function then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.

Since not all events are associated with callbacks, this function may return without a callback having been called even if you are monitoring all callbacks.

On some platforms, a window move, resize or menu operation will cause event processing to block. This is due to how event processing is designed on those platforms. You can use the window refresh callback to redraw the contents of your window when necessary during such operations.

On some platforms, certain callbacks may be called outside of a call to one of the event processing functions.

If no windows exist, this function returns immediately. For synchronization of threads in applications that do not create windows, use your threading library of choice.

Event processing is not required for joystick input to work.

Reentrancy

This function may not be called from a callback.

Thread Safety

This function may only be called from the main thread.

See also

events
glfwPollEvents

Since

Added in GLFW 2.5.

5.5.3.31 GLFWAPI void glfwWindowHint (int *target*, int *hint*)

Sets the specified window hint to the desired value.

This function sets hints for the next call to glfwCreateWindow. The hints, once set, retain their values until changed by a call to glfwWindowHint or glfwDefaultWindowHints, or until the library is terminated.

Parameters

in	*target*	The window hint to set.
in	*hint*	The new value of the window hint.

Thread Safety

This function may only be called from the main thread.

See also

window_hints
glfwDefaultWindowHints

Since

Added in GLFW 3.0. Replaces `glfwOpenWindowHint`.

5.5.3.32 GLFWAPI int glfwWindowShouldClose (GLFWwindow ∗ *window*)

Checks the close flag of the specified window.

This function returns the value of the close flag of the specified window.

Parameters

in	*window*	The window to query.

Returns

The value of the close flag.

Thread Safety

This function may be called from any thread. Access is not synchronized.

See also

window_close

Since

Added in GLFW 3.0.

5.6 Keyboard keys

Macros

- #define **GLFW_KEY_UNKNOWN** -1
- #define **GLFW_KEY_SPACE** 32
- #define **GLFW_KEY_APOSTROPHE** 39 /∗ ' ∗/
- #define **GLFW_KEY_COMMA** 44 /∗ , ∗/
- #define **GLFW_KEY_MINUS** 45 /∗ - ∗/
- #define **GLFW_KEY_PERIOD** 46 /∗ . ∗/
- #define **GLFW_KEY_SLASH** 47 /∗ / ∗/
- #define **GLFW_KEY_0** 48
- #define **GLFW_KEY_1** 49
- #define **GLFW_KEY_2** 50
- #define **GLFW_KEY_3** 51
- #define **GLFW_KEY_4** 52
- #define **GLFW_KEY_5** 53
- #define **GLFW_KEY_6** 54
- #define **GLFW_KEY_7** 55
- #define **GLFW_KEY_8** 56
- #define **GLFW_KEY_9** 57
- #define **GLFW_KEY_SEMICOLON** 59 /∗ ; ∗/
- #define **GLFW_KEY_EQUAL** 61 /∗ = ∗/
- #define **GLFW_KEY_A** 65
- #define **GLFW_KEY_B** 66
- #define **GLFW_KEY_C** 67
- #define **GLFW_KEY_D** 68
- #define **GLFW_KEY_E** 69
- #define **GLFW_KEY_F** 70
- #define **GLFW_KEY_G** 71
- #define **GLFW_KEY_H** 72
- #define **GLFW_KEY_I** 73
- #define **GLFW_KEY_J** 74
- #define **GLFW_KEY_K** 75
- #define **GLFW_KEY_L** 76
- #define **GLFW_KEY_M** 77
- #define **GLFW_KEY_N** 78
- #define **GLFW_KEY_O** 79
- #define **GLFW_KEY_P** 80
- #define **GLFW_KEY_Q** 81
- #define **GLFW_KEY_R** 82
- #define **GLFW_KEY_S** 83
- #define **GLFW_KEY_T** 84
- #define **GLFW_KEY_U** 85
- #define **GLFW_KEY_V** 86
- #define **GLFW_KEY_W** 87
- #define **GLFW_KEY_X** 88
- #define **GLFW_KEY_Y** 89
- #define **GLFW_KEY_Z** 90
- #define **GLFW_KEY_LEFT_BRACKET** 91 /∗ [∗/
- #define **GLFW_KEY_BACKSLASH** 92 /∗ \ ∗/
- #define **GLFW_KEY_RIGHT_BRACKET** 93 /∗] ∗/
- #define **GLFW_KEY_GRAVE_ACCENT** 96 /∗ ‘ ∗/
- #define **GLFW_KEY_WORLD_1** 161 /∗ non-US #1 ∗/

- #define **GLFW_KEY_WORLD_2** 162 /∗ non-US #2 ∗/
- #define **GLFW_KEY_ESCAPE** 256
- #define **GLFW_KEY_ENTER** 257
- #define **GLFW_KEY_TAB** 258
- #define **GLFW_KEY_BACKSPACE** 259
- #define **GLFW_KEY_INSERT** 260
- #define **GLFW_KEY_DELETE** 261
- #define **GLFW_KEY_RIGHT** 262
- #define **GLFW_KEY_LEFT** 263
- #define **GLFW_KEY_DOWN** 264
- #define **GLFW_KEY_UP** 265
- #define **GLFW_KEY_PAGE_UP** 266
- #define **GLFW_KEY_PAGE_DOWN** 267
- #define **GLFW_KEY_HOME** 268
- #define **GLFW_KEY_END** 269
- #define **GLFW_KEY_CAPS_LOCK** 280
- #define **GLFW_KEY_SCROLL_LOCK** 281
- #define **GLFW_KEY_NUM_LOCK** 282
- #define **GLFW_KEY_PRINT_SCREEN** 283
- #define **GLFW_KEY_PAUSE** 284
- #define **GLFW_KEY_F1** 290
- #define **GLFW_KEY_F2** 291
- #define **GLFW_KEY_F3** 292
- #define **GLFW_KEY_F4** 293
- #define **GLFW_KEY_F5** 294
- #define **GLFW_KEY_F6** 295
- #define **GLFW_KEY_F7** 296
- #define **GLFW_KEY_F8** 297
- #define **GLFW_KEY_F9** 298
- #define **GLFW_KEY_F10** 299
- #define **GLFW_KEY_F11** 300
- #define **GLFW_KEY_F12** 301
- #define **GLFW_KEY_F13** 302
- #define **GLFW_KEY_F14** 303
- #define **GLFW_KEY_F15** 304
- #define **GLFW_KEY_F16** 305
- #define **GLFW_KEY_F17** 306
- #define **GLFW_KEY_F18** 307
- #define **GLFW_KEY_F19** 308
- #define **GLFW_KEY_F20** 309
- #define **GLFW_KEY_F21** 310
- #define **GLFW_KEY_F22** 311
- #define **GLFW_KEY_F23** 312
- #define **GLFW_KEY_F24** 313
- #define **GLFW_KEY_F25** 314
- #define **GLFW_KEY_KP_0** 320
- #define **GLFW_KEY_KP_1** 321
- #define **GLFW_KEY_KP_2** 322
- #define **GLFW_KEY_KP_3** 323
- #define **GLFW_KEY_KP_4** 324
- #define **GLFW_KEY_KP_5** 325
- #define **GLFW_KEY_KP_6** 326
- #define **GLFW_KEY_KP_7** 327
- #define **GLFW_KEY_KP_8** 328
- #define **GLFW_KEY_KP_9** 329

- #define **GLFW_KEY_KP_DECIMAL** 330
- #define **GLFW_KEY_KP_DIVIDE** 331
- #define **GLFW_KEY_KP_MULTIPLY** 332
- #define **GLFW_KEY_KP_SUBTRACT** 333
- #define **GLFW_KEY_KP_ADD** 334
- #define **GLFW_KEY_KP_ENTER** 335
- #define **GLFW_KEY_KP_EQUAL** 336
- #define **GLFW_KEY_LEFT_SHIFT** 340
- #define **GLFW_KEY_LEFT_CONTROL** 341
- #define **GLFW_KEY_LEFT_ALT** 342
- #define **GLFW_KEY_LEFT_SUPER** 343
- #define **GLFW_KEY_RIGHT_SHIFT** 344
- #define **GLFW_KEY_RIGHT_CONTROL** 345
- #define **GLFW_KEY_RIGHT_ALT** 346
- #define **GLFW_KEY_RIGHT_SUPER** 347
- #define **GLFW_KEY_MENU** 348
- #define **GLFW_KEY_LAST** GLFW_KEY_MENU
- #define **GLFW_KEY_UNKNOWN** -1
- #define **GLFW_KEY_SPACE** 32
- #define **GLFW_KEY_APOSTROPHE** 39 /∗ ' ∗/
- #define **GLFW_KEY_COMMA** 44 /∗ , ∗/
- #define **GLFW_KEY_MINUS** 45 /∗ - ∗/
- #define **GLFW_KEY_PERIOD** 46 /∗ . ∗/
- #define **GLFW_KEY_SLASH** 47 /∗ / ∗/
- #define **GLFW_KEY_0** 48
- #define **GLFW_KEY_1** 49
- #define **GLFW_KEY_2** 50
- #define **GLFW_KEY_3** 51
- #define **GLFW_KEY_4** 52
- #define **GLFW_KEY_5** 53
- #define **GLFW_KEY_6** 54
- #define **GLFW_KEY_7** 55
- #define **GLFW_KEY_8** 56
- #define **GLFW_KEY_9** 57
- #define **GLFW_KEY_SEMICOLON** 59 /∗ ; ∗/
- #define **GLFW_KEY_EQUAL** 61 /∗ = ∗/
- #define **GLFW_KEY_A** 65
- #define **GLFW_KEY_B** 66
- #define **GLFW_KEY_C** 67
- #define **GLFW_KEY_D** 68
- #define **GLFW_KEY_E** 69
- #define **GLFW_KEY_F** 70
- #define **GLFW_KEY_G** 71
- #define **GLFW_KEY_H** 72
- #define **GLFW_KEY_I** 73
- #define **GLFW_KEY_J** 74
- #define **GLFW_KEY_K** 75
- #define **GLFW_KEY_L** 76
- #define **GLFW_KEY_M** 77
- #define **GLFW_KEY_N** 78
- #define **GLFW_KEY_O** 79
- #define **GLFW_KEY_P** 80
- #define **GLFW_KEY_Q** 81
- #define **GLFW_KEY_R** 82
- #define **GLFW_KEY_S** 83

- #define **GLFW_KEY_T** 84
- #define **GLFW_KEY_U** 85
- #define **GLFW_KEY_V** 86
- #define **GLFW_KEY_W** 87
- #define **GLFW_KEY_X** 88
- #define **GLFW_KEY_Y** 89
- #define **GLFW_KEY_Z** 90
- #define **GLFW_KEY_LEFT_BRACKET** 91 /∗ [∗/
- #define **GLFW_KEY_BACKSLASH** 92 /∗ \ ∗/
- #define **GLFW_KEY_RIGHT_BRACKET** 93 /∗] ∗/
- #define **GLFW_KEY_GRAVE_ACCENT** 96 /∗ ‘ ∗/
- #define **GLFW_KEY_WORLD_1** 161 /∗ non-US #1 ∗/
- #define **GLFW_KEY_WORLD_2** 162 /∗ non-US #2 ∗/
- #define **GLFW_KEY_ESCAPE** 256
- #define **GLFW_KEY_ENTER** 257
- #define **GLFW_KEY_TAB** 258
- #define **GLFW_KEY_BACKSPACE** 259
- #define **GLFW_KEY_INSERT** 260
- #define **GLFW_KEY_DELETE** 261
- #define **GLFW_KEY_RIGHT** 262
- #define **GLFW_KEY_LEFT** 263
- #define **GLFW_KEY_DOWN** 264
- #define **GLFW_KEY_UP** 265
- #define **GLFW_KEY_PAGE_UP** 266
- #define **GLFW_KEY_PAGE_DOWN** 267
- #define **GLFW_KEY_HOME** 268
- #define **GLFW_KEY_END** 269
- #define **GLFW_KEY_CAPS_LOCK** 280
- #define **GLFW_KEY_SCROLL_LOCK** 281
- #define **GLFW_KEY_NUM_LOCK** 282
- #define **GLFW_KEY_PRINT_SCREEN** 283
- #define **GLFW_KEY_PAUSE** 284
- #define **GLFW_KEY_F1** 290
- #define **GLFW_KEY_F2** 291
- #define **GLFW_KEY_F3** 292
- #define **GLFW_KEY_F4** 293
- #define **GLFW_KEY_F5** 294
- #define **GLFW_KEY_F6** 295
- #define **GLFW_KEY_F7** 296
- #define **GLFW_KEY_F8** 297
- #define **GLFW_KEY_F9** 298
- #define **GLFW_KEY_F10** 299
- #define **GLFW_KEY_F11** 300
- #define **GLFW_KEY_F12** 301
- #define **GLFW_KEY_F13** 302
- #define **GLFW_KEY_F14** 303
- #define **GLFW_KEY_F15** 304
- #define **GLFW_KEY_F16** 305
- #define **GLFW_KEY_F17** 306
- #define **GLFW_KEY_F18** 307
- #define **GLFW_KEY_F19** 308
- #define **GLFW_KEY_F20** 309
- #define **GLFW_KEY_F21** 310
- #define **GLFW_KEY_F22** 311
- #define **GLFW_KEY_F23** 312

- #define **GLFW_KEY_F24** 313
- #define **GLFW_KEY_F25** 314
- #define **GLFW_KEY_KP_0** 320
- #define **GLFW_KEY_KP_1** 321
- #define **GLFW_KEY_KP_2** 322
- #define **GLFW_KEY_KP_3** 323
- #define **GLFW_KEY_KP_4** 324
- #define **GLFW_KEY_KP_5** 325
- #define **GLFW_KEY_KP_6** 326
- #define **GLFW_KEY_KP_7** 327
- #define **GLFW_KEY_KP_8** 328
- #define **GLFW_KEY_KP_9** 329
- #define **GLFW_KEY_KP_DECIMAL** 330
- #define **GLFW_KEY_KP_DIVIDE** 331
- #define **GLFW_KEY_KP_MULTIPLY** 332
- #define **GLFW_KEY_KP_SUBTRACT** 333
- #define **GLFW_KEY_KP_ADD** 334
- #define **GLFW_KEY_KP_ENTER** 335
- #define **GLFW_KEY_KP_EQUAL** 336
- #define **GLFW_KEY_LEFT_SHIFT** 340
- #define **GLFW_KEY_LEFT_CONTROL** 341
- #define **GLFW_KEY_LEFT_ALT** 342
- #define **GLFW_KEY_LEFT_SUPER** 343
- #define **GLFW_KEY_RIGHT_SHIFT** 344
- #define **GLFW_KEY_RIGHT_CONTROL** 345
- #define **GLFW_KEY_RIGHT_ALT** 346
- #define **GLFW_KEY_RIGHT_SUPER** 347
- #define **GLFW_KEY_MENU** 348
- #define **GLFW_KEY_LAST** GLFW_KEY_MENU

5.6.1 Detailed Description

See key input for how these are used.

These key codes are inspired by the *USB HID Usage Tables v1.12* (p. 53-60), but re-arranged to map to 7-bit ASCII for printable keys (function keys are put in the 256+ range).

The naming of the key codes follow these rules:

- The US keyboard layout is used
- Names of printable alpha-numeric characters are used (e.g. "A", "R", "3", etc.)
- For non-alphanumeric characters, Unicode:ish names are used (e.g. "COMMA", "LEFT_SQUARE_BRAC↩ KET", etc.). Note that some names do not correspond to the Unicode standard (usually for brevity)
- Keys that lack a clear US mapping are named "WORLD_x"
- For non-printable keys, custom names are used (e.g. "F4", "BACKSPACE", etc.)

5.7 Modifier key flags

Macros

- #define GLFW_MOD_SHIFT 0x0001

  If this bit is set one or more Shift keys were held down.
- #define GLFW_MOD_CONTROL 0x0002

  If this bit is set one or more Control keys were held down.
- #define GLFW_MOD_ALT 0x0004

  If this bit is set one or more Alt keys were held down.
- #define GLFW_MOD_SUPER 0x0008

  If this bit is set one or more Super keys were held down.
- #define GLFW_MOD_SHIFT 0x0001

  If this bit is set one or more Shift keys were held down.
- #define GLFW_MOD_CONTROL 0x0002

  If this bit is set one or more Control keys were held down.
- #define GLFW_MOD_ALT 0x0004

  If this bit is set one or more Alt keys were held down.
- #define GLFW_MOD_SUPER 0x0008

  If this bit is set one or more Super keys were held down.

5.7.1 Detailed Description

See key input for how these are used.

5.8 Mouse buttons

Macros

- #define **GLFW_MOUSE_BUTTON_1** 0
- #define **GLFW_MOUSE_BUTTON_2** 1
- #define **GLFW_MOUSE_BUTTON_3** 2
- #define **GLFW_MOUSE_BUTTON_4** 3
- #define **GLFW_MOUSE_BUTTON_5** 4
- #define **GLFW_MOUSE_BUTTON_6** 5
- #define **GLFW_MOUSE_BUTTON_7** 6
- #define **GLFW_MOUSE_BUTTON_8** 7
- #define **GLFW_MOUSE_BUTTON_LAST** GLFW_MOUSE_BUTTON_8
- #define **GLFW_MOUSE_BUTTON_LEFT** GLFW_MOUSE_BUTTON_1
- #define **GLFW_MOUSE_BUTTON_RIGHT** GLFW_MOUSE_BUTTON_2
- #define **GLFW_MOUSE_BUTTON_MIDDLE** GLFW_MOUSE_BUTTON_3
- #define **GLFW_MOUSE_BUTTON_1** 0
- #define **GLFW_MOUSE_BUTTON_2** 1
- #define **GLFW_MOUSE_BUTTON_3** 2
- #define **GLFW_MOUSE_BUTTON_4** 3
- #define **GLFW_MOUSE_BUTTON_5** 4
- #define **GLFW_MOUSE_BUTTON_6** 5
- #define **GLFW_MOUSE_BUTTON_7** 6
- #define **GLFW_MOUSE_BUTTON_8** 7
- #define **GLFW_MOUSE_BUTTON_LAST** GLFW_MOUSE_BUTTON_8
- #define **GLFW_MOUSE_BUTTON_LEFT** GLFW_MOUSE_BUTTON_1
- #define **GLFW_MOUSE_BUTTON_RIGHT** GLFW_MOUSE_BUTTON_2
- #define **GLFW_MOUSE_BUTTON_MIDDLE** GLFW_MOUSE_BUTTON_3

5.8.1 Detailed Description

See mouse button input for how these are used.

5.9 Joysticks

Macros

- #define **GLFW_JOYSTICK_1** 0
- #define **GLFW_JOYSTICK_2** 1
- #define **GLFW_JOYSTICK_3** 2
- #define **GLFW_JOYSTICK_4** 3
- #define **GLFW_JOYSTICK_5** 4
- #define **GLFW_JOYSTICK_6** 5
- #define **GLFW_JOYSTICK_7** 6
- #define **GLFW_JOYSTICK_8** 7
- #define **GLFW_JOYSTICK_9** 8
- #define **GLFW_JOYSTICK_10** 9
- #define **GLFW_JOYSTICK_11** 10
- #define **GLFW_JOYSTICK_12** 11
- #define **GLFW_JOYSTICK_13** 12
- #define **GLFW_JOYSTICK_14** 13
- #define **GLFW_JOYSTICK_15** 14
- #define **GLFW_JOYSTICK_16** 15
- #define **GLFW_JOYSTICK_LAST** GLFW_JOYSTICK_16

- #define **GLFW_JOYSTICK_1** 0
- #define **GLFW_JOYSTICK_2** 1
- #define **GLFW_JOYSTICK_3** 2
- #define **GLFW_JOYSTICK_4** 3
- #define **GLFW_JOYSTICK_5** 4
- #define **GLFW_JOYSTICK_6** 5
- #define **GLFW_JOYSTICK_7** 6
- #define **GLFW_JOYSTICK_8** 7
- #define **GLFW_JOYSTICK_9** 8
- #define **GLFW_JOYSTICK_10** 9
- #define **GLFW_JOYSTICK_11** 10
- #define **GLFW_JOYSTICK_12** 11
- #define **GLFW_JOYSTICK_13** 12
- #define **GLFW_JOYSTICK_14** 13
- #define **GLFW_JOYSTICK_15** 14
- #define **GLFW_JOYSTICK_16** 15
- #define **GLFW_JOYSTICK_LAST** GLFW_JOYSTICK_16

5.9.1 Detailed Description

See joystick input for how these are used.

5.10 Error codes

Macros

- #define GLFW_NOT_INITIALIZED 0x00010001

  GLFW has not been initialized.
- #define GLFW_NO_CURRENT_CONTEXT 0x00010002

  No context is current for this thread.
- #define GLFW_INVALID_ENUM 0x00010003

  One of the arguments to the function was an invalid enum value.
- #define GLFW_INVALID_VALUE 0x00010004

  One of the arguments to the function was an invalid value.
- #define GLFW_OUT_OF_MEMORY 0x00010005

  A memory allocation failed.
- #define GLFW_API_UNAVAILABLE 0x00010006

  GLFW could not find support for the requested client API on the system.
- #define GLFW_VERSION_UNAVAILABLE 0x00010007

  The requested OpenGL or OpenGL ES version is not available.
- #define GLFW_PLATFORM_ERROR 0x00010008

  A platform-specific error occurred that does not match any of the more specific categories.
- #define GLFW_FORMAT_UNAVAILABLE 0x00010009

  The requested format is not supported or available.

- #define GLFW_NOT_INITIALIZED 0x00010001

  GLFW has not been initialized.
- #define GLFW_NO_CURRENT_CONTEXT 0x00010002

  No context is current for this thread.
- #define GLFW_INVALID_ENUM 0x00010003

  One of the arguments to the function was an invalid enum value.
- #define GLFW_INVALID_VALUE 0x00010004

  One of the arguments to the function was an invalid value.
- #define GLFW_OUT_OF_MEMORY 0x00010005

  A memory allocation failed.
- #define GLFW_API_UNAVAILABLE 0x00010006

  GLFW could not find support for the requested client API on the system.
- #define GLFW_VERSION_UNAVAILABLE 0x00010007

  The requested OpenGL or OpenGL ES version is not available.
- #define GLFW_PLATFORM_ERROR 0x00010008

  A platform-specific error occurred that does not match any of the more specific categories.
- #define GLFW_FORMAT_UNAVAILABLE 0x00010009

  The requested format is not supported or available.

5.10.1 Detailed Description

See error handling for how these are used.

5.10.2 Macro Definition Documentation

5.10.2.1 #define GLFW_API_UNAVAILABLE 0x00010006

GLFW could not find support for the requested client API on the system.

GLFW could not find support for the requested client API on the system. If emitted by functions other than glfw↩ CreateWindow, no supported client API was found.

Analysis

The installed graphics driver does not support the requested client API, or does not support it via the chosen context creation backend. Below are a few examples.

Some pre-installed Windows graphics drivers do not support OpenGL. AMD only supports OpenGL ES via EGL, while Nvidia and Intel only support it via a WGL or GLX extension. OS X does not provide OpenGL ES at all. The Mesa EGL, OpenGL and OpenGL ES libraries do not interface with the Nvidia binary driver.

5.10.2.2 #define GLFW_API_UNAVAILABLE 0x00010006

GLFW could not find support for the requested client API on the system.

GLFW could not find support for the requested client API on the system. If emitted by functions other than glfw↩ CreateWindow, no supported client API was found.

Analysis

The installed graphics driver does not support the requested client API, or does not support it via the chosen context creation backend. Below are a few examples.

Some pre-installed Windows graphics drivers do not support OpenGL. AMD only supports OpenGL ES via EGL, while Nvidia and Intel only support it via a WGL or GLX extension. OS X does not provide OpenGL ES at all. The Mesa EGL, OpenGL and OpenGL ES libraries do not interface with the Nvidia binary driver.

5.10.2.3 #define GLFW_FORMAT_UNAVAILABLE 0x00010009

The requested format is not supported or available.

If emitted during window creation, the requested pixel format is not supported.

If emitted when querying the clipboard, the contents of the clipboard could not be converted to the requested format.

Analysis

If emitted during window creation, one or more hard constraints did not match any of the available pixel formats. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

If emitted when querying the clipboard, ignore the error or report it to the user, as appropriate.

5.10.2.4 #define GLFW_FORMAT_UNAVAILABLE 0x00010009

The requested format is not supported or available.

If emitted during window creation, the requested pixel format is not supported.

If emitted when querying the clipboard, the contents of the clipboard could not be converted to the requested format.

Analysis

If emitted during window creation, one or more hard constraints did not match any of the available pixel formats. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

If emitted when querying the clipboard, ignore the error or report it to the user, as appropriate.

5.10.2.5 #define GLFW_INVALID_ENUM 0x00010003

One of the arguments to the function was an invalid enum value.

One of the arguments to the function was an invalid enum value, for example requesting GLFW_RED_BITS with glfwGetWindowAttrib.

Analysis

Application programmer error. Fix the offending call.

5.10.2.6 #define GLFW_INVALID_ENUM 0x00010003

One of the arguments to the function was an invalid enum value.

One of the arguments to the function was an invalid enum value, for example requesting GLFW_RED_BITS with glfwGetWindowAttrib.

Analysis

Application programmer error. Fix the offending call.

5.10.2.7 #define GLFW_INVALID_VALUE 0x00010004

One of the arguments to the function was an invalid value.

One of the arguments to the function was an invalid value, for example requesting a non-existent OpenGL or OpenGL ES version like 2.7.

Requesting a valid but unavailable OpenGL or OpenGL ES version will instead result in a GLFW_VERSION_UN↩
AVAILABLE error.

Analysis

Application programmer error. Fix the offending call.

5.10.2.8 #define GLFW_INVALID_VALUE 0x00010004

One of the arguments to the function was an invalid value.

One of the arguments to the function was an invalid value, for example requesting a non-existent OpenGL or OpenGL ES version like 2.7.

Requesting a valid but unavailable OpenGL or OpenGL ES version will instead result in a GLFW_VERSION_UN←AVAILABLE error.

Analysis

Application programmer error. Fix the offending call.

5.10.2.9 #define GLFW_NO_CURRENT_CONTEXT 0x00010002

No context is current for this thread.

This occurs if a GLFW function was called that needs and operates on the current OpenGL or OpenGL ES context but no context is current on the calling thread. One such function is glfwSwapInterval.

Analysis

Application programmer error. Ensure a context is current before calling functions that require a current context.

5.10.2.10 #define GLFW_NO_CURRENT_CONTEXT 0x00010002

No context is current for this thread.

This occurs if a GLFW function was called that needs and operates on the current OpenGL or OpenGL ES context but no context is current on the calling thread. One such function is glfwSwapInterval.

Analysis

Application programmer error. Ensure a context is current before calling functions that require a current context.

5.10.2.11 #define GLFW_NOT_INITIALIZED 0x00010001

GLFW has not been initialized.

This occurs if a GLFW function was called that may not be called unless the library is initialized.

Analysis

Application programmer error. Initialize GLFW before calling any function that requires initialization.

5.10.2.12 #define GLFW_NOT_INITIALIZED 0x00010001

GLFW has not been initialized.

This occurs if a GLFW function was called that may not be called unless the library is initialized.

Analysis

Application programmer error. Initialize GLFW before calling any function that requires initialization.

5.10.2.13 #define GLFW_OUT_OF_MEMORY 0x00010005

A memory allocation failed.

A memory allocation failed.

Analysis

A bug in GLFW or the underlying operating system. Report the bug to our `issue tracker`.

5.10.2.14 #define GLFW_OUT_OF_MEMORY 0x00010005

A memory allocation failed.

A memory allocation failed.

Analysis

A bug in GLFW or the underlying operating system. Report the bug to our `issue tracker`.

5.10.2.15 #define GLFW_PLATFORM_ERROR 0x00010008

A platform-specific error occurred that does not match any of the more specific categories.

A platform-specific error occurred that does not match any of the more specific categories.

Analysis

A bug or configuration error in GLFW, the underlying operating system or its drivers, or a lack of required resources. Report the issue to our `issue tracker`.

5.10.2.16 #define GLFW_PLATFORM_ERROR 0x00010008

A platform-specific error occurred that does not match any of the more specific categories.

A platform-specific error occurred that does not match any of the more specific categories.

Analysis

A bug or configuration error in GLFW, the underlying operating system or its drivers, or a lack of required resources. Report the issue to our `issue tracker`.

5.10.2.17 #define GLFW_VERSION_UNAVAILABLE 0x00010007

The requested OpenGL or OpenGL ES version is not available.

The requested OpenGL or OpenGL ES version (including any requested context or framebuffer hints) is not available on this machine.

Analysis

The machine does not support your requirements. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

Future invalid OpenGL and OpenGL ES versions, for example OpenGL 4.8 if 5.0 comes out before the 4.↩
x series gets that far, also fail with this error and not GLFW_INVALID_VALUE, because GLFW cannot know what future versions will exist.

5.10.2.18 #define GLFW_VERSION_UNAVAILABLE 0x00010007

The requested OpenGL or OpenGL ES version is not available.

The requested OpenGL or OpenGL ES version (including any requested context or framebuffer hints) is not available on this machine.

Analysis

The machine does not support your requirements. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

Future invalid OpenGL and OpenGL ES versions, for example OpenGL 4.8 if 5.0 comes out before the 4.↩
x series gets that far, also fail with this error and not GLFW_INVALID_VALUE, because GLFW cannot know what future versions will exist.

5.11 Standard cursor shapes

Macros

- #define GLFW_ARROW_CURSOR 0x00036001

  The regular arrow cursor shape.
- #define GLFW_IBEAM_CURSOR 0x00036002

  The text input I-beam cursor shape.
- #define GLFW_CROSSHAIR_CURSOR 0x00036003

  The crosshair shape.
- #define GLFW_HAND_CURSOR 0x00036004

  The hand shape.
- #define GLFW_HRESIZE_CURSOR 0x00036005

  The horizontal resize arrow shape.
- #define GLFW_VRESIZE_CURSOR 0x00036006

  The vertical resize arrow shape.
- #define GLFW_ARROW_CURSOR 0x00036001

  The regular arrow cursor shape.
- #define GLFW_IBEAM_CURSOR 0x00036002

  The text input I-beam cursor shape.
- #define GLFW_CROSSHAIR_CURSOR 0x00036003

  The crosshair shape.
- #define GLFW_HAND_CURSOR 0x00036004

  The hand shape.
- #define GLFW_HRESIZE_CURSOR 0x00036005

  The horizontal resize arrow shape.
- #define GLFW_VRESIZE_CURSOR 0x00036006

  The vertical resize arrow shape.

5.11.1 Detailed Description

See standard cursor creation for how these are used.

5.11.2 Macro Definition Documentation

5.11.2.1 #define GLFW_ARROW_CURSOR 0x00036001

The regular arrow cursor shape.

The regular arrow cursor.

5.11.2.2 #define GLFW_ARROW_CURSOR 0x00036001

The regular arrow cursor shape.

The regular arrow cursor.

5.11.2.3 #define GLFW_CROSSHAIR_CURSOR 0x00036003

The crosshair shape.

The crosshair shape.

5.11.2.4 #define GLFW_CROSSHAIR_CURSOR 0x00036003

The crosshair shape.

The crosshair shape.

5.11.2.5 #define GLFW_HAND_CURSOR 0x00036004

The hand shape.

The hand shape.

5.11.2.6 #define GLFW_HAND_CURSOR 0x00036004

The hand shape.

The hand shape.

5.11.2.7 #define GLFW_HRESIZE_CURSOR 0x00036005

The horizontal resize arrow shape.

The horizontal resize arrow shape.

5.11.2.8 #define GLFW_HRESIZE_CURSOR 0x00036005

The horizontal resize arrow shape.

The horizontal resize arrow shape.

5.11.2.9 #define GLFW_IBEAM_CURSOR 0x00036002

The text input I-beam cursor shape.

The text input I-beam cursor shape.

5.11.2.10 #define GLFW_IBEAM_CURSOR 0x00036002

The text input I-beam cursor shape.

The text input I-beam cursor shape.

5.11.2.11 #define GLFW_VRESIZE_CURSOR 0x00036006

The vertical resize arrow shape.

The vertical resize arrow shape.

5.11.2.12 #define GLFW_VRESIZE_CURSOR 0x00036006

The vertical resize arrow shape.

The vertical resize arrow shape.

5.12 Native access

By using the native access functions you assert that you know what you're doing and how to fix problems caused by using them. If you don't, you shouldn't be using them.

Before the inclusion of glfw3native.h, you must define exactly one window system API macro and exactly one context creation API macro. Failure to do this will cause a compile-time error.

The available window API macros are:

- `GLFW_EXPOSE_NATIVE_WIN32`
- `GLFW_EXPOSE_NATIVE_COCOA`
- `GLFW_EXPOSE_NATIVE_X11`

The available context API macros are:

- `GLFW_EXPOSE_NATIVE_WGL`
- `GLFW_EXPOSE_NATIVE_NSGL`
- `GLFW_EXPOSE_NATIVE_GLX`
- `GLFW_EXPOSE_NATIVE_EGL`

These macros select which of the native access functions that are declared and which platform-specific headers to include. It is then up your (by definition platform-specific) code to handle which of these should be defined.

Chapter 6

Class Documentation

6.1 _GPU_DEVICE Struct Reference

Public Attributes

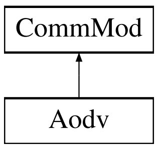
- DWORD **cb**
- CHAR **DeviceName** [32]
- CHAR **DeviceString** [128]
- DWORD **Flags**
- RECT **rcVirtualScreen**

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

- code/liboctodrone/include/GL/wglew.h

6.2 Aodv Class Reference

Inheritance diagram for Aodv:



Public Member Functions

- Aodv (Environment ∗, std::string, std::atomic_flag ∗, bool)

  Implementation of the AODV routing protocol.

Protected Member Functions

- void comm_function ()

  The main communications loop which handles incomming and outgoing messages.

Private Member Functions

- Aodv_rreq ∗ create_hello ()
- Aodv_rreq ∗ create_rreq (std::string, std::string, int)

  Helper method for creating route requests.
- Aodv_rrep ∗ create_rrep (std::string, std::string, int)
- Aodv_rerr ∗ create_rerr (std::string, int)

  Helper method for creating route errors.
- void process_rreq (Aodv_rreq ∗)

  Handle an RREQ that was received.
- void process_rrep (Aodv_rrep ∗)

  Handle an RREP that was received.
- void process_rerr (Aodv_rerr ∗)

  Handle an RERR that was received.
- void process_data (std::string)

  Handle an data packet that was received.
- std::string get_attribute (std::string)

  Attribute extraction for AODV messages.
- bool have_route (std::string)

  Checks if a route to the destination is known.
- void add_route (std::string, int, int, std::string)

  Creates a route in the routing table.
- Aodv_rreq ∗ deserialize_rreq (std::string)

  Deserialization for route requests.
- Aodv_rrep ∗ deserialize_rrep (std::string)

  Deserialization for route replies.
- Aodv_rerr ∗ deserialize_rerr (std::string)

  Deserialization for route errors.
- void log (std::string)

  Helper function to log internal information.
- void broadcast (std::string)

  Helper function to broadcast a message through the environment.

Private Attributes

- std::map< std::string, Aodv_route ∗ > route_table

  Routing table for the communication module.
- std::string ip_address

  The IP address of the communication module.
- double HELLO_INTERVAL

  The interval at which hello messages are sent to discover nearby nodes.
- int SEQUENCE_NUMBER

  The AODV sequence number of the communication module.
- double ACTIVE_ROUTE_TIMEOUT

  The AODV active route timeout used to determine how long routes stay fresh for.
- double PATH_DISCOVERY_TIME

  How long to wait for relies to route requests.
- int BROADCAST_ID

  AODV broadcast ID used to prevent loops and ensure fresh information.
- int RANGE

The amount of power used to broadcast messages.

- int TTL

  Default time to live for messages, dependent on network size.
- double last_hello

  The time at which the last hello message was sent.
- std::pair< std::string, std::string > **current_message**
- int state

  The internal state of the AODV implementation.
- std::atomic_flag ∗ lock

  An atomic lock to regulate access to stdout.
- bool logging

  Switch to enable or disable logging.

Additional Inherited Members

6.2.1 Member Function Documentation

6.2.1.1 void Aodv::add_route (std::string *ip*, int *dest_seq*, int *hop_count*, std::string *next_hop*) `[private]`

Creates a route in the routing table.

Adds in the standard route timeout and ensures that all variables are set to prevent memory issues later

6.2.1.2 void Aodv::broadcast (std::string *message*) `[private]`

Helper function to broadcast a message through the environment.

Broadcast requires the coordinates of the broadcasting unit which can be unwieldy to do every time This process is simplified by wrapping it in a helper function

6.2.1.3 void Aodv::comm_function ( ) `[protected]`, `[virtual]`

The main communications loop which handles incomming and outgoing messages.

Every loop we check to see if we should send a hello, based on the hello interval Next, any incomming messages are deserialized and handled appropriately If our messageable told us to shut down, the communications module exits Next, any outgoing messages are either immediately sent (if we have a route to the destination already) or a route request is distributed (if we do not have a route)

Implements CommMod.

6.2.1.4 **Aodv_rreq** ∗ Aodv::create_hello ( ) `[private]`

Creates a special route request for a route to this node, with TTL 1

6.2.1.5 **Aodv_rerr** ∗ Aodv::create_rerr (std::string *dst_ip,* int *ttl*) [private]

Helper method for creating route errors.

Abstracts away the destination sequence number lookup for convienence

6.2.1.6 **Aodv_rrep** ∗ Aodv::create_rrep (std::string *dst_ip,* std::string *src_ip,* int *ttl*) [private]

Abstracts away next hop lookup for convienence

6.2.1.7 **Aodv_rreq** ∗ Aodv::create_rreq (std::string *dst_ip,* std::string *src_ip,* int *ttl*) [private]

Helper method for creating route requests.

Abstracts away the destination sequence number lookup for convienence

6.2.1.8 **Aodv_rerr** ∗ Aodv::deserialize_rerr (std::string *message*) [private]

Deserialization for route errors.

Takes a raw message string of a route error message and returns an AODV object representing that message

6.2.1.9 **Aodv_rrep** ∗ Aodv::deserialize_rrep (std::string *message*) [private]

Deserialization for route replies.

Takes a raw message string of a route reply message and returns an AODV object representing that message

6.2.1.10 **Aodv_rreq** ∗ Aodv::deserialize_rreq (std::string *message*) [private]

Deserialization for route requests.

Takes a raw message string of a route reuest message and returns an AODV object representing that message

6.2.1.11 std::string Aodv::get_attribute (std::string *message*) [private]

Attribute extraction for AODV messages.

Takes in a message string and returns the first field of that message (semicolon delimited)

6.2.1.12 bool Aodv::have_route (std::string *ip*) [private]

Checks if a route to the destination is known.

Determines if a route to the destination is known, and if that route is fresh

6.2.1.13 void Aodv::log (std::string *log_message*) [private]

Helper function to log internal information.

Makes use of the stdout lock we were passed on creation to make sure that only one thread is printing at any one time Also prints the ip of the communications module and the current environment time to help with debugging

6.2.1.14 void Aodv::process_data (std::string *message*) [private]

Handle an data packet that was received.

First we check if the message was for us, and if so we deliver the contents to our messageable If the message is destined for another node and we are specified as the next hop, then we forward that packet to the next hop according to our own routing table Note that messages destined for other nodes that we are not specified as a next hop for will be dropped

6.2.1.15 void Aodv::process_rerr (Aodv_rerr * *message*) [private]

Handle an RERR that was received.

This functionality os not yet implemented

6.2.1.16 void Aodv::process_rrep (Aodv_rrep * *message*) [private]

Handle an RREP that was received.

First we check if the message contains new information about other nodes, and if so we always update our routing table If the message was a response to a request we initiated, then we should send our data packet. If the message was a response to a request we forwarded for another node, then we should pass it on Note that nodes will only act as a middle hop for one message at a time (other reponses will be dropped)

6.2.1.17 void Aodv::process_rreq (Aodv_rreq * *message*) [private]

Handle an RREQ that was received.

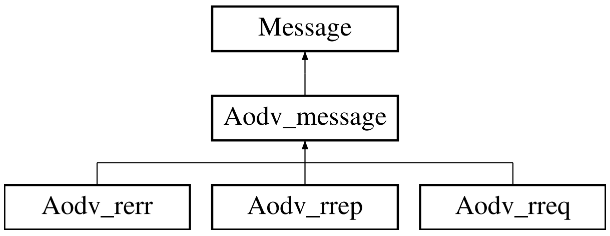
First we check if the message was a hello message, and if so we always respond to it If the message was a normal request, then we either answer it (if we have the info) or forward it if we do not Note that non-hello RREQs will be dropped if we are currently trying to send a message

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

- code/liboctodronecomms/include/Aodv.hpp
- code/liboctodronecomms/src/Aodv.cpp

6.3 Aodv_message Class Reference

Inheritance diagram for Aodv_message:



Public Member Functions

- Aodv_message (std::string, int, int)

  Base class for all AODV message types.
- std::string get_dest_ip ()

  Getter method for the IP address of the destination node.
- std::string serialize ()

  Returns a string representation of the message.
- int get_dest_seq ()

  Getter method for the sequence number of the destination node.
- int get_ttl ()

  Getter method for the time to live.

Private Attributes

- std::string dest_ip

  The IP address of the detination node.
- int dest_seq

  The sequence number of the destination node.
- int ttl

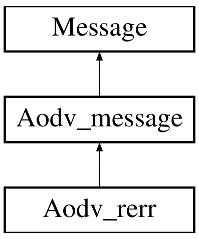
  The time to live of the message.

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

- code/liboctodronecomms/include/Aodv_message.hpp
- code/liboctodronecomms/src/Aodv_message.cpp

6.4 Aodv_rerr Class Reference

Inheritance diagram for Aodv_rerr:



Public Member Functions

- Aodv_rerr (std::string dst_ip, int dst_seq, int ttl)

  AODV route error message.
- std::string to_string ()

  Returns a string representation of the message.

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

- code/liboctodronecomms/include/Aodv_rerr.hpp
- code/liboctodronecomms/src/Aodv_rerr.cpp

6.5 Aodv_route Class Reference

Public Member Functions

- Aodv_route (int, int, std::string, int)

  Entry in ann AODV routing table.
- int get_seq ()

  Getter method for the sequence number of the route destination.
- int get_hop ()

  Getter method for the route hop count.
- std::string get_next_hop ()

  Getter method for the next hop on this route.
- int get_life ()

  Getter method for the route life time.

Private Attributes

- int dst_seq

  Sequence number of the route destination.
- int hop_count

  Number of hops required to get to the destination node.
- std::string next_hop

  Next hop on this route.
- int life_time

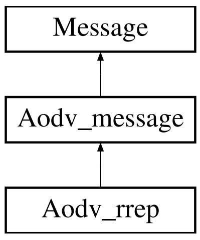
  Lifetime of this route.

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

- code/liboctodronecomms/include/Aodv_route.hpp
- code/liboctodronecomms/src/Aodv_route.cpp

6.6 Aodv_rrep Class Reference

Inheritance diagram for Aodv_rrep:



Public Member Functions

- Aodv_rrep (int, std::string, std::string, std::string, int, int, int, std::string)

  Aodv route reply message.
- int get_hop_count ()

  Getter method for the hop count of the message.
- std::string get_source_ip ()

  Getter method for the IP address of the sending node.
- std::string to_string ()

  Returns a string representation of the message.
- int get_life_time ()

  Getter method for the life time of the route.
- std::string get_last_hop ()

  Getter method for te last hop on this route.
- std::string get_next_hop ()

  Getter method for the next hop on this route.

Private Attributes

- int m_hop_count

  Number of hops required to get from the source to the destination of the route.
- std::string m_source_ip

  IP address of the node at which the route terminates.
- int m_life_time

  Time for which this route is valid.
- std::string m_last_hop

  The last node which forwarded this reply.
- std::string m_next_hop

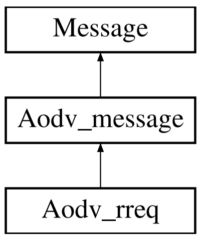
  The next node to which this reply will be forwarded.

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

- code/liboctodronecomms/include/Aodv_rrep.hpp
- code/liboctodronecomms/src/Aodv_rrep.cpp

6.7 Aodv_rreq Class Reference

Inheritance diagram for Aodv_rreq:



Public Member Functions

- Aodv_rreq (int hop, std::string src_ip, std::string dst_ip, int src_seq, int dst_seq, int ttl)

    AODV route request message.
- int get_hop_count ()

    Getter method for the hop count of the message.
- std::string get_source_ip ()

    Getter method for the IP address of the sending node.
- std::string to_string ()

    Returns a string representation of the message.
- int get_source_seq ()

    Getter method for the sequence number of the sending node.

Private Attributes

- int hop_count

    The number of hops taken towards the destination.
- std::string source_ip

    The IP address of the sending node.
- int source_seq

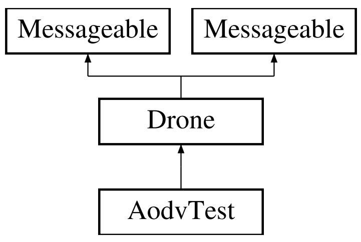
    The sequence number of the sending node.

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

- code/liboctodronecomms/include/Aodv_rreq.hpp
- code/liboctodronecomms/src/Aodv_rreq.cpp

6.8 AodvTest Class Reference

Inheritance diagram for AodvTest:



Public Member Functions

- **AodvTest** (CommMod ∗, double, double, double, double, Environment ∗, int, int ∗, std::atomic_flag ∗)
- bool **message_callback** (Message ∗)
- void **run** ()

Private Attributes

- int **m_task**
- int ∗ **m_flag**
- std::atomic_flag ∗ **m_lock**
- Environment ∗ **m_env**

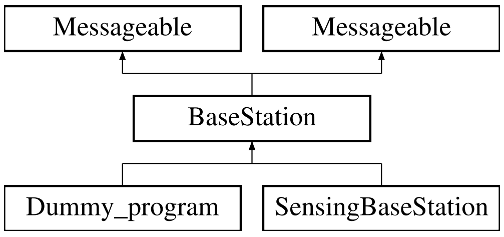
Additional Inherited Members

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

- code/programs/include/AodvTest.hpp
- code/programs/src/AodvTest.cpp

6.9 BaseStation Class Reference

Inheritance diagram for BaseStation:



Public Member Functions

- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp, bool vs)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp, bool vs)

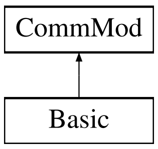
Additional Inherited Members

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

- code/liboctodrone/include/BaseStation.hpp
- code/liboctodrone/src/BaseStation.cpp

6.10 Basic Class Reference

Inheritance diagram for Basic:



Public Member Functions

- Basic (Environment ∗, std::atomic_flag ∗)

  Basic messaging protocol for testing where nodes receive all messages sent if they are within range.

Protected Member Functions

- void comm_function ()

  The main communications loop which handles incomming and outgoing messages.

Private Member Functions

- void log (std::string)

  Helper function to log internal information.

Private Attributes

- double RANGE

  The amount of power used to broadcast messages.
- std::atomic_flag ∗ lock

  An atomic lock to regulate access to stdout.

Additional Inherited Members

6.10.1 Member Function Documentation

6.10.1.1 void Basic::comm_function ( ) `[protected]`, `[virtual]`

The main communications loop which handles incomming and outgoing messages.

Every loop we send any messages that are waiting to be sent Then we deliver any messages that we have received

Implements CommMod.

6.10.1.2 void Basic::log (std::string *log_message*) [private]

Helper function to log internal information.

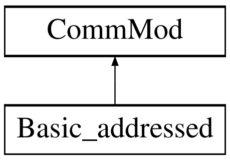
Makes use of the stdout lock we were passed on creation to make sure that only one thread is printing at any one time

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

- code/liboctodronecomms/include/Basic.hpp
- code/liboctodronecomms/src/Basic.cpp

6.11 Basic_addressed Class Reference

Inheritance diagram for Basic_addressed:



Public Member Functions

- Basic_addressed (Environment ∗, std::atomic_flag ∗, std::string ip)

  Basic messaging protocol with addressing for testing, where nodes receive all messages sent if they are within range and addressed to them.

Protected Member Functions

- void comm_function ()

  The main communications loop which handles incomming and outgoing messages.

Private Member Functions

- void **log** (std::string)
- std::string **get_attribute** (std::string)

Private Attributes

- double RANGE

  The amount of power used to boradcast messages.
- std::atomic_flag ∗ lock

  An atomic lock to regulate access to stdout.
- std::string ip_address

  The IP address of the communication module.

Additional Inherited Members

6.11.1 Member Function Documentation

6.11.1.1 void Basic_addressed::comm_function ( ) [protected], [virtual]

The main communications loop which handles incomming and outgoing messages.

Every loop any incomming messages are deserialized and delivered if they match our IP address Next, any outgoing messages are sent Note that received messages not addressed to this IP address will be dropped

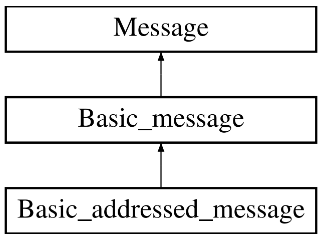
Implements CommMod.

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

- code/liboctodronecomms/include/Basic_addressed.hpp
- code/liboctodronecomms/src/Basic_addressed.cpp

6.12 Basic_addressed_message Class Reference

Inheritance diagram for Basic_addressed_message:



Public Member Functions

- Basic_addressed_message (std::string, std::string, std::string)

    Basic message with addressing information.
- std::string to_string ()

    Returns a string representation of the message.
- std::string get_message ()

    Getter method for the message content.
- std::string get_destination ()

    Getter method for the IP address of the node this message is destined for.
- std::string get_source ()

    Getter method for the IP address of the node which sent this message.

Private Attributes

- std::string message

  The contents of the message.
- std::string destination

  The IP address of the messages destination.
- std::string source

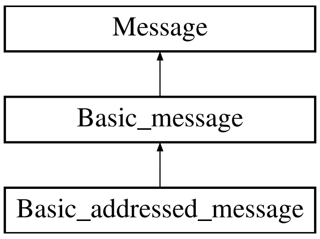
  The IP address of the node which sent the message.

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

- code/liboctodronecomms/include/Basic_addressed_message.hpp
- code/liboctodronecomms/src/Basic_addressed_message.cpp

6.13 Basic_message Class Reference

Inheritance diagram for Basic_message:



Public Member Functions

- Basic_message (std::string)

  A basic message containing a payload wth no addressing information.
- std::string to_string ()

  Returns a string representation of the message.

Private Attributes

- std::string message

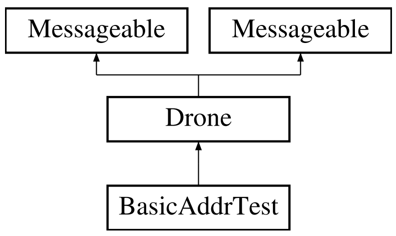
  The contents of the message.

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

- code/liboctodronecomms/include/Basic_message.hpp
- code/liboctodronecomms/src/Basic_message.cpp

6.14 BasicAddrTest Class Reference

Inheritance diagram for BasicAddrTest:



Public Member Functions

- **BasicAddrTest** (CommMod ∗, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

Private Attributes

- bool **sink_node**

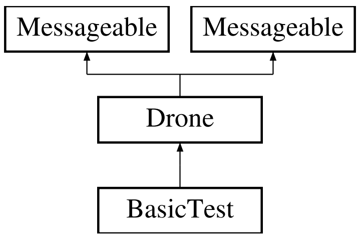
Additional Inherited Members

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

- code/programs/include/BasicAddrTest.hpp
- code/programs/src/BasicAddrTest.cpp

6.15 BasicTest Class Reference

Inheritance diagram for BasicTest:



Public Member Functions

- **BasicTest** (CommMod ∗, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

Private Attributes

- bool **sink_node**

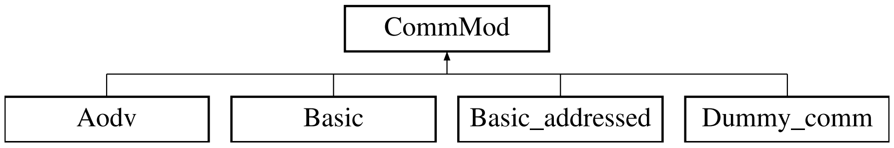
Additional Inherited Members

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

- code/programs/include/BasicTest.hpp
- code/programs/src/BasicTest.cpp

6.16 CommMod Class Reference

Inheritance diagram for CommMod:



Public Member Functions

- CommMod (Environment ∗env)

  Base class for communication algorithm implementations.
- void set_messageable (Messageable ∗msg)

  Set the messageble associated with this communications module.
- void broadcast (std::string message, double xPos, double yPos, double zPos, double range)

  Passes a message to the environment for transmission.
- void broadcast (Message ∗message, double xPos, double yPos, double zPos, double range)

  Passes a message to the environment for transmission.
- void push_out_message (Message ∗message)

  Called by a messagable to send a message.
- void push_in_message (std::string message)

  Called by a messagable to receive a message.
- void **pass_message** (Message ∗message)
- void **comm_function_wrapper** ()
- virtual void comm_function ()=0

  Main loop which must be defined by communications implementations.
- bool **getAlive** ()
- double getTime ()

  Get the time from the environment.
- **CommMod** (Environment ∗env)
- void **set_messageable** (Messageable ∗msg)
- void **broadcast** (std::string message, double xPos, double yPos, double zPos, double range)
- void **broadcast** (Message ∗message, double xPos, double yPos, double zPos, double range)
- void **push_out_message** (Message ∗message)
- void **push_in_message** (std::string message)
- void **pass_message** (Message ∗message)
- void **comm_function_wrapper** ()
- virtual void comm_function ()=0

  Main loop which must be defined by communications implementations.
- bool **getAlive** ()
- double **getTime** ()

Protected Attributes

- std::queue< Message ∗ > outQueue

    the queue of messages to be sent
- std::queue< std::string > inQueue

    the queue of received messages waiting for collection by the messageable
- Environment ∗ environment

    Reference to the simulation environment.
- Messageable ∗ messageable

    Reference to the associated messageable.

Private Attributes

- bool **alive**

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

- code/liboctodrone/include/CommMod.hpp
- code/liboctodrone/src/CommMod.cpp

6.17 Coord Struct Reference

Public Attributes

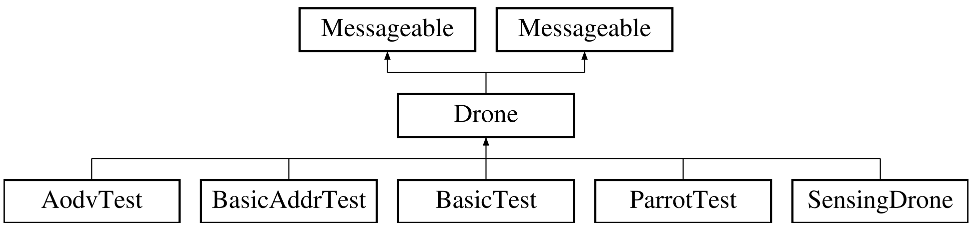
- double **x**
- double **y**
- double **z**

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

- code/liboctodrone/include/Messageable.hpp

6.18 Drone Class Reference

Inheritance diagram for Drone:



Public Member Functions

- Drone (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e)

  Modified to start flight when the program runs.
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e, bool vis)
- bool **isAlive** ()
- void upkeep (bool)

  Modified to make the drone hover when it has finished a movement instruction.
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e)
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e, bool vis)
- bool **isAlive** ()
- void **upkeep** (bool)
- void execute (std::string, double)

  Takes a command and sends it to the nodeServer which is connected to the drone.
- void **execute** (std::string)

Protected Member Functions

- void kill ()

  Modified to end flight when the program terminates.
- void turn (double dAngle)

  Modified to create estimates based on real time and to send instructions to nodeServer.
- void move (Direction direction, double speed, double distance)

  Modified to calculate duration based on real time and to send instructions to nodeServer.
- double **getMaxSpeed** ()
- double **getSpeed** ()
- double **getAngle** ()
- bool hasFinishedMoving ()

  Modified to evaluate estimated move finishing times.
- double **sense** (std::string type)
- void **kill** ()
- void **turn** (double dAngle)
- void **move** (Direction direction, double speed, double distance)
- double **getMaxSpeed** ()
- double **getAngle** ()
- bool **hasFinishedMoving** ()
- double **sense** (std::string type)

Private Attributes

- double **oTime**
- bool **alive** = true
- Direction **dir**
- double **maxSpeed**
- double **ang** = 0
- Environment ∗ **env**
- double **moveDR**
- double **moveSpd**
- bool **visualise**

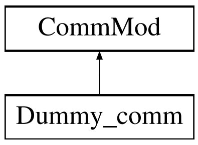
Additional Inherited Members

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

- code/liboctodrone/include/Drone.hpp
- code/liboctodrone/src/Drone.cpp

6.19 Dummy_comm Class Reference

Inheritance diagram for Dummy_comm:



Public Member Functions

- **Dummy_comm** (Environment ∗)

Protected Member Functions

- void comm_function ()
  Main loop which must be defined by communications implementations.

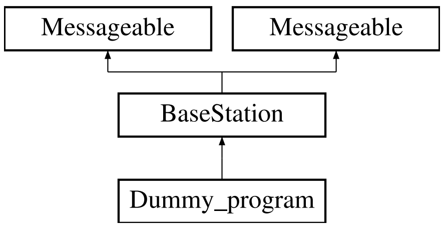
Additional Inherited Members

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

- code/liboctodronecomms/include/Dummy_comm.hpp
- code/liboctodronecomms/src/Dummy_comm.cpp

6.20 Dummy_program Class Reference

Inheritance diagram for Dummy_program:



Public Member Functions

- **Dummy_program** (CommMod ∗, double, double, double)
- bool **message_callback** (Message ∗)
- void **run** ()

Additional Inherited Members

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

- code/programs/include/Dummy_program.hpp
- code/programs/src/Dummy_program.cpp

6.21 Element Class Reference

Public Member Functions

- **Element** (IMG img, int x, int y, int size, int steps)
- void **draw** ()
- bool **step** ()

Private Attributes

- int **stepsLeft**
- int **x**
- int **y**
- int **size**
- IMG **image**

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

- code/liboctodrone/include/Visualisation.hpp
- code/liboctodrone/src/Visualisation.cpp

6.22 Environment Class Reference

Public Types

- typedef std::vector< std::vector< std::vector< double > > > **data_type**

Public Member Functions

- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, double timestep)
- **Environment** (std::map< std::string, data_type >, double timestep)
- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, double timestep, bool visualise)
- **Environment** (std::map< std::string, data_type >, double timestep, bool visualise)
- void **broadcast** (std::string message, double xOrigin, double yOrigin, double zOrigin, double range, Comm↩ Mod ∗)
- void **addData** (std::string type, data_type d)
- void **addDrone** (Drone ∗m)
- void **setBaseStation** (BaseStation ∗m)
- double **getData** (std::string type, double x, double y, double z)
- double getTime ()

    Modified to use real time.
- void **run** ()
- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, std::string)
- **Environment** (std::map< std::string, data_type >, std::string)
- void **broadcast** (std::string message, double xOrigin, double yOrigin, double zOrigin, double range, Comm↩ Mod ∗)
- void **addData** (std::string type, data_type d)
- void **addDrone** (Drone ∗m)
- void **setBaseStation** (BaseStation ∗m)
- double **getData** (std::string type, double x, double y, double z)
- Drone ∗ getDrone ()

    Returns the only drone in the sharded environment.
- double **getTime** ()
- void **run** ()

Private Types

- typedef std::vector< std::vector< std::vector< double > > > **data_type**

Private Attributes

- double **timeElapsed**
- double **timeStep**
- bool **visualise**
- BaseStation ∗ **baseStation**
- std::vector< Drone ∗ > **drones**
- std::map< std::string, data_type > **data**
- std::function< std::string(std::string)> **noiseFun**
- std::string **if_addr**

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

- code/liboctodrone/include/Environment.hpp
- code/liboctodrone/src/Environment.cpp

6.23 GLFWgammaramp Struct Reference

Gamma ramp.

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

Public Attributes

- unsigned short ∗ red
- unsigned short ∗ green
- unsigned short ∗ blue
- unsigned int size

6.23.1 Detailed Description

Gamma ramp.

This describes the gamma ramp for a monitor.

See also

glfwGetGammaRamp glfwSetGammaRamp

6.23.2 Member Data Documentation

6.23.2.1 unsigned short ∗ GLFWgammaramp::blue

An array of value describing the response of the blue channel.

6.23.2.2 unsigned short ∗ GLFWgammaramp::green

An array of value describing the response of the green channel.

6.23.2.3 unsigned short ∗ GLFWgammaramp::red

An array of value describing the response of the red channel.

6.23.2.4 unsigned int GLFWgammaramp::size

The number of elements in each array.

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

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

6.24 GLFWimage Struct Reference

Image data.

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

Public Attributes

- int width
- int height
- unsigned char ∗ pixels

6.24.1 Detailed Description

Image data.

6.24.2 Member Data Documentation

6.24.2.1 int GLFWimage::height

The height, in pixels, of this image.

6.24.2.2 unsigned char ∗ GLFWimage::pixels

The pixel data of this image, arranged left-to-right, top-to-bottom.

6.24.2.3 int GLFWimage::width

The width, in pixels, of this image.

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

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

6.25 GLFWvidmode Struct Reference

Video mode type.

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

Public Attributes

- int width
- int height
- int redBits
- int greenBits
- int blueBits
- int refreshRate

6.25.1 Detailed Description

Video mode type.

This describes a single video mode.

6.25.2 Member Data Documentation

6.25.2.1 int GLFWvidmode::blueBits

The bit depth of the blue channel of the video mode.

6.25.2.2 int GLFWvidmode::greenBits

The bit depth of the green channel of the video mode.

6.25.2.3 int GLFWvidmode::height

The height, in screen coordinates, of the video mode.

6.25.2.4 int GLFWvidmode::redBits

The bit depth of the red channel of the video mode.

6.25.2.5 int GLFWvidmode::refreshRate

The refresh rate, in Hz, of the video mode.

6.25.2.6 int GLFWvidmode::width

The width, in screen coordinates, of the video mode.

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

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

6.26 IpAllocator Class Reference

Public Member Functions

- **IpAllocator** (int, int, int, int)
- std::string **next** ()
- **IpAllocator** (int, int, int, int)
- std::string **next** ()

Private Attributes

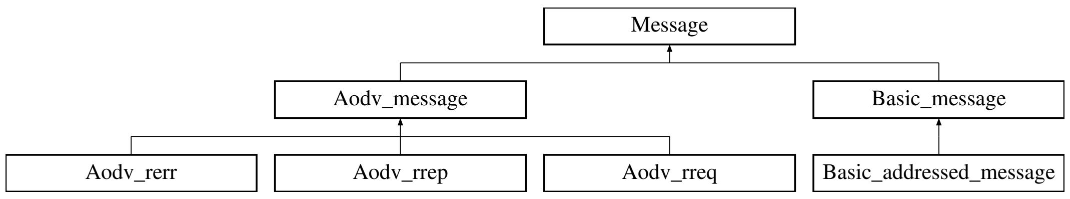
- int **m_first**
- int **m_second**
- int **m_third**
- int **m_fourth**

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

- code/liboctodrone/include/IpAllocator.hpp
- code/liboctodrone/src/IpAllocator.cpp

6.27 Message Class Reference

Inheritance diagram for Message:



Public Member Functions

- **Message** (std::string type)
- time_t **get_current_time** ()
- virtual std::string **to_string** ()=0
- std::string **get_type** ()
- **Message** (std::string type)
- time_t **get_current_time** ()
- virtual std::string **to_string** ()=0
- std::string **get_type** ()

Private Attributes

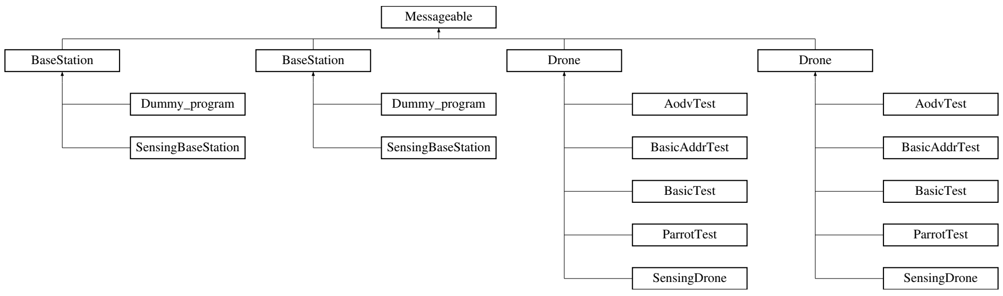
- std::string **type**

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

- code/liboctodrone/include/Message.hpp
- code/liboctodrone/src/Message.cpp

6.28 Messageable Class Reference

Inheritance diagram for Messageable:



Public Member Functions

- **Messageable** (CommMod ∗cm, double xp, double yp, double zp)
- void **send_message** (Message ∗contents)
- Message ∗ **wait_for_message** ()
- void **push_message** (Message ∗contents)
- void **receive_message** (std::string contents)
- CommMod ∗ **get_comm_mod** ()
- double **getX** ()
- double **getY** ()
- double **getZ** ()
- Coord **getPosition** ()
- double **getTime** ()
- virtual bool **message_callback** (Message ∗message)=0
- bool **getAlive** ()
- void **run_wrapper** ()
- virtual void **run** ()=0
- void **runCommMod** ()
- **Messageable** (CommMod ∗cm, double xp, double yp, double zp)
- void **send_message** (Message ∗contents)
- Message ∗ **wait_for_message** ()
- void **push_message** (Message ∗contents)
- void **receive_message** (std::string contents)
- CommMod ∗ **get_comm_mod** ()
- double **getX** ()
- double **getY** ()
- double **getZ** ()
- Coord **getPosition** ()
- double **getTime** ()
- virtual bool **message_callback** (Message ∗message)=0
- bool **getAlive** ()
- void **run_wrapper** ()
- virtual void **run** ()=0
- void **runCommMod** ()

Protected Attributes

- std::queue< Message ∗ > **inQueue**
- CommMod ∗ **communicationsModule**
- Coord **position**

Private Attributes

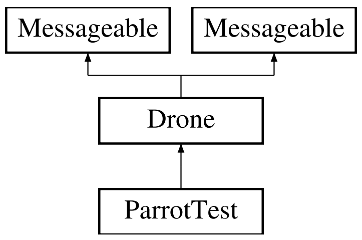
- bool **alive**

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

- code/liboctodrone/include/Messageable.hpp
- code/liboctodrone/src/Messageable.cpp

6.29 ParrotTest Class Reference

Inheritance diagram for ParrotTest:



Public Member Functions

- **ParrotTest** (CommMod $*$, double, double, double, double, Environment $*$, bool)
- bool **message_callback** (Message $*$)
- void **run** ()

Private Attributes

- bool **m_sink_node**

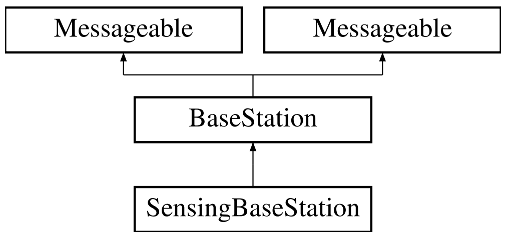
Additional Inherited Members

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

- code/programs/include/ParrotTest.hpp
- code/programs/src/ParrotTest.cpp

6.30 SensingBaseStation Class Reference

Inheritance diagram for SensingBaseStation:



Public Member Functions

- **SensingBaseStation** (CommMod ∗cm, double xp, double yp, double zp, double areaX1, double areaY1, double areaX2, double areaY2)
- void **run** ()
- bool **message_callback** (Message ∗message)

Private Member Functions

- void **interpretMessage** (Message ∗message)

Private Attributes

- std::vector< std::string > **droneIPs**
- double **areaX1**
- double **areaX2**
- double **areaY1**
- double **areaY2**

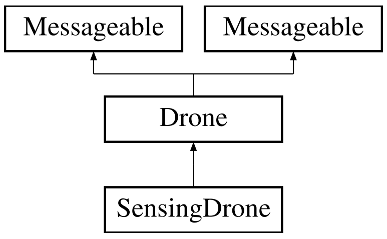
Additional Inherited Members

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

- code/programs/include/SensingBaseStation.hpp
- code/programs/src/SensingBaseStation.cpp

6.31 SensingDrone Class Reference

Inheritance diagram for SensingDrone:



Public Member Functions

- **SensingDrone** (CommMod ∗, double, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

Private Member Functions

- void **continueJob** ()
- void **interpretMessage** (Message ∗message)
- void **sendDataPoint** (double, double, double, double)
- int **atLoc** (Coord location)
- void **newArea** (double x1, double y1, double x2, double y2, double height)
- void **quit** ()

Private Attributes

- int **m_task**
- int ∗ **m_flag**
- double **sensorRadius**
- int **routingPriority**
- double **lastTime**
- double **waitTimer**
- bool **waiting**
- std::queue< Coord > **remainingPoints**
- bool **sink_node**
- std::string **baseStationIP**
- std::vector< std::string > **droneIPs**

Additional Inherited Members

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

- code/programs/include/SensingDrone.hpp
- code/programs/src/SensingDrone.cpp

Index