

TinyWindow

1.3.8

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>TinyWindow</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	windowManager::window_t Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	8
4.1.2	Constructor & Destructor Documentation . . . . .	8
4.1.2.1	window_t(const char *name=nullptr, unsigned int iD=0, unsigned int color← Bits=0, unsigned int depthBits=0, unsigned int stencilBits=0, bool should← Close=false, tinyWindowState_t currentState=tinyWindowState_t::NORMAL, std::function< void(unsigned int, tinyWindowKeyState_t)> keyEvent=nullptr, std::function< void(tinyWindowMouseButton_t, tinyWindowButtonState_t)> mouseButtonEvent=nullptr, std::function< void(tinyWindowMouseScroll_t)> mouseWheelEvent=nullptr, std::function< void(void)> destroyedEvent=nullptr, std::function< void(void)> maximizedEvent=nullptr, std::function< void(void)> minimizedEvent=nullptr, std::function< void(bool)> focusEvent=nullptr, std← ::function< void(unsigned int, unsigned int)> movedEvent=nullptr, std← ::function< void(unsigned int, unsigned int)> resizeEvent=nullptr, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> mouseMove← Event=nullptr) . . . . .	8
4.1.3	Member Data Documentation . . . . .	9
4.1.3.1	AtomActionClose . . . . .	9
4.1.3.2	AtomActionMaximizeHorz . . . . .	9
4.1.3.3	AtomActionMaximizeVert . . . . .	9
4.1.3.4	AtomActionMinimize . . . . .	9
4.1.3.5	AtomActionResize . . . . .	10

4.1.3.6	AtomActionShade	10
4.1.3.7	AtomActive	10
4.1.3.8	AtomAllowedActions	10
4.1.3.9	AtomCardinal	10
4.1.3.10	AtomClose	10
4.1.3.11	AtomDemandsAttention	10
4.1.3.12	AtomDesktopGeometry	11
4.1.3.13	AtomFocused	11
4.1.3.14	AtomFullScreen	11
4.1.3.15	AtomHidden	11
4.1.3.16	AtomHints	11
4.1.3.17	AtomIcon	11
4.1.3.18	AtomMaxHorz	12
4.1.3.19	AtomMaxVert	12
4.1.3.20	AtomState	12
4.1.3.21	AtomWindowType	12
4.1.3.22	AtomWindowTypeDesktop	12
4.1.3.23	AtomWindowTypeNormal	12
4.1.3.24	AtomWindowTypeSplash	12
4.1.3.25	attributes	12
4.1.3.26	colorBits	13
4.1.3.27	context	13
4.1.3.28	contextCreated	13
4.1.3.29	currentState	13
4.1.3.30	currentWindowStyle	13
4.1.3.31	decorators	13
4.1.3.32	depthBits	14
4.1.3.33	destroyedEvent	14
4.1.3.34	focusEvent	14
4.1.3.35	iD	14

4.1.3.36	<a href="#">inFocus</a>	14
4.1.3.37	<a href="#">initialized</a>	14
4.1.3.38	<a href="#">isCurrentContext</a>	15
4.1.3.39	<a href="#">keyEvent</a>	15
4.1.3.40	<a href="#">keys</a>	15
4.1.3.41	<a href="#">maximizedEvent</a>	15
4.1.3.42	<a href="#">minimizedEvent</a>	15
4.1.3.43	<a href="#">mouseButton</a>	15
4.1.3.44	<a href="#">mouseButtonEvent</a>	15
4.1.3.45	<a href="#">mouseMoveEvent</a>	16
4.1.3.46	<a href="#">mousePosition</a>	16
4.1.3.47	<a href="#">mouseWheelEvent</a>	16
4.1.3.48	<a href="#">movedEvent</a>	16
4.1.3.49	<a href="#">name</a>	16
4.1.3.50	<a href="#">position</a>	16
4.1.3.51	<a href="#">resizeEvent</a>	17
4.1.3.52	<a href="#">resolution</a>	17
4.1.3.53	<a href="#">setAttributes</a>	17
4.1.3.54	<a href="#">shouldClose</a>	17
4.1.3.55	<a href="#">stencilBits</a>	17
4.1.3.56	<a href="#">visualInfo</a>	17
4.1.3.57	<a href="#">windowHandle</a>	18
4.2	<a href="#">windowManager Class Reference</a>	18
4.2.1	<a href="#">Detailed Description</a>	21
4.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	22
4.2.2.1	<a href="#">windowManager()</a>	22
4.2.2.2	<a href="#">~windowManager(void)</a>	23
4.2.3	<a href="#">Member Function Documentation</a>	23
4.2.3.1	<a href="#">AddWindow(const char *windowName, unsigned int width=DEFAULT_WINDOW_WIDTH, unsigned int height=DEFAULT_WINDOW_HEIGHT, unsigned int colourBits=8, unsigned int depthBits=8, unsigned int stencilBits=8)</a>	23

4.2.3.2	DisableWindowDecoratorByIndex(unsigned int windowIndex, unsigned int decorators) . . . . .	24
4.2.3.3	DisableWindowDecoratorByName(const char *windowName, unsigned int decorators) . . . . .	25
4.2.3.4	DoesExistByIndex(unsigned int windowIndex) . . . . .	26
4.2.3.5	DoesExistByName(const char *windowName) . . . . .	27
4.2.3.6	EnableWindowDecoratorsByIndex(unsigned int windowIndex, unsigned int decorators) . . . . .	29
4.2.3.7	EnableWindowDecoratorsByName(const char *windowName, unsigned int decorators) . . . . .	30
4.2.3.8	FocusWindowByIndex(unsigned int windowIndex, bool newState) . . . . .	31
4.2.3.9	FocusWindowByName(const char *windowName, bool newState) . . . . .	32
4.2.3.10	GetInstance(void) . . . . .	33
4.2.3.11	GetMousePositionInScreen(unsigned int &x, unsigned int &y) . . . . .	34
4.2.3.12	GetMousePositionInScreen(void) . . . . .	34
4.2.3.13	GetMousePositionInWindowByIndex(unsigned int windowIndex, unsigned int &x, unsigned int &y) . . . . .	35
4.2.3.14	GetMousePositionInWindowByIndex(unsigned int windowIndex) . . . . .	35
4.2.3.15	GetMousePositionInWindowByName(const char *windowName, unsigned int &x, unsigned int &y) . . . . .	36
4.2.3.16	GetMousePositionInWindowByName(const char *windowName) . . . . .	37
4.2.3.17	GetNumWindows(void) . . . . .	37
4.2.3.18	GetScreenResolution(void) . . . . .	38
4.2.3.19	GetScreenResolution(unsigned int &width, unsigned int &Height) . . . . .	39
4.2.3.20	GetWindowByIndex(unsigned int windowIndex) . . . . .	40
4.2.3.21	GetWindowByName(const char *windowName) . . . . .	41
4.2.3.22	GetWindowIndexByName(const char *windowName) . . . . .	43
4.2.3.23	GetWindowsFullScreenByIndex(unsigned int windowIndex) . . . . .	44
4.2.3.24	GetWindowsFullScreenByName(const char *windowName) . . . . .	45
4.2.3.25	GetWindowsInFocusByIndex(unsigned int windowIndex) . . . . .	46
4.2.3.26	GetWindowsInFocusByName(const char *windowName) . . . . .	46
4.2.3.27	GetWindowsMaximizedByIndex(unsigned int windowIndex) . . . . .	47
4.2.3.28	GetWindowsMaximizedByName(const char *windowName) . . . . .	48

4.2.3.29	<a href="#">GetWindowIsMinimizedByIndex(unsigned int windowIndex)</a>	48
4.2.3.30	<a href="#">GetWindowIsMinimizedByName(const char *windowName)</a>	49
4.2.3.31	<a href="#">GetWindowNameByIndex(unsigned int windowIndex)</a>	50
4.2.3.32	<a href="#">GetWindowPositionByIndex(unsigned int windowIndex, unsigned int &amp;x, unsigned int &amp;y)</a>	50
4.2.3.33	<a href="#">GetWindowPositionByIndex(unsigned int windowIndex)</a>	51
4.2.3.34	<a href="#">GetWindowPositionByName(const char *windowName, unsigned int &amp;x, unsigned int &amp;y)</a>	51
4.2.3.35	<a href="#">GetWindowPositionByName(const char *windowName)</a>	52
4.2.3.36	<a href="#">GetWindowResolutionByIndex(unsigned int windowIndex, unsigned int &amp;width, unsigned int &amp;height)</a>	53
4.2.3.37	<a href="#">GetWindowResolutionByIndex(unsigned int windowIndex)</a>	54
4.2.3.38	<a href="#">GetWindowResolutionByName(const char *windowName, unsigned int &amp;width, unsigned int &amp;height)</a>	54
4.2.3.39	<a href="#">GetWindowResolutionByName(const char *windowName)</a>	55
4.2.3.40	<a href="#">GetWindowShouldCloseByIndex(unsigned int windowIndex)</a>	56
4.2.3.41	<a href="#">GetWindowShouldCloseByName(const char *windowName)</a>	57
4.2.3.42	<a href="#">Initialize(void)</a>	57
4.2.3.43	<a href="#">IsInitialized(void)</a>	58
4.2.3.44	<a href="#">IsValid(const char *stringParameter)</a>	59
4.2.3.45	<a href="#">MakeWindowCurrentContextByIndex(unsigned int windowIndex)</a>	60
4.2.3.46	<a href="#">MakeWindowCurrentContextByName(const char *windowName)</a>	61
4.2.3.47	<a href="#">MaximizeWindowByIndex(unsigned int windowIndex, bool newState)</a>	62
4.2.3.48	<a href="#">MaximizeWindowByName(const char *windowName, bool newState)</a>	63
4.2.3.49	<a href="#">MinimizeWindowByIndex(unsigned int windowIndex, bool newState)</a>	64
4.2.3.50	<a href="#">MinimizeWindowByName(const char *windowName, bool newState)</a>	64
4.2.3.51	<a href="#">Platform_DisableWindowDecorators(window_t *window, unsigned int decorators)</a>	65
4.2.3.52	<a href="#">Platform_EnableWindowDecorators(window_t *window, unsigned int decorators)</a>	67
4.2.3.53	<a href="#">Platform_FocusWindow(window_t *window, bool newState)</a>	69
4.2.3.54	<a href="#">Platform_InitializeGL(window_t *window)</a>	69
4.2.3.55	<a href="#">Platform_InitializeWindow(window_t *window)</a>	70
4.2.3.56	<a href="#">Platform_MakeCurrentContext(window_t *window)</a>	71

4.2.3.57	<code>Platform_MaximizeWindow(window_t *window, bool newState)</code>	71
4.2.3.58	<code>Platform_MinimizeWindow(window_t *window, bool newState)</code>	72
4.2.3.59	<code>Platform_RestoreWindow(window_t *window)</code>	73
4.2.3.60	<code>Platform_SetFullScreen(window_t *window)</code>	74
4.2.3.61	<code>Platform_SetMousePositionInWindow(window_t *window, unsigned int x, unsigned int y)</code>	75
4.2.3.62	<code>Platform_SetWindowPosition(window_t *window, unsigned int x, unsigned int y)</code>	76
4.2.3.63	<code>Platform_SetWindowResolution(window_t *window)</code>	76
4.2.3.64	<code>Platform_SetWindowStyle(window_t *window, tinyWindowStyle_t windowStyle)</code>	77
4.2.3.65	<code>Platform_SetWindowTitleBar(window_t *window, const char *newTitle)</code>	79
4.2.3.66	<code>Platform_SwapBuffers(window_t *window)</code>	79
4.2.3.67	<code>PollForEvents(void)</code>	80
4.2.3.68	<code>RemoveWindowByIndex(unsigned int windowIndex)</code>	81
4.2.3.69	<code>RemoveWindowByName(const char *windowName)</code>	82
4.2.3.70	<code>RestoreWindowByIndex(unsigned int windowIndex)</code>	82
4.2.3.71	<code>RestoreWindowByName(const char *windowName)</code>	83
4.2.3.72	<code>SetFullScreenByIndex(unsigned int windowIndex, bool newState)</code>	84
4.2.3.73	<code>SetFullScreenByName(const char *windowName, bool newState)</code>	84
4.2.3.74	<code>SetMousePositionInScreen(unsigned int x, unsigned int y)</code>	85
4.2.3.75	<code>SetMousePositionInWindowByIndex(unsigned int windowIndex, unsigned int x, unsigned int y)</code>	86
4.2.3.76	<code>SetMousePositionInWindowByName(const char *windowName, unsigned int x, unsigned int y)</code>	86
4.2.3.77	<code>SetWindowIconByIndex(void)</code>	87
4.2.3.78	<code>SetWindowIconByName(void)</code>	88
4.2.3.79	<code>SetWindowOnDestroyedByIndex(unsigned int windowIndex, std::function&lt; void(void)&gt; onDestroyed)</code>	89
4.2.3.80	<code>SetWindowOnDestroyedByName(const char *windowName, std::function&lt; void(void)&gt; onDestroyed)</code>	89
4.2.3.81	<code>SetWindowOnFocusByIndex(unsigned int windowIndex, std::function&lt; void(bool)&gt; onFocus)</code>	90
4.2.3.82	<code>SetWindowOnFocusByName(const char *windowName, std::function&lt; void(bool)&gt; onFocus)</code>	91



4.2.3.83	SetWindowOnKeyEventByIndex(unsigned int windowIndex, std::function< void(unsigned int, tinyWindowKeyState_t)> onKey)	91
4.2.3.84	SetWindowOnKeyEventByName(const char *windowName, std::function< void(unsigned int, tinyWindowKeyState_t)> onKey)	92
4.2.3.85	SetWindowOnMaximizedByIndex(unsigned int windowIndex, std::function< void(void)> onMaximized)	93
4.2.3.86	SetWindowOnMaximizedByName(const char *windowName, std::function< void(void)> onMaximized)	93
4.2.3.87	SetWindowOnMinimizedByIndex(unsigned int windowIndex, std::function< void(void)> onMinimized)	94
4.2.3.88	SetWindowOnMinimizedByName(const char *windowName, std::function< void(void)> onMinimized)	95
4.2.3.89	SetWindowOnMouseButtonEventByIndex(unsigned int windowIndex, std::function< void(tinyWindowMouseButton_t, tinyWindowButtonState_t)> onMouseButton)	95
4.2.3.90	SetWindowOnMouseButtonEventByName(const char *windowName, std::function< void(tinyWindowMouseButton_t, tinyWindowButtonState_t)> onMouseButton)	96
4.2.3.91	SetWindowOnMouseMoveByIndex(unsigned int windowIndex, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> onMouseMove)	97
4.2.3.92	SetWindowOnMouseMoveByName(const char *windowName, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> onMouseMove)	98
4.2.3.93	SetWindowOnMouseWheelEventByIndex(unsigned int windowIndex, std::function< void(tinyWindowMouseScroll_t)> onMouseWheel)	98
4.2.3.94	SetWindowOnMouseWheelEventByName(const char *windowName, std::function< void(tinyWindowMouseScroll_t)> onMouseWheel)	99
4.2.3.95	SetWindowOnMovedByIndex(unsigned int windowIndex, std::function< void(unsigned int, unsigned int)> onMoved)	100
4.2.3.96	SetWindowOnMovedByName(const char *windowName, std::function< void(unsigned int, unsigned int)> onMoved)	100
4.2.3.97	SetWindowOnResizeByIndex(unsigned int windowIndex, std::function< void(unsigned int, unsigned int)> onResize)	101
4.2.3.98	SetWindowOnResizeByName(const char *windowName, std::function< void(unsigned int, unsigned int)> onResize)	102
4.2.3.99	SetWindowPositionByIndex(unsigned int windowIndex, unsigned int x, unsigned int y)	102
4.2.3.100	SetWindowPositionByName(const char *windowName, unsigned int x, unsigned int y)	103
4.2.3.101	SetWindowResolutionByIndex(unsigned int windowIndex, unsigned int width, unsigned int height)	104

4.2.3.102 SetWindowResolutionByName(const char *windowName, unsigned int width, unsigned int height) . . . . .	105
4.2.3.103 SetWindowStyleByIndex(unsigned int windowIndex, tinyWindowStyle_t windowStyle) . . . . .	105
4.2.3.104 SetWindowStyleByName(const char *windowName, tinyWindowStyle_t windowStyle) . . . . .	106
4.2.3.105 SetWindowTitleBarByIndex(unsigned int windowIndex, const char *newName) . . . . .	107
4.2.3.106 SetWindowTitleBarByName(const char *windowName, const char *newTitle) . . . . .	108
4.2.3.107 ShutDown(void) . . . . .	108
4.2.3.108 ShutdownWindow(window_t *window) . . . . .	109
4.2.3.109 WaitForEvents(void) . . . . .	110
4.2.3.110 WindowExists(unsigned int windowIndex) . . . . .	111
4.2.3.111 WindowGetKeyByIndex(unsigned int windowIndex, unsigned int key) . . . . .	111
4.2.3.112 WindowGetKeyByName(const char *windowName, unsigned int key) . . . . .	112
4.2.3.113 WindowSwapBuffersByIndex(unsigned int windowIndex) . . . . .	113
4.2.3.114 WindowSwapBuffersByName(const char *windowName) . . . . .	114
4.2.4 Member Data Documentation . . . . .	114
4.2.4.1 instance . . . . .	114
4.2.4.2 isInitialized . . . . .	114
4.2.4.3 screenMousePosition . . . . .	115
4.2.4.4 screenResolution . . . . .	115
4.2.4.5 windowList . . . . .	115

<b>5</b>	<b>File Documentation</b>	<b>117</b>
5.1	Example/CMakeFiles/3.4.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference	117
5.1.1	Macro Definition Documentation	117
5.1.1.1	ARCHITECTURE_ID	117
5.1.1.2	COMPILER_ID	117
5.1.1.3	DEC	118
5.1.1.4	HEX	118
5.1.1.5	PLATFORM_ID	118
5.1.1.6	STRINGIFY	118
5.1.1.7	STRINGIFY_HELPER	118
5.1.2	Function Documentation	118
5.1.2.1	main(int argc, char *argv[])	118
5.1.3	Variable Documentation	119
5.1.3.1	info_arch	119
5.1.3.2	info_compiler	119
5.1.3.3	info_language_dialect_default	119
5.1.3.4	info_platform	119
5.2	Example/CMakeFiles/feature_tests.cxx File Reference	119
5.2.1	Function Documentation	120
5.2.1.1	main(int argc, char **argv)	120
5.2.2	Variable Documentation	120
5.2.2.1	features	120
5.3	Example/Example.cpp File Reference	120
5.3.1	Function Documentation	121
5.3.1.1	handleKeyPresses(unsigned int key, tinyWindowKeyState_t keyState)	121
5.3.1.2	main()	121
5.4	Include/TinyWindow.h File Reference	122
5.4.1	Enumeration Type Documentation	124
5.4.1.1	tinyWindowButtonState_t	124
5.4.1.2	tinyWindowDecorator_t	125

5.4.1.3	<a href="#">tinyWindowError_t</a>	125
5.4.1.4	<a href="#">tinyWindowKey_t</a>	126
5.4.1.5	<a href="#">tinyWindowKeyState_t</a>	128
5.4.1.6	<a href="#">tinyWindowMouseButton_t</a>	129
5.4.1.7	<a href="#">tinyWindowMouseScroll_t</a>	129
5.4.1.8	<a href="#">tinyWindowState_t</a>	129
5.4.1.9	<a href="#">tinyWindowStyle_t</a>	130
5.4.2	<a href="#">Function Documentation</a>	130
5.4.2.1	<a href="#">TinyWindow_PrintErrorMessage(const tinyWindowError_t errorNumber)</a>	130
5.4.3	<a href="#">Variable Documentation</a>	132
5.4.3.1	<a href="#">DEFAULT_WINDOW_HEIGHT</a>	132
5.4.3.2	<a href="#">DEFAULT_WINDOW_WIDTH</a>	133
5.4.3.3	<a href="#">LINUX_DECORATOR</a>	133
5.4.3.4	<a href="#">LINUX_FUNCTION</a>	133
5.5	<a href="#">README.md File Reference</a>	133
<b>Index</b>		<b>135</b>

## Chapter 1

# TinyWindow

a cross platform single header window management API



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">windowManager::window_t</a>	. . . . .	??
<a href="#">windowManager</a>	. . . . .	??





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

Example/ <a href="#">Example.cpp</a> . . . . .	??
Example/CMakeFiles/ <a href="#">feature_tests.cxx</a> . . . . .	??
Example/CMakeFiles/3.4.2/CompilerIdCXX/ <a href="#">CMakeCXXCompilerId.cpp</a> . . . . .	??
Include/ <a href="#">TinyWindow.h</a> . . . . .	??



## Chapter 4

# Class Documentation

### 4.1 windowManager::window\_t Struct Reference

#### Public Member Functions

- [window\\_t](#) (const char \*[name](#)=nullptr, unsigned int [iD](#)=0, unsigned int [colorBits](#)=0, unsigned int [depthBits](#)=0, unsigned int [stencilBits](#)=0, bool [shouldClose](#)=false, [tinyWindowState\\_t](#) [currentState](#)=[tinyWindowState\\_t::NORMAL](#), std::function< void(unsigned int, [tinyWindowKeyState\\_t](#))> [keyEvent](#)=nullptr, std::function< void([tinyWindowMouseButton\\_t](#), [tinyWindowButtonState\\_t](#))> [mouseButtonEvent](#)=nullptr, std::function< void([tinyWindowMouseScroll\\_t](#))> [mouseWheelEvent](#)=nullptr, std::function< void(void)> [destroyedEvent](#)=nullptr, std::function< void(void)> [maximizedEvent](#)=nullptr, std::function< void(void)> [minimizedEvent](#)=nullptr, std::function< void(bool)> [focusEvent](#)=nullptr, std::function< void(unsigned int, unsigned int)> [movedEvent](#)=nullptr, std::function< void(unsigned int, unsigned int)> [resizeEvent](#)=nullptr, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> [mouseMoveEvent](#)=nullptr)

#### Public Attributes

- const char \* [name](#)
- unsigned int [iD](#)
- int [colorBits](#)
- int [depthBits](#)
- int [stencilBits](#)
- [tinyWindowKeyState\\_t](#) [keys](#) [[KEY\\_LAST](#)]
- [tinyWindowButtonState\\_t](#) [mouseButton](#) [(unsigned int) [tinyWindowMouseButton\\_t::LAST](#)]
- unsigned int [resolution](#) [2]
- unsigned int [position](#) [2]
- unsigned int [mousePosition](#) [2]
- bool [shouldClose](#)
- bool [inFocus](#)
- bool [initialized](#)
- bool [contextCreated](#)
- bool [isCurrentContext](#)
- [tinyWindowState\\_t](#) [currentState](#)
- unsigned int [currentWindowStyle](#)
- std::function< void(unsigned int, [tinyWindowKeyState\\_t](#))> [keyEvent](#)
- std::function< void([tinyWindowMouseButton\\_t](#), [tinyWindowButtonState\\_t](#))> [mouseButtonEvent](#)
- std::function< void([tinyWindowMouseScroll\\_t](#))> [mouseWheelEvent](#)

- std::function< void(void)> [destroyedEvent](#)
- std::function< void(void)> [maximizedEvent](#)
- std::function< void(void)> [minimizedEvent](#)
- std::function< void(bool)> [focusEvent](#)
- std::function< void(unsigned int, unsigned int)> [movedEvent](#)
- std::function< void(unsigned int, unsigned int)> [resizeEvent](#)
- std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> [mouseMoveEvent](#)
- Window [windowHandle](#)
- GLXContext [context](#)
- XVisualInfo \* [visualInfo](#)
- int \* [attributes](#)
- XSetWindowAttributes [setAttributes](#)
- unsigned int [decorators](#)
- Atom [AtomState](#)
- Atom [AtomHidden](#)
- Atom [AtomFullScreen](#)
- Atom [AtomMaxHorz](#)
- Atom [AtomMaxVert](#)
- Atom [AtomClose](#)
- Atom [AtomActive](#)
- Atom [AtomDemandsAttention](#)
- Atom [AtomFocused](#)
- Atom [AtomCardinal](#)
- Atom [AtomIcon](#)
- Atom [AtomHints](#)
- Atom [AtomWindowType](#)
- Atom [AtomWindowTypeDesktop](#)
- Atom [AtomWindowTypeSplash](#)
- Atom [AtomWindowTypeNormal](#)
- Atom [AtomAllowedActions](#)
- Atom [AtomActionResize](#)
- Atom [AtomActionMinimize](#)
- Atom [AtomActionShade](#)
- Atom [AtomActionMaximizeHorz](#)
- Atom [AtomActionMaximizeVert](#)
- Atom [AtomActionClose](#)
- Atom [AtomDesktopGeometry](#)

#### 4.1.1 Detailed Description

Definition at line 2292 of file TinyWindow.h.

#### 4.1.2 Constructor & Destructor Documentation

- 4.1.2.1 **windowManager::window\_t::window\_t** ( const char \* *name* = nullptr, unsigned int *iD* = 0, unsigned int *colorBits* = 0, unsigned int *depthBits* = 0, unsigned int *stencilBits* = 0, bool *shouldClose* = false, tinyWindowState\_t *currentState* = tinyWindowState\_t::NORMAL, std::function< void(unsigned int, tinyWindowKeyState\_t)> *keyEvent* = nullptr, std::function< void(tinyWindowMouseButton\_t, tinyWindowButtonState\_t)> *mouseButtonEvent* = nullptr, std::function< void(tinyWindowMouseScroll\_t)> *mouseWheelEvent* = nullptr, std::function< void(void)> *destroyedEvent* = nullptr, std::function< void(void)> *maximizedEvent* = nullptr, std::function< void(void)> *minimizedEvent* = nullptr, std::function< void(bool)> *focusEvent* = nullptr, std::function< void(unsigned int, unsigned int)> *movedEvent* = nullptr, std::function< void(unsigned int, unsigned int)> *resizeEvent* = nullptr, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> *mouseMoveEvent* = nullptr ) [inline]

Definition at line 2374 of file TinyWindow.h.

References colorBits, contextCreated, currentState, currentWindowStyle, DEFAULT, depthBits, iD, initialized, name, shouldClose, and stencilBits.

```

2383     {
2384         this->name = name;
2385         this->iD = iD;
2386         this->colorBits = colorBits;
2387         this->depthBits = depthBits;
2388         this->stencilBits = stencilBits;
2389         this->shouldClose = shouldClose;
2390         this->currentState = currentState;
2391
2392         this->keyEvent = keyEvent;
2393         this->mouseButtonEvent = mouseButtonEvent;
2394         this->mouseWheelEvent = mouseWheelEvent;
2395         this->destroyedEvent = destroyedEvent;
2396         this->maximizedEvent = maximizedEvent;
2397         this->minimizedEvent = minimizedEvent;
2398         this->focusEvent = focusEvent;
2399         this->movedEvent = movedEvent;
2400         this->resizeEvent = resizeEvent;
2401         this->mouseMoveEvent = mouseMoveEvent;
2402
2403         initialized = false;
2404         contextCreated = false;
2405         currentWindowStyle = (unsigned int)
tinyWindowStyle_t::DEFAULT;
2406
2407 #if defined( __linux )
2408     context = 0;
2409 #endif
2410     }

```

### 4.1.3 Member Data Documentation

#### 4.1.3.1 Atom WindowManager::window\_t::AtomActionClose

Atom for allowing the window to be closed

Definition at line 2369 of file TinyWindow.h.

#### 4.1.3.2 Atom WindowManager::window\_t::AtomActionMaximizeHorz

Atom for allowing the window to be maximized horizontally

Definition at line 2367 of file TinyWindow.h.

#### 4.1.3.3 Atom WindowManager::window\_t::AtomActionMaximizeVert

Atom for allowing the window to be maximized vertically

Definition at line 2368 of file TinyWindow.h.

#### 4.1.3.4 Atom WindowManager::window\_t::AtomActionMinimize

Atom for allowing the window to be minimized

Definition at line 2365 of file TinyWindow.h.

#### 4.1.3.5 Atom windowManager::window\_t::AtomActionResize

Atom for allowing the window to be resized

Definition at line 2364 of file TinyWindow.h.

#### 4.1.3.6 Atom windowManager::window\_t::AtomActionShade

Atom for allowing the window to be shaded

Definition at line 2366 of file TinyWindow.h.

#### 4.1.3.7 Atom windowManager::window\_t::AtomActive

Atom for the active window

Definition at line 2351 of file TinyWindow.h.

#### 4.1.3.8 Atom windowManager::window\_t::AtomAllowedActions

Atom for allowed window actions

Definition at line 2363 of file TinyWindow.h.

#### 4.1.3.9 Atom windowManager::window\_t::AtomCardinal

Atom for cardinal coordinates

Definition at line 2354 of file TinyWindow.h.

#### 4.1.3.10 Atom windowManager::window\_t::AtomClose

Atom for closing the window

Definition at line 2350 of file TinyWindow.h.

#### 4.1.3.11 Atom windowManager::window\_t::AtomDemandsAttention

Atom for when the window demands attention

Definition at line 2352 of file TinyWindow.h.

#### 4.1.3.12 Atom windowManager::window\_t::AtomDesktopGeometry

Atom for Desktop Geometry

Definition at line 2371 of file TinyWindow.h.

Referenced by windowManager::AddWindow(), windowManager::DisableWindowDecoratorByIndex(), windowManager::DisableWindowDecoratorByName(), windowManager::EnableWindowDecoratorsByIndex(), windowManager::EnableWindowDecoratorsByName(), windowManager::FocusWindowByIndex(), windowManager::FocusWindowByName(), windowManager::MakeWindowCurrentContextByIndex(), windowManager::MakeWindowCurrentContextByName(), windowManager::MaximizeWindowByIndex(), windowManager::MaximizeWindowByName(), windowManager::MinimizeWindowByIndex(), windowManager::MinimizeWindowByName(), windowManager::RestoreWindowByIndex(), windowManager::RestoreWindowByName(), windowManager::SetFullScreenByIndex(), windowManager::SetFullScreenByName(), windowManager::SetMousePositionInWindowByIndex(), windowManager::SetMousePositionInWindowByName(), windowManager::SetWindowPositionByIndex(), windowManager::SetWindowPositionByName(), windowManager::SetWindowResolutionByIndex(), windowManager::SetWindowResolutionByName(), windowManager::SetWindowStyleByIndex(), windowManager::SetWindowStyleByName(), windowManager::SetWindowTitleBarByIndex(), windowManager::SetWindowTitleBarByName(), windowManager::WindowSwapBuffersByIndex(), and windowManager::WindowSwapBuffersByName().

#### 4.1.3.13 Atom windowManager::window\_t::AtomFocused

Atom for the focused state of the window

Definition at line 2353 of file TinyWindow.h.

#### 4.1.3.14 Atom windowManager::window\_t::AtomFullScreen

Atom for the full screen state of the window

Definition at line 2347 of file TinyWindow.h.

#### 4.1.3.15 Atom windowManager::window\_t::AtomHidden

Atom for the current hidden state of the window

Definition at line 2346 of file TinyWindow.h.

#### 4.1.3.16 Atom windowManager::window\_t::AtomHints

Atom for the window decorations

Definition at line 2356 of file TinyWindow.h.

#### 4.1.3.17 Atom windowManager::window\_t::AtomIcon

Atom for the icon of the window

Definition at line 2355 of file TinyWindow.h.

**4.1.3.18 Atom windowManager::window\_t::AtomMaxHorz**

Atom for the maximized horizontally state of the window

Definition at line 2348 of file TinyWindow.h.

**4.1.3.19 Atom windowManager::window\_t::AtomMaxVert**

Atom for the maximized vertically state of the window

Definition at line 2349 of file TinyWindow.h.

**4.1.3.20 Atom windowManager::window\_t::AtomState**

Atom for the state of the window

Definition at line 2345 of file TinyWindow.h.

**4.1.3.21 Atom windowManager::window\_t::AtomWindowType**

Atom for the type of window

Definition at line 2358 of file TinyWindow.h.

**4.1.3.22 Atom windowManager::window\_t::AtomWindowTypeDesktop**

Atom for the desktop window type

Definition at line 2359 of file TinyWindow.h.

**4.1.3.23 Atom windowManager::window\_t::AtomWindowTypeNormal**

Atom for the normal splash screen window type

Definition at line 2361 of file TinyWindow.h.

**4.1.3.24 Atom windowManager::window\_t::AtomWindowTypeSplash**

Atom for the splash screen window type

Definition at line 2360 of file TinyWindow.h.

**4.1.3.25 int\* windowManager::window\_t::attributes**

Attributes of the window. RGB, depth, stencil, etc

Definition at line 2339 of file TinyWindow.h.



#### 4.1.3.26 int windowManager::window\_t::colorBits

color format of the window. ( defaults to 32 bit color )

Definition at line 2296 of file TinyWindow.h.

Referenced by windowManager::AddWindow(), and window\_t().

#### 4.1.3.27 GLXContext windowManager::window\_t::context

The handle to the GLX rendering context

Definition at line 2337 of file TinyWindow.h.

#### 4.1.3.28 bool windowManager::window\_t::contextCreated

Whether the OpenGL context has been successfully created

Definition at line 2308 of file TinyWindow.h.

Referenced by windowManager::Platform\_InitializeGL(), and window\_t().

#### 4.1.3.29 tinyWindowState\_t windowManager::window\_t::currentState

The current state of the window. these states include Normal, Minimized, Maximized and Full screen

Definition at line 2311 of file TinyWindow.h.

Referenced by windowManager::GetWindowIsFullScreenByIndex(), windowManager::GetWindowIsFullScreenBy↵  
Name(), windowManager::GetWindowIsMaximizedByIndex(), windowManager::GetWindowIsMaximizedByName(),  
windowManager::GetWindowIsMinimizedByIndex(), windowManager::GetWindowIsMinimizedByName(), window↵  
Manager::Platform\_MaximizeWindow(), windowManager::Platform\_MinimizeWindow(), windowManager::SetFull↵  
ScreenByIndex(), windowManager::SetFullScreenByName(), and window\_t().

#### 4.1.3.30 unsigned int windowManager::window\_t::currentWindowStyle

The current style of the window

Definition at line 2312 of file TinyWindow.h.

Referenced by windowManager::Platform\_DisableWindowDecorators(), windowManager::Platform\_Enable↵  
WindowDecorators(), and window\_t().

#### 4.1.3.31 unsigned int windowManager::window\_t::decorators

Enabled window decorators

Definition at line 2341 of file TinyWindow.h.

#### 4.1.3.32 `int windowManager::window_t::depthBits`

Size of the Depth buffer. ( defaults to 8 bit depth )

Definition at line 2297 of file TinyWindow.h.

Referenced by `windowManager::AddWindow()`, and `window_t()`.

#### 4.1.3.33 `std::function<void(void)> windowManager::window_t::destroyedEvent`

This is the callback to be used when the window has been closed in a non-programmatic fashion

Definition at line 2317 of file TinyWindow.h.

#### 4.1.3.34 `std::function<void(bool)> windowManager::window_t::focusEvent`

This is the callback to be used when the window has been given focus in a non-programmatic fashion

Definition at line 2320 of file TinyWindow.h.

#### 4.1.3.35 `unsigned int windowManager::window_t::iD`

ID of the Window. ( where it belongs in the window manager )

Definition at line 2295 of file TinyWindow.h.

Referenced by `windowManager::AddWindow()`, `windowManager::GetWindowIndexByName()`, and `window_t()`.

#### 4.1.3.36 `bool windowManager::window_t::inFocus`

Whether the Window is currently in focus( if it is the current window be used )

Definition at line 2305 of file TinyWindow.h.

Referenced by `windowManager::GetWindowIsInFocusByIndex()`, and `windowManager::GetWindowIsInFocusByName()`.

#### 4.1.3.37 `bool windowManager::window_t::initialized`

Whether the window has been successfully initialized

Definition at line 2307 of file TinyWindow.h.

Referenced by `window_t()`.

**4.1.3.38 bool windowManager::window\_t::isCurrentContext**

Whether the window is the current window being drawn to

Definition at line 2309 of file TinyWindow.h.

**4.1.3.39 std::function<void(unsigned int, tinyWindowKeyState\_t)> windowManager::window\_t::keyEvent**

This is the callback to be used when a key has been pressed

Definition at line 2314 of file TinyWindow.h.

**4.1.3.40 tinyWindowKeyState\_t windowManager::window\_t::keys[KEY\_LAST]**

Record of keys that are either pressed or released in the respective window

Definition at line 2299 of file TinyWindow.h.

Referenced by windowManager::WindowGetKeyByIndex(), and windowManager::WindowGetKeyByName().

**4.1.3.41 std::function<void(void)> windowManager::window\_t::maximizedEvent**

This is the callback to be used when the window has been maximized in a non-programmatic fashion

Definition at line 2318 of file TinyWindow.h.

**4.1.3.42 std::function<void(void)> windowManager::window\_t::minimizedEvent**

This is the callback to be used when the window has been minimized in a non-programmatic fashion

Definition at line 2319 of file TinyWindow.h.

**4.1.3.43 tinyWindowButtonState\_t windowManager::window\_t::mouseButton[(unsigned int) tinyWindowMouseButton\_t::LAST]**

Record of mouse buttons that are either presses or released

Definition at line 2300 of file TinyWindow.h.

**4.1.3.44 std::function<void(tinyWindowMouseButton\_t, tinyWindowButtonState\_t)> windowManager::window\_t::mouseButtonEvent**

This is the callback to be used when a mouse button has been pressed

Definition at line 2315 of file TinyWindow.h.

**4.1.3.45** `std::function<void(unsigned int, unsigned int, unsigned int, unsigned int)> windowManager::window_t::mouseMoveEvent`

This is a callback to be used when the mouse has been moved

Definition at line 2323 of file TinyWindow.h.

**4.1.3.46** `unsigned int windowManager::window_t::mousePosition[2]`

Position of the Mouse cursor relative to the window co-ordinates

Definition at line 2303 of file TinyWindow.h.

Referenced by `windowManager::GetMousePositionInWindowByIndex()`, `windowManager::GetMousePositionInWindowByName()`, `windowManager::SetMousePositionInWindowByIndex()`, and `windowManager::SetMousePositionInWindowByName()`.

**4.1.3.47** `std::function<void(tinyWindowMouseScroll_t)> windowManager::window_t::mouseWheelEvent`

This is the callback to be used when the mouse wheel has been scrolled.

Definition at line 2316 of file TinyWindow.h.

**4.1.3.48** `std::function<void(unsigned int, unsigned int)> windowManager::window_t::movedEvent`

This is the callback to be used the window has been moved in a non-programmatic fashion

Definition at line 2321 of file TinyWindow.h.

**4.1.3.49** `const char* windowManager::window_t::name`

Name of the window

Definition at line 2294 of file TinyWindow.h.

Referenced by `windowManager::AddWindow()`, `windowManager::GetWindowNameByIndex()`, `windowManager::Platform_SetWindowStyle()`, `windowManager::ShutdownWindow()`, and `window_t()`.

**4.1.3.50** `unsigned int windowManager::window_t::position[2]`

Position of the Window relative to the screen co-ordinates

Definition at line 2302 of file TinyWindow.h.

Referenced by `windowManager::GetWindowPositionByIndex()`, `windowManager::GetWindowPositionByName()`, `windowManager::Platform_SetWindowResolution()`, `windowManager::SetWindowPositionByIndex()`, and `windowManager::SetWindowPositionByName()`.

**4.1.3.51** `std::function<void(unsigned int, unsigned int)> windowManager::window_t::resizeEvent`

This is a callback to be used when the window has been resized in a non-programmatic fashion

Definition at line 2322 of file TinyWindow.h.

**4.1.3.52** `unsigned int windowManager::window_t::resolution[2]`

Resolution/Size of the window stored in an array

Definition at line 2301 of file TinyWindow.h.

Referenced by `windowManager::AddWindow()`, `windowManager::GetWindowResolutionByIndex()`, `windowManager::GetWindowResolutionByName()`, `windowManager::Platform_SetWindowPosition()`, `windowManager::Platform_SetWindowResolution()`, `windowManager::SetWindowResolutionByIndex()`, and `windowManager::SetWindowResolutionByName()`.

**4.1.3.53** `XSetWindowAttributes windowManager::window_t::setAttributes`

The attributes to be set for the window

Definition at line 2340 of file TinyWindow.h.

**4.1.3.54** `bool windowManager::window_t::shouldClose`

Whether the Window should be closing

Definition at line 2304 of file TinyWindow.h.

Referenced by `windowManager::GetWindowShouldCloseByIndex()`, `windowManager::GetWindowShouldCloseByName()`, and `window_t()`.

**4.1.3.55** `int windowManager::window_t::stencilBits`

Size of the stencil buffer, ( defaults to 8 bit )

Definition at line 2298 of file TinyWindow.h.

Referenced by `windowManager::AddWindow()`, and `window_t()`.

**4.1.3.56** `XVisualInfo* windowManager::window_t::visualInfo`

The handle to the Visual Information. similar purpose to `PixelFormatDescriptor`

Definition at line 2338 of file TinyWindow.h.

#### 4.1.3.57 Window windowManager::window\_t::windowHandle

The X11 handle to the window. I wish they didn't name the type 'Window'

Definition at line 2336 of file TinyWindow.h.

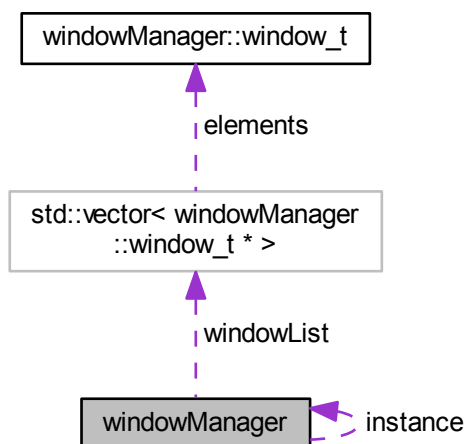
The documentation for this struct was generated from the following file:

- Include/[TinyWindow.h](#)

## 4.2 windowManager Class Reference

```
#include <TinyWindow.h>
```

Collaboration diagram for windowManager:



### Classes

- struct [window\\_t](#)

### Public Member Functions

- [windowManager](#) ()
- [~windowManager](#) (void)

## Static Public Member Functions

- static void [ShutDown](#) (void)
- static [windowManager](#) \* [AddWindow](#) (const char \*windowName, unsigned int width=DEFAULT\_WINDOW\_WIDTH, unsigned int height=DEFAULT\_WINDOW\_HEIGHT, unsigned int colourBits=8, unsigned int depthBits=8, unsigned int stencilBits=8)
- static int [GetNumWindows](#) (void)
- static bool [GetMousePositionInScreen](#) (unsigned int &x, unsigned int &y)
- static unsigned int \* [GetMousePositionInScreen](#) (void)
- static bool [SetMousePositionInScreen](#) (unsigned int x, unsigned int y)
- static unsigned int \* [GetScreenResolution](#) (void)
- static bool [GetScreenResolution](#) (unsigned int &width, unsigned int &Height)
- static bool [GetWindowResolutionByName](#) (const char \*windowName, unsigned int &width, unsigned int &height)
- static bool [GetWindowResolutionByIndex](#) (unsigned int windowIndex, unsigned int &width, unsigned int &height)
- static unsigned int \* [GetWindowResolutionByName](#) (const char \*windowName)
- static unsigned int \* [GetWindowResolutionByIndex](#) (unsigned int windowIndex)
- static bool [SetWindowResolutionByName](#) (const char \*windowName, unsigned int width, unsigned int height)
- static bool [SetWindowResolutionByIndex](#) (unsigned int windowIndex, unsigned int width, unsigned int height)
- static bool [GetWindowPositionByName](#) (const char \*windowName, unsigned int &x, unsigned int &y)
- static bool [GetWindowPositionByIndex](#) (unsigned int windowIndex, unsigned int &x, unsigned int &y)
- static unsigned int \* [GetWindowPositionByName](#) (const char \*windowName)
- static unsigned int \* [GetWindowPositionByIndex](#) (unsigned int windowIndex)
- static bool [SetWindowPositionByName](#) (const char \*windowName, unsigned int x, unsigned int y)
- static bool [SetWindowPositionByIndex](#) (unsigned int windowIndex, unsigned int x, unsigned int y)
- static bool [GetMousePositionInWindowByName](#) (const char \*windowName, unsigned int &x, unsigned int &y)
- static bool [GetMousePositionInWindowByIndex](#) (unsigned int windowIndex, unsigned int &x, unsigned int &y)
- static unsigned int \* [GetMousePositionInWindowByName](#) (const char \*windowName)
- static unsigned int \* [GetMousePositionInWindowByIndex](#) (unsigned int windowIndex)
- static bool [SetMousePositionInWindowByName](#) (const char \*windowName, unsigned int x, unsigned int y)
- static bool [SetMousePositionInWindowByIndex](#) (unsigned int windowIndex, unsigned int x, unsigned int y)
- static [tinyWindowKeyState\\_t](#) [WindowGetKeyByName](#) (const char \*windowName, unsigned int key)
- static [tinyWindowKeyState\\_t](#) [WindowGetKeyByIndex](#) (unsigned int windowIndex, unsigned int key)
- static bool [GetWindowShouldCloseByName](#) (const char \*windowName)
- static bool [GetWindowShouldCloseByIndex](#) (unsigned int windowIndex)
- static bool [WindowSwapBuffersByName](#) (const char \*windowName)
- static bool [WindowSwapBuffersByIndex](#) (unsigned int windowIndex)
- static bool [MakeWindowCurrentContextByName](#) (const char \*windowName)
- static bool [MakeWindowCurrentContextByIndex](#) (unsigned int windowIndex)
- static bool [GetWindowsFullScreenByName](#) (const char \*windowName)
- static bool [GetWindowsFullScreenByIndex](#) (unsigned int windowIndex)
- static bool [SetFullScreenByName](#) (const char \*windowName, bool newState)
- static bool [SetFullScreenByIndex](#) (unsigned int windowIndex, bool newState)
- static bool [GetWindowsMinimizedByName](#) (const char \*windowName)
- static bool [GetWindowsMinimizedByIndex](#) (unsigned int windowIndex)
- static bool [MinimizeWindowByName](#) (const char \*windowName, bool newState)
- static bool [MinimizeWindowByIndex](#) (unsigned int windowIndex, bool newState)
- static bool [GetWindowsMaximizedByName](#) (const char \*windowName)
- static bool [GetWindowsMaximizedByIndex](#) (unsigned int windowIndex)
- static bool [MaximizeWindowByName](#) (const char \*windowName, bool newState)
- static bool [MaximizeWindowByIndex](#) (unsigned int windowIndex, bool newState)
- static const char \* [GetWindowNameByIndex](#) (unsigned int windowIndex)
- static unsigned int [GetWindowIndexByName](#) (const char \*windowName)
- static bool [SetWindowTitleBarByName](#) (const char \*windowName, const char \*newTitle)

- static bool [SetWindowTitleBarByIndex](#) (unsigned int windowIndex, const char \*newName)
- static bool [SetWindowIconByName](#) (void)
- static bool [SetWindowIconByIndex](#) (void)
- static bool [GetWindowIsInFocusByName](#) (const char \*windowName)
- static bool [GetWindowIsInFocusByIndex](#) (unsigned int windowIndex)
- static bool [FocusWindowByName](#) (const char \*windowName, bool newState)
- static bool [FocusWindowByIndex](#) (unsigned int windowIndex, bool newState)
- static bool [RestoreWindowByName](#) (const char \*windowName)
- static bool [RestoreWindowByIndex](#) (unsigned int windowIndex)
- static bool [Initialize](#) (void)
- static bool [IsInitialized](#) (void)
- static void [PollForEvents](#) (void)
- static void [WaitForEvents](#) (void)
- static bool [RemoveWindowByName](#) (const char \*windowName)
- static bool [RemoveWindowByIndex](#) (unsigned int windowIndex)
- static bool [SetWindowStyleByName](#) (const char \*windowName, [tinyWindowStyle\\_t](#) windowStyle)
- static bool [SetWindowStyleByIndex](#) (unsigned int windowIndex, [tinyWindowStyle\\_t](#) windowStyle)
- static bool [EnableWindowDecoratorsByName](#) (const char \*windowName, unsigned int decorators)
- static bool [EnableWindowDecoratorsByIndex](#) (unsigned int windowIndex, unsigned int decorators)
- static bool [DisableWindowDecoratorByName](#) (const char \*windowName, unsigned int decorators)
- static bool [DisableWindowDecoratorByIndex](#) (unsigned int windowIndex, unsigned int decorators)
- static bool [SetWindowOnKeyEventByName](#) (const char \*windowName, std::function< void(unsigned int, [tinyWindowKeyState\\_t](#))> onKey)
- static bool [SetWindowOnKeyEventByIndex](#) (unsigned int windowIndex, std::function< void(unsigned int, [tinyWindowKeyState\\_t](#))> onKey)
- static bool [SetWindowOnMouseButtonEventByName](#) (const char \*windowName, std::function< void([tinyWindowMouseButton\\_t](#), [tinyWindowButtonState\\_t](#))> onMouseButton)
- static bool [SetWindowOnMouseButtonEventByIndex](#) (unsigned int windowIndex, std::function< void([tinyWindowMouseButton\\_t](#), [tinyWindowButtonState\\_t](#))> onMouseButton)
- static bool [SetWindowOnMouseWheelEventByName](#) (const char \*windowName, std::function< void([tinyWindowMouseScroll\\_t](#))> onMouseWheel)
- static bool [SetWindowOnMouseWheelEventByIndex](#) (unsigned int windowIndex, std::function< void([tinyWindowMouseScroll\\_t](#))> onMouseWheel)
- static bool [SetWindowOnDestroyedByName](#) (const char \*windowName, std::function< void(void)> onDestroyed)
- static bool [SetWindowOnDestroyedByIndex](#) (unsigned int windowIndex, std::function< void(void)> onDestroyed)
- static bool [SetWindowOnMaximizedByName](#) (const char \*windowName, std::function< void(void)> onMaximized)
- static bool [SetWindowOnMaximizedByIndex](#) (unsigned int windowIndex, std::function< void(void)> onMaximized)
- static bool [SetWindowOnMinimizedByName](#) (const char \*windowName, std::function< void(void)> onMinimized)
- static bool [SetWindowOnMinimizedByIndex](#) (unsigned int windowIndex, std::function< void(void)> onMinimized)
- static bool [SetWindowOnFocusByName](#) (const char \*windowName, std::function< void(bool)> onFocus)
- static bool [SetWindowOnFocusByIndex](#) (unsigned int windowIndex, std::function< void(bool)> onFocus)
- static bool [SetWindowOnMovedByName](#) (const char \*windowName, std::function< void(unsigned int, unsigned int)> onMoved)
- static bool [SetWindowOnMovedByIndex](#) (unsigned int windowIndex, std::function< void(unsigned int, unsigned int)> onMoved)
- static bool [SetWindowOnResizeByName](#) (const char \*windowName, std::function< void(unsigned int, unsigned int)> onResize)
- static bool [SetWindowOnResizeByIndex](#) (unsigned int windowIndex, std::function< void(unsigned int, unsigned int)> onResize)



- static bool [SetWindowOnMouseMoveByName](#) (const char \*windowName, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> onMouseMove)
- static bool [SetWindowOnMouseMoveByIndex](#) (unsigned int windowIndex, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> onMouseMove)

### Static Private Member Functions

- static bool [IsValid](#) (const char \*stringParameter)
- static bool [WindowExists](#) (unsigned int windowIndex)
- static [windowManager](#) \* [GetInstance](#) (void)
- static void [Platform\\_InitializeWindow](#) ([window\\_t](#) \*window)
- static bool [Platform\\_InitializeGL](#) ([window\\_t](#) \*window)
- static void [Platform\\_SetWindowResolution](#) ([window\\_t](#) \*window)
- static void [Platform\\_SetWindowPosition](#) ([window\\_t](#) \*window, unsigned int x, unsigned int y)
- static void [Platform\\_SetMousePositionInWindow](#) ([window\\_t](#) \*window, unsigned int x, unsigned int y)
- static void [Platform\\_SwapBuffers](#) ([window\\_t](#) \*window)
- static void [Platform\\_MakeCurrentContext](#) ([window\\_t](#) \*window)
- static void [Platform\\_SetFullScreen](#) ([window\\_t](#) \*window)
- static void [Platform\\_MinimizeWindow](#) ([window\\_t](#) \*window, bool newState)
- static void [Platform\\_MaximizeWindow](#) ([window\\_t](#) \*window, bool newState)
- static void [Platform\\_SetWindowTitleBar](#) ([window\\_t](#) \*window, const char \*newTitle)
- static void [Platform\\_FocusWindow](#) ([window\\_t](#) \*window, bool newState)
- static void [Platform\\_RestoreWindow](#) ([window\\_t](#) \*window)
- static void [Platform\\_SetWindowStyle](#) ([window\\_t](#) \*window, [tinyWindowStyle\\_t](#) windowStyle)
- static void [Platform\\_EnableWindowDecorators](#) ([window\\_t](#) \*window, unsigned int decorators)
- static void [Platform\\_DisableWindowDecorators](#) ([window\\_t](#) \*window, unsigned int decorators)
- static void [ShutdownWindow](#) ([window\\_t](#) \*window)
- static bool [DoesExistByName](#) (const char \*windowName)
- static bool [DoesExistByIndex](#) (unsigned int windowIndex)
- static [window\\_t](#) \* [GetWindowByName](#) (const char \*windowName)
- static [window\\_t](#) \* [GetWindowByIndex](#) (unsigned int windowIndex)

### Private Attributes

- std::vector< [window\\_t](#) \* > [windowList](#)
- unsigned int [screenResolution](#) [2]
- unsigned int [screenMousePosition](#) [2]
- bool [isInitialized](#)

### Static Private Attributes

- static [windowManager](#) \* [instance](#) = nullptr

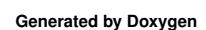
#### 4.2.1 Detailed Description

Definition at line 342 of file TinyWindow.h.

#### 4.2.2.1 windowManager::windowManager ( ) [inline]

Referenced by `GetInstance()`.

Here is the caller graph for this function:



## 4.2.2.2 WindowManager::~~WindowManager ( void ) [inline]

Shutdown and delete all windows in the manager

Definition at line 353 of file TinyWindow.h.

```

354     {
355         if ( !GetInstance()->windowList.empty() )
356         {
357             for ( auto CurrentWindow : GetInstance()->windowList )
358             {
359                 delete CurrentWindow;
360             }
361             GetInstance()->windowList.clear();
362         }
363     }

```

## 4.2.3 Member Function Documentation

4.2.3.1 static WindowManager\* WindowManager::AddWindow ( const char \* *windowName*, unsigned int *width* = DEFAULT\_WINDOW\_WIDTH, unsigned int *height* = DEFAULT\_WINDOW\_HEIGHT, unsigned int *colourBits* = 8, unsigned int *depthBits* = 8, unsigned int *stencilBits* = 8 ) [inline], [static]

Use this to add a window to the manager. returns a pointer to the manager which allows for the easy creation of multiple windows

Definition at line 392 of file TinyWindow.h.

References `WindowManager::window_t::AtomDesktopGeometry`, `WindowManager::window_t::colorBits`, `WindowManager::window_t::depthBits`, `GetInstance()`, `GetNumWindows()`, `WindowManager::window_t::iD`, `instance`, `INVALID_WINDOW_NAME`, `IsInitialized()`, `IsValid()`, `WindowManager::window_t::name`, `NOT_INITIALIZED`, `Platform_InitializeWindow()`, `WindowManager::window_t::resolution`, `WindowManager::window_t::stencilBits`, and `TinyWindow_PrintErrorMessage()`.

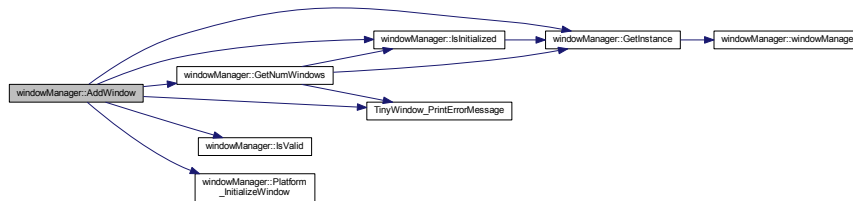
Referenced by `main()`.

```

393     {
394         if ( GetInstance()->IsInitialized() )
395         {
396             if ( IsValid( windowName ) )
397             {
398                 window_t* newWindow = new window_t;
399                 newWindow->name = windowName;
400                 newWindow->resolution[ 0 ] = width;
401                 newWindow->resolution[ 1 ] = height;
402                 newWindow->colorBits = colourBits;
403                 newWindow->depthBits = depthBits;
404                 newWindow->stencilBits = stencilBits;
405
406                 instance->windowList.push_back( newWindow );
407                 newWindow->iD = GetNumWindows() - 1;
408
409                 Platform_InitializeWindow( newWindow );
410
411                 return instance;
412             }
413             TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_WINDOW_NAME);
414             return nullptr;
415         }
416
417         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
418         return nullptr;
419     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.2** `static bool windowManager::DisableWindowDecoratorByIndex ( unsigned int windowIndex, unsigned int decorators )`  
`[inline], [static]`

Disable windows decorators by index

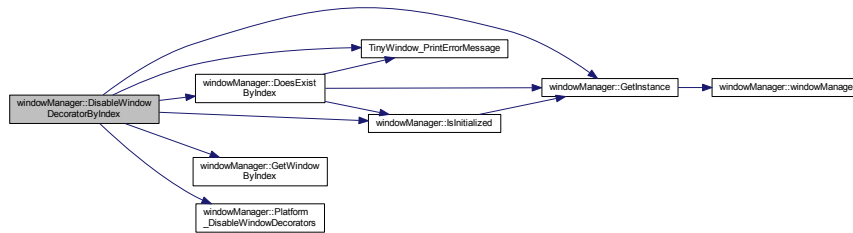
Definition at line 1801 of file TinyWindow.h.

References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_DisableWindowDecorators()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1802     {
1803         if ( GetInstance()->IsInitialized() )
1804         {
1805             if ( DoesExistByIndex( windowIndex ) )
1806             {
1807                 window_t* window = GetWindowByIndex(windowIndex);
1808                 Platform_DisableWindowDecorators(window, decorators);
1809                 return true;
1810             }
1811             TinyWindow_PrintErrorMessage(
1812                 tinyWindowError_t::WINDOW_NOT_FOUND);
1812             return false;
1813         }
1814         TinyWindow_PrintErrorMessage(
1815             tinyWindowError_t::NOT_INITIALIZED );
1815         return false;
1816     }
  
```

Here is the call graph for this function:



**4.2.3.3** `static bool windowManager::DisableWindowDecoratorByName ( const char * windowName, unsigned int decorators )`  
`[inline], [static]`

Disable windows decorators by name

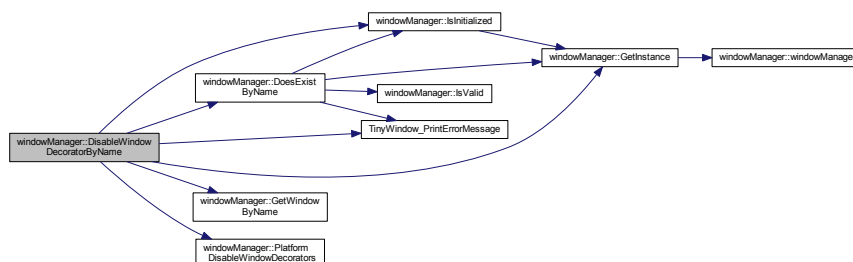
Definition at line 1782 of file TinyWindow.h.

References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_DisableWindowDecorators()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1783     {
1784         if ( GetInstance()->IsInitialized() )
1785         {
1786             if ( DoesExistByName( windowName ) )
1787             {
1788                 window_t* window = GetWindowByName(windowName);
1789                 Platform_DisableWindowDecorators(window, decorators);
1790                 return true;
1791             }
1792             TinyWindow_PrintErrorMessage(
1793 tinyWindowError_t::WINDOW_NOT_FOUND);
1794             return false;
1795         }
1796         TinyWindow_PrintErrorMessage(
1797 tinyWindowError_t::NOT_INITIALIZED );
1798         return false;
1799     }
  
```

Here is the call graph for this function:



**4.2.3.4** `static bool WindowManager::DoesExistByIndex ( unsigned int windowIndex )` `[inline], [static], [private]`

Definition at line 3073 of file TinyWindow.h.

References `GetInstance()`, `INVALID_WINDOW_INDEX`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

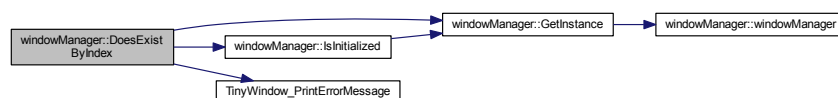
Referenced by `DisableWindowDecoratorByIndex()`, `EnableWindowDecoratorsByIndex()`, `FocusWindowByIndex()`, `GetMousePositionInWindowByIndex()`, `GetWindowIsFullScreenByIndex()`, `GetWindowIsInFocusByIndex()`, `GetWindowIsMaximizedByIndex()`, `GetWindowIsMinimizedByIndex()`, `GetWindowNameByIndex()`, `GetWindowPositionByIndex()`, `GetWindowResolutionByIndex()`, `GetWindowShouldCloseByIndex()`, `MakeWindowCurrentContextByIndex()`, `MaximizeWindowByIndex()`, `MinimizeWindowByIndex()`, `RemoveWindowByIndex()`, `RestoreWindowByIndex()`, `SetFullScreenByIndex()`, `SetMousePositionInWindowByIndex()`, `SetWindowOnDestroyedByIndex()`, `SetWindowOnFocusByIndex()`, `SetWindowOnKeyEventByIndex()`, `SetWindowOnMaximizedByIndex()`, `SetWindowOnMinimizedByIndex()`, `SetWindowOnMouseButtonEventByIndex()`, `SetWindowOnMouseMoveByIndex()`, `SetWindowOnMouseWheelEventByIndex()`, `SetWindowOnMovedByIndex()`, `SetWindowOnResizeByIndex()`, `SetWindowPositionByIndex()`, `SetWindowStyleByIndex()`, `SetWindowTitleBarByIndex()`, `WindowGetKeyByIndex()`, and `WindowSwapBuffersByIndex()`.

```

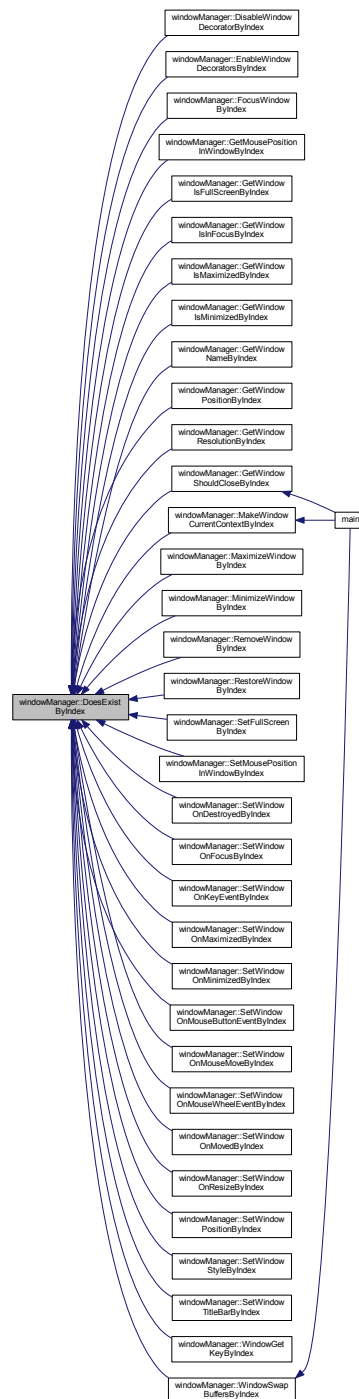
3074     {
3075         if ( GetInstance()->IsInitialized() )
3076         {
3077             if ( windowIndex <= ( instance->windowList.size() - 1 ) )
3078             {
3079                 return true;
3080             }
3081
3082             TinyWindow_PrintErrorMessage(
3083 tinyWindowError_t::INVALID_WINDOW_INDEX );
3084             return false;
3085         }
3086
3087         TinyWindow_PrintErrorMessage(
3088 tinyWindowError_t::NOT_INITIALIZED );
3089         return false;
3090     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.5** `static bool windowManager::DoesExistByName ( const char * windowName )` `[inline], [static], [private]`

Definition at line 3052 of file TinyWindow.h.

References `GetInstance()`, `INVALID_WINDOW_NAME`, `IsInitialized()`, `IsValid()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

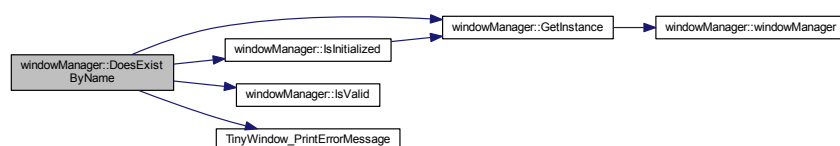
Referenced by DisableWindowDecoratorByName(), EnableWindowDecoratorsByName(), FocusWindowByName(), GetMousePositionInWindowByName(), GetWindowIndexByName(), GetWindowIsFullScreenByName(), GetWindowIsInFocusByName(), GetWindowIsMaximizedByName(), GetWindowIsMinimizedByName(), GetWindowPositionByName(), GetWindowResolutionByName(), GetWindowShouldCloseByName(), MakeWindowCurrentContextByName(), MaximizeWindowByName(), MinimizeWindowByName(), RemoveWindowByName(), RestoreWindowByName(), SetFullScreenByName(), SetMousePositionInWindowByName(), SetWindowOnDestroyedByName(), SetWindowOnFocusByName(), SetWindowOnKeyEventByName(), SetWindowOnMaximizedByName(), SetWindowOnMinimizedByName(), SetWindowOnMouseButtonEventByName(), SetWindowOnMouseMoveByName(), SetWindowOnMouseWheelEventByName(), SetWindowOnMovedByName(), SetWindowOnResizeByName(), SetWindowPositionByName(), SetWindowResolutionByName(), SetWindowStyleByName(), SetWindowTitleBarByName(), WindowGetKeyByName(), and WindowSwapBuffersByName().

```

3053     {
3054         if ( GetInstance()->IsInitialized() )
3055         {
3056             if ( IsValid( windowName ) )
3057             {
3058                 for ( auto window : instance->windowList )
3059                 {
3060                     if( !strcmp( window->name, windowName ) )
3061                     {
3062                         return true;
3063                     }
3064                 }
3065             }
3066             TinyWindow_PrintErrorMessage(
3067 tinyWindowError_t::INVALID_WINDOW_NAME );
3068             return false;
3069         }
3070         TinyWindow_PrintErrorMessage(
3071 tinyWindowError_t::NOT_INITIALIZED);
3072         return false;
3073     }

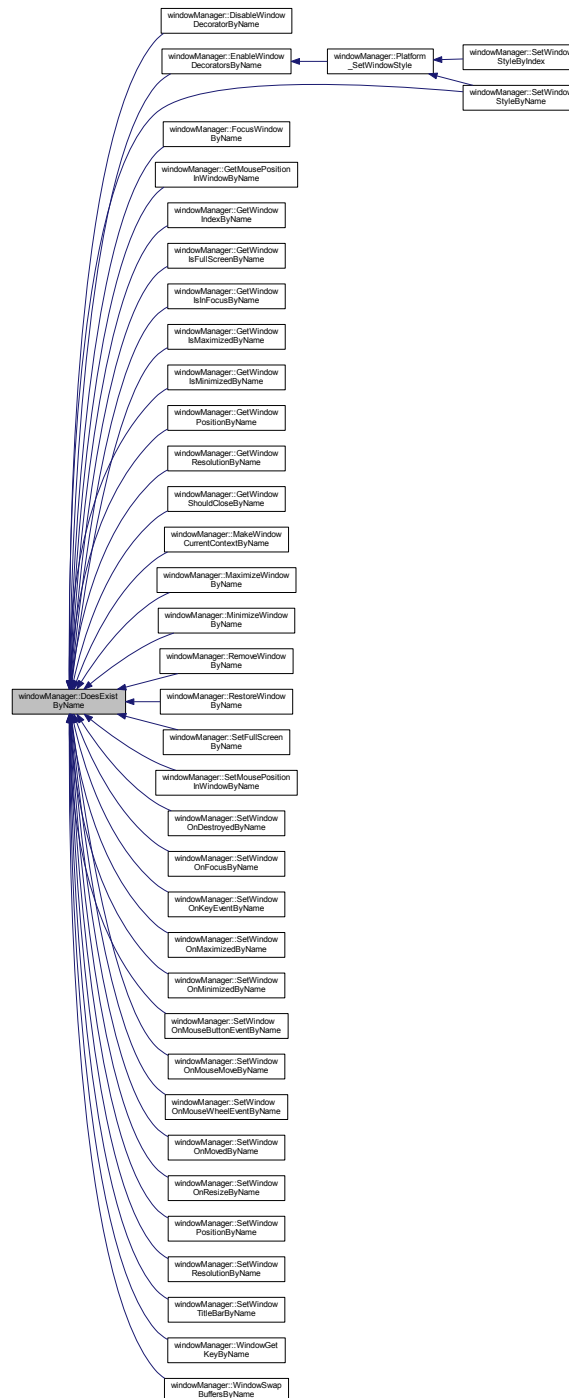
```

Here is the call graph for this function:





Here is the caller graph for this function:



**4.2.3.6** `static bool WindowManager::EnableWindowDecoratorsByIndex ( unsigned int windowIndex, unsigned int decorators )`  
`[inline], [static]`

Enable windows decorators by index

Definition at line 1762 of file TinyWindow.h.

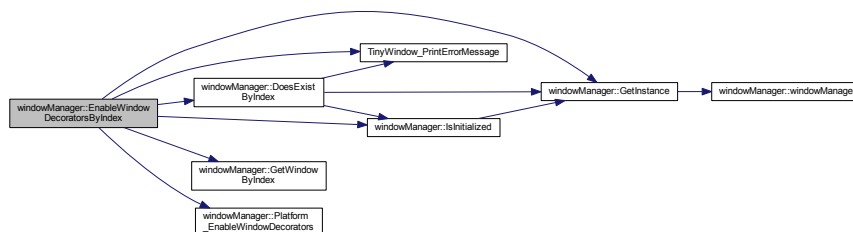
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_EnableWindowDecorators()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1763     {
1764         if (GetInstance()->IsInitialized())
1765         {
1766             if (DoesExistByIndex(windowIndex))
1767             {
1768                 window_t* window = GetWindowByIndex(windowIndex);
1769                 Platform_EnableWindowDecorators(window, decorators);
1770                 return true;
1771             }
1772             TinyWindow_PrintErrorMessage(
1773                 tinyWindowError_t::WINDOW_NOT_FOUND);
1774             return false;
1775         }
1776         TinyWindow_PrintErrorMessage(
1777             tinyWindowError_t::NOT_INITIALIZED);
1778         return false;
1779     }

```

Here is the call graph for this function:



#### 4.2.3.7 static bool windowManager::EnableWindowDecoratorsByName ( const char \* *windowName*, unsigned int *decorators* ) [inline], [static]

Enable window decorators by name

Definition at line 1743 of file `TinyWindow.h`.

References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_EnableWindowDecorators()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

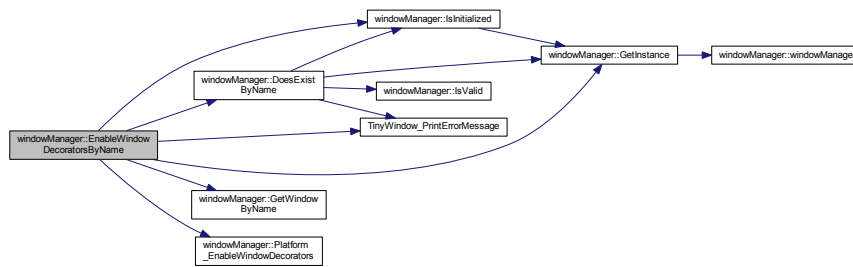
Referenced by `Platform_SetWindowStyle()`.

```

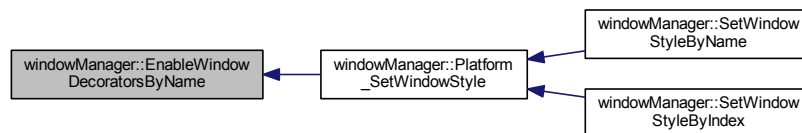
1744     {
1745         if ( GetInstance()->IsInitialized() )
1746         {
1747             if ( DoesExistByName( windowName ) )
1748             {
1749                 window_t* window = GetWindowByName(windowName);
1750                 Platform_EnableWindowDecorators(window, decorators);
1751                 return true;
1752             }
1753             TinyWindow_PrintErrorMessage(
1754                 tinyWindowError_t::WINDOW_NOT_FOUND);
1755             return false;
1756         }
1757         TinyWindow_PrintErrorMessage(
1758             tinyWindowError_t::NOT_INITIALIZED );
1759         return false;
1760     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.8** `static bool WindowManager::FocusWindowByIndex ( unsigned int windowIndex, bool newState ) [inline], [static]`

Set the window to be in focus by index

Definition at line 1493 of file TinyWindow.h.

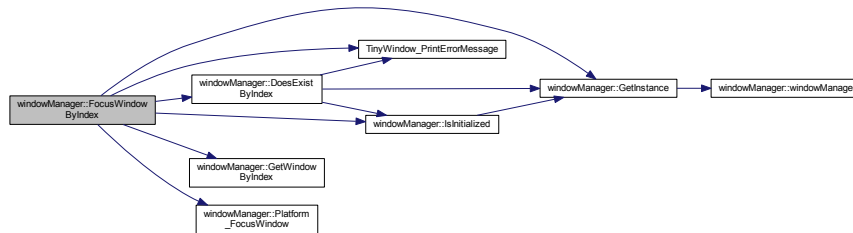
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_FocusWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1494     {
1495         if (GetInstance()->IsInitialized())
1496         {
1497             if (DoesExistByIndex(windowIndex))
1498             {
1499                 window_t* window = GetWindowByIndex(windowIndex);
1500                 Platform_FocusWindow(window, newState);
1501                 return true;
1502             }
1503             TinyWindow_PrintErrorMessage(
1504                 tinyWindowError_t::WINDOW_NOT_FOUND);
1505             return false;
1506         }
1507         TinyWindow_PrintErrorMessage(
1508             tinyWindowError_t::NOT_INITIALIZED);
1509         return false;
1510     }

```

Here is the call graph for this function:



**4.2.3.9** `static bool windowManager::FocusWindowByName ( const char * windowName, bool newState )` `[inline]`,  
`[static]`

Set the window to be in focus by name

Definition at line 1474 of file TinyWindow.h.

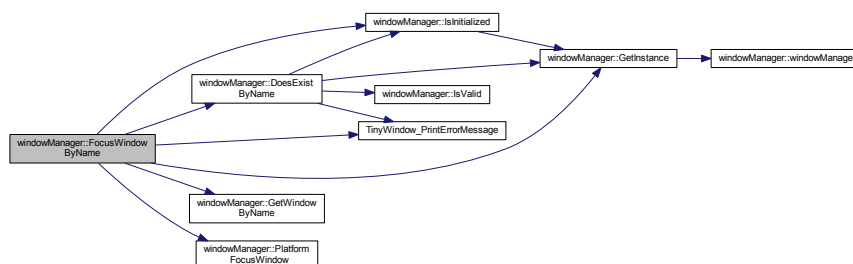
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_FocusWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1475     {
1476         if ( GetInstance()->IsInitialized() )
1477         {
1478             if ( DoesExistByName( windowName ) )
1479             {
1480                 window_t* window = GetWindowByName(windowName);
1481                 Platform_FocusWindow(window, newState);
1482                 return true;
1483             }
1484             TinyWindow_PrintErrorMessage(
1485                 tinyWindowError_t::WINDOW_NOT_FOUND);
1486             return false;
1487         }
1488         TinyWindow_PrintErrorMessage(
1489             tinyWindowError_t::NOT_INITIALIZED );
1490         return false;
1491     }

```

Here is the call graph for this function:



## 4.2.3.10 static windowManager\* windowManager::GetInstance ( void ) [inline],[static],[private]

Definition at line 2432 of file TinyWindow.h.

References instance, and windowManager().

Referenced by AddWindow(), DisableWindowDecoratorByIndex(), DisableWindowDecoratorByName(), DoesExistByIndex(), DoesExistByName(), EnableWindowDecoratorsByIndex(), EnableWindowDecoratorsByName(), FocusWindowByIndex(), FocusWindowByName(), GetMousePositionInScreen(), GetMousePositionInWindowByIndex(), GetMousePositionInWindowByName(), GetNumWindows(), GetScreenResolution(), GetWindowIndexByName(), GetWindowsFullScreenByIndex(), GetWindowsFullScreenByName(), GetWindowsInFocusByIndex(), GetWindowsInFocusByName(), GetWindowsMaximizedByIndex(), GetWindowsMaximizedByName(), GetWindowsMinimizedByIndex(), GetWindowsMinimizedByName(), GetWindowNameByIndex(), GetWindowPositionByIndex(), GetWindowPositionByName(), GetWindowResolutionByIndex(), GetWindowResolutionByName(), GetWindowShouldCloseByIndex(), GetWindowShouldCloseByName(), Initialize(), IsInitialized(), MakeWindowCurrentContextByIndex(), MakeWindowCurrentContextByName(), MaximizeWindowByIndex(), MaximizeWindowByName(), MinimizeWindowByIndex(), MinimizeWindowByName(), PollForEvents(), RemoveWindowByIndex(), RemoveWindowByName(), RestoreWindowByIndex(), RestoreWindowByName(), SetFullScreenByIndex(), SetFullScreenByName(), SetMousePositionInScreen(), SetMousePositionInWindowByIndex(), SetMousePositionInWindowByName(), SetWindowOnDestroyedByIndex(), SetWindowOnDestroyedByName(), SetWindowOnFocusByIndex(), SetWindowOnFocusByName(), SetWindowOnKeyEventByIndex(), SetWindowOnKeyEventByName(), SetWindowOnMaximizedByIndex(), SetWindowOnMaximizedByName(), SetWindowOnMinimizedByIndex(), SetWindowOnMinimizedByName(), SetWindowOnMouseButtonEventByIndex(), SetWindowOnMouseButtonEventByName(), SetWindowOnMouseMoveByIndex(), SetWindowOnMouseMoveByName(), SetWindowOnMouseWheelEventByIndex(), SetWindowOnMouseWheelEventByName(), SetWindowOnMovedByIndex(), SetWindowOnMovedByName(), SetWindowOnResizeByIndex(), SetWindowOnResizeByName(), SetWindowPositionByIndex(), SetWindowPositionByName(), SetWindowResolutionByIndex(), SetWindowResolutionByName(), SetWindowStyleByIndex(), SetWindowStyleByName(), SetWindowTitleBarByIndex(), SetWindowTitleBarByName(), ShutDown(), WaitForEvents(), WindowGetKeyByIndex(), WindowGetKeyByName(), WindowSwapBuffersByIndex(), and WindowSwapBuffersByName().

```

2433     {
2434         if ( windowManager::instance == nullptr )
2435         {
2436             windowManager::instance = new windowManager();
2437             return windowManager::instance;
2438         }
2439         else
2440         {
2441             return windowManager::instance;
2442         }
2443     }
2444 }
```

Here is the call graph for this function:



#### 4.2.3.11 static bool windowManager::GetMousePositionInScreen ( unsigned int & x, unsigned int & y ) [inline], [static]

Return the mouse position in screen co-ordinates

Definition at line 438 of file TinyWindow.h.

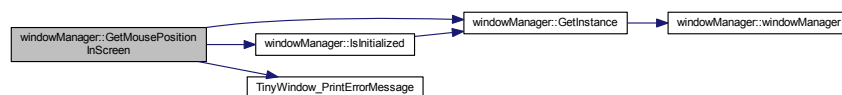
References [GetInstance\(\)](#), [instance](#), [IsInitialized\(\)](#), [NOT\\_INITIALIZED](#), [screenMousePosition](#), and [TinyWindow\\_↵PrintErrorMessage\(\)](#).

```

439     {
440         if ( GetInstance()->IsInitialized() )
441         {
442             x = instance->screenMousePosition[0];
443             y = instance->screenMousePosition[1];
444             return true;
445         }
446
447         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
448         return false;
449     }

```

Here is the call graph for this function:



#### 4.2.3.12 static unsigned int\* windowManager::GetMousePositionInScreen ( void ) [inline], [static]

Return the mouse position in screen co-ordinates

Definition at line 453 of file TinyWindow.h.

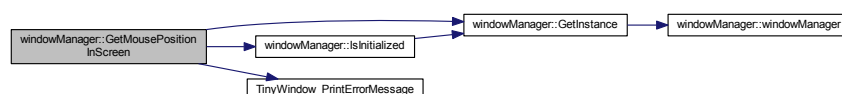
References [GetInstance\(\)](#), [instance](#), [IsInitialized\(\)](#), [NOT\\_INITIALIZED](#), [screenMousePosition](#), and [TinyWindow\\_↵PrintErrorMessage\(\)](#).

```

454     {
455         if ( GetInstance()->IsInitialized() )
456         {
457             return instance->screenMousePosition;
458         }
459
460         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
461         return nullptr;
462     }

```

Here is the call graph for this function:



#### 4.2.3.13 static bool WindowManager::GetMousePositionInWindowByIndex ( unsigned int *windowIndex*, unsigned int & *x*, unsigned int & *y* ) [inline],[static]

Return the mouse position relative to the given window's co-ordinates by setting X and Y

Definition at line 812 of file TinyWindow.h.

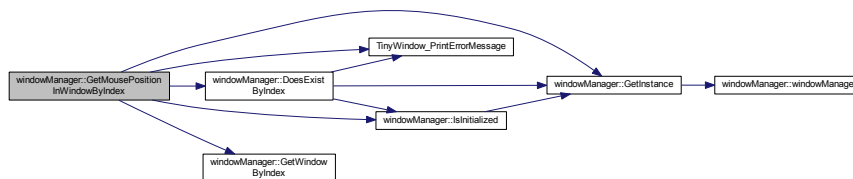
References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `WindowManager::window_t::mousePosition`, `NOT_INITIALIZED`, `TINYWINDOW_ERROR`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

813     {
814         if ( GetInstance()->IsInitialized() )
815         {
816             if ( DoesExistByIndex( windowIndex ) )
817             {
818                 x = GetWindowByIndex( windowIndex )->
mousePosition[ 0 ];
819                 y = GetWindowByIndex( windowIndex )->
mousePosition[ 1 ];
820                 return true;
821             }
822             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
823             return false;
824         }
825         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
826         return (bool)tinyWindowError_t::TINYWINDOW_ERROR;
827     }

```

Here is the call graph for this function:



#### 4.2.3.14 static unsigned int\* WindowManager::GetMousePositionInWindowByIndex ( unsigned int *windowIndex* ) [inline],[static]

Return the mouse Position relative to the given window's co-ordinates as an array

Definition at line 849 of file TinyWindow.h.

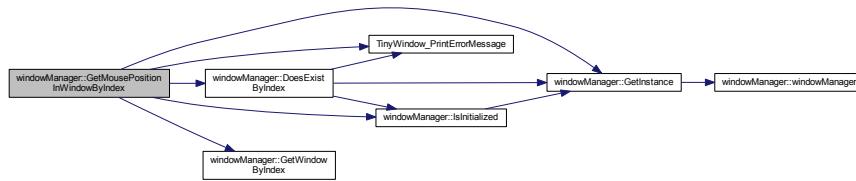
References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `WindowManager::window_t::mousePosition`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

```

850     {
851         if ( GetInstance()->IsInitialized() )
852         {
853             if ( DoesExistByIndex( windowIndex ) )
854             {
855                 return GetWindowByIndex( windowIndex )->
mousePosition;
856             }
857             return nullptr;
858         }
859         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
860         return nullptr;
861     }

```

Here is the call graph for this function:



**4.2.3.15** `static bool windowManager::GetMousePositionInWindowByName ( const char * windowName, unsigned int & x, unsigned int & y )` `[inline], [static]`

Return the mouse Position relative to the given window's co-ordinates by setting X and Y

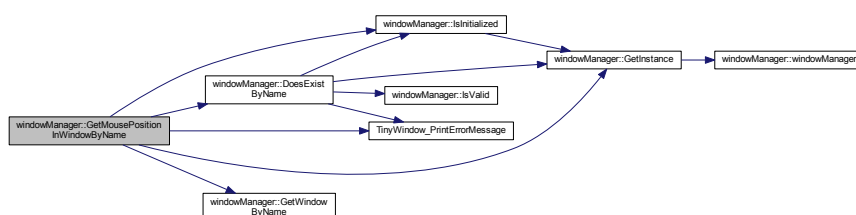
Definition at line 793 of file TinyWindow.h.

References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `windowManager::window_t::mousePosition`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

794 {
795     if ( GetInstance()->IsInitialized() )
796     {
797         if ( DoesExistByName( windowName ) )
798         {
799             x = GetWindowByName( windowName )->mousePosition[ 0 ];
800             y = GetWindowByName( windowName )->mousePosition[ 1 ];
801             return true;
802         }
803         TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
804         return false;
805     }
806     TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
807     return false;
808 }
  
```

Here is the call graph for this function:





#### 4.2.3.16 static unsigned int\* WindowManager::GetMousePositionInWindowByName ( const char \* *windowName* ) [inline],[static]

Return the mouse Position relative to the given window's co-ordinates as an array

Definition at line 832 of file TinyWindow.h.

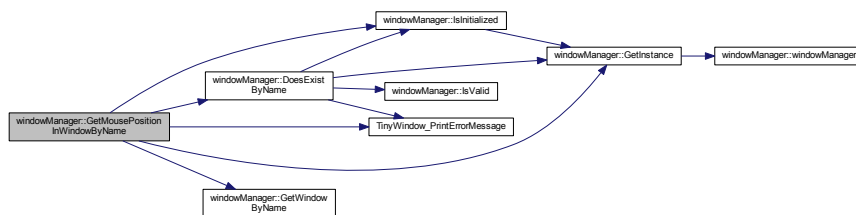
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `WindowManager::window_`, `tinyWindowError_t::mousePosition`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

833     {
834         if ( GetInstance()->IsInitialized() )
835         {
836             if ( DoesExistByName( windowName ) )
837             {
838                 return GetWindowByName( windowName )->
mousePosition;
839             }
840             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
841             return nullptr;
842         }
843         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
844         return nullptr;
845     }

```

Here is the call graph for this function:



#### 4.2.3.17 static int WindowManager::GetNumWindows ( void ) [inline],[static]

Return the total amount of windows the manager has

Definition at line 424 of file TinyWindow.h.

References `GetInstance()`, `IsInitialized()`, `NOT_INITIALIZED`, `TINYWINDOW_ERROR`, and `TinyWindow_PrintErrorMessage()`.

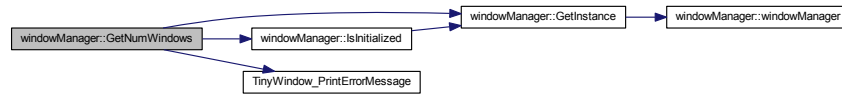
Referenced by `AddWindow()`.

```

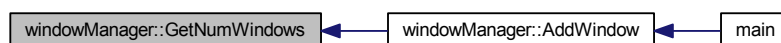
425     {
426         if ( GetInstance()->IsInitialized() )
427         {
428             return instance->windowList.size();
429         }
430
431         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
432         return (int)tinyWindowError_t::TINYWINDOW_ERROR;
433     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.3.18 static unsigned int\* windowManager::GetScreenResolution ( void ) [inline],[static]

Return the Resolution of the current screen

Definition at line 492 of file TinyWindow.h.

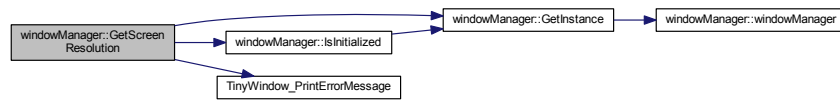
References `GetInstance()`, `instance`, `IsInitialized()`, `NOT_INITIALIZED`, `screenResolution`, and `TinyWindow_PrintErrorMessage()`.

Referenced by `Platform_SetFullScreen()`.

```

493     {
494         if ( GetInstance()->IsInitialized() )
495         {
496 #if defined( _WIN32 ) || defined( _WIN64 )
497             RECT screen;
498             HWND desktop = GetDesktopWindow();
499             GetWindowRect( desktop, &screen );
500
501             instance->screenResolution[0] = screen.right;
502             instance->screenResolution[1] = screen.bottom;
503             return instance->screenResolution;
504 #elif defined( __linux__ )
505             instance->screenResolution[0] = WidthOfScreen(XDefaultScreenOfDisplay(
506             instance->currentDisplay));
507             instance->screenResolution[1] = HeightOfScreen(XDefaultScreenOfDisplay(
508             instance->currentDisplay));
509             return instance->screenResolution;
510 #endif
511         }
512         TinyWindow_PrintErrorMessage(
513         tinyWindowError_t::NOT_INITIALIZED );
514         return nullptr;
515     }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.19** `static bool WindowManager::GetScreenResolution ( unsigned int & width, unsigned int & Height ) [inline], [static]`

Return the Resolution of the current screen

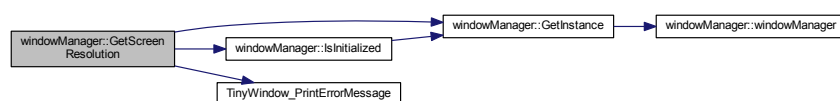
Definition at line 519 of file TinyWindow.h.

References `GetInstance()`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

```

520     {
521         if ( GetInstance()->IsInitialized() )
522         {
523             #if defined( _WIN32 ) || defined( _WIN64 )
524                 RECT screen;
525                 HWND desktop = GetDesktopWindow();
526                 GetWindowRect( desktop, &screen );
527                 width = screen.right;
528                 Height = screen.bottom;
529             #elif defined( __linux__ )
530                 width = WidthOfScreen(XDefaultScreenOfDisplay(instance->currentDisplay));
531                 Height = HeightOfScreen(XDefaultScreenOfDisplay(instance->currentDisplay));
532             #endif
533             instance->screenResolution[0] = width;
534             instance->screenResolution[1] = Height;
535         #endif
536         return true;
537     }
538
539     TinyWindow_PrintErrorMessage(
540         tinyWindowError_t::NOT_INITIALIZED );
541     return false;
542 }
  
```

Here is the call graph for this function:



**4.2.3.20** `static window_t* WindowManager::GetWindowByIndex ( unsigned int windowIndex )` `[inline],[static],`  
`[private]`

Definition at line 3104 of file TinyWindow.h.

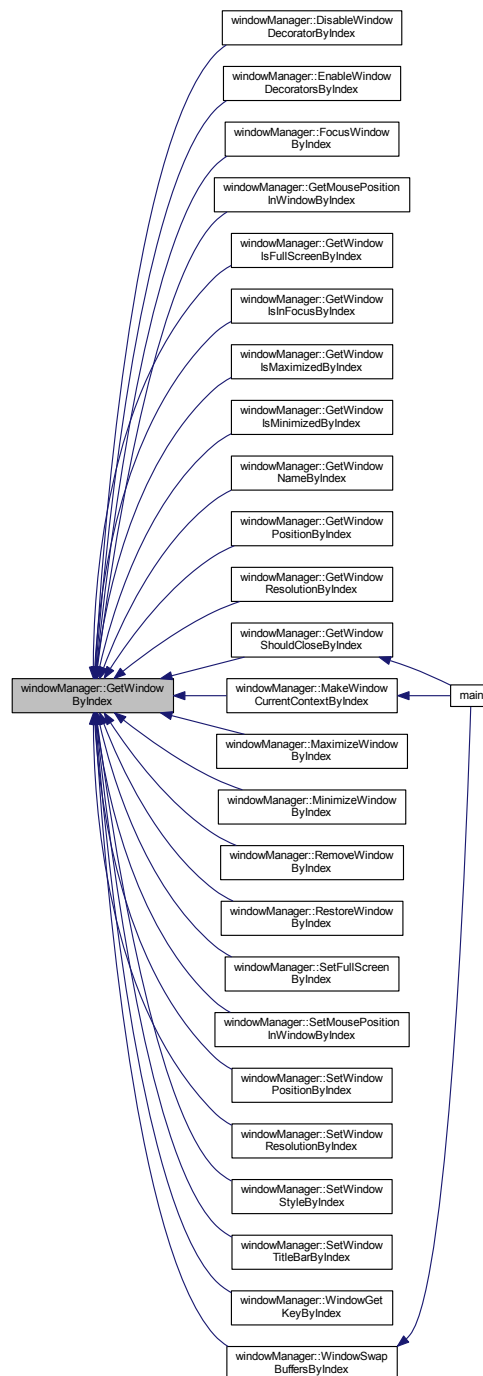
Referenced by `DisableWindowDecoratorByIndex()`, `EnableWindowDecoratorsByIndex()`, `FocusWindowByIndex()`, `GetMousePositionInWindowByIndex()`, `GetWindowIsFullScreenByIndex()`, `GetWindowIsInFocusByIndex()`, `GetWindowIsMaximizedByIndex()`, `GetWindowIsMinimizedByIndex()`, `GetWindowNameByIndex()`, `GetWindowPositionByIndex()`, `GetWindowResolutionByIndex()`, `GetWindowShouldCloseByIndex()`, `MakeWindowCurrentContextByIndex()`, `MaximizeWindowByIndex()`, `MinimizeWindowByIndex()`, `RemoveWindowByIndex()`, `RestoreWindowByIndex()`, `SetFullScreenByIndex()`, `SetMousePositionInWindowByIndex()`, `SetWindowPositionByIndex()`, `SetWindowResolutionByIndex()`, `SetWindowStyleByIndex()`, `SetWindowTitleBarByIndex()`, `WindowGetKeyByIndex()`, and `WindowSwapBuffersByIndex()`.

```

3105     {
3106         if ( windowIndex <= instance->windowList.size() - 1 )
3107         {
3108             return instance->windowList[windowIndex];
3109         }
3110         return nullptr;
3111     }

```

Here is the caller graph for this function:



**4.2.3.21** `static window_t* WindowManager::GetWindowByName ( const char * windowName )` `[inline], [static], [private]`

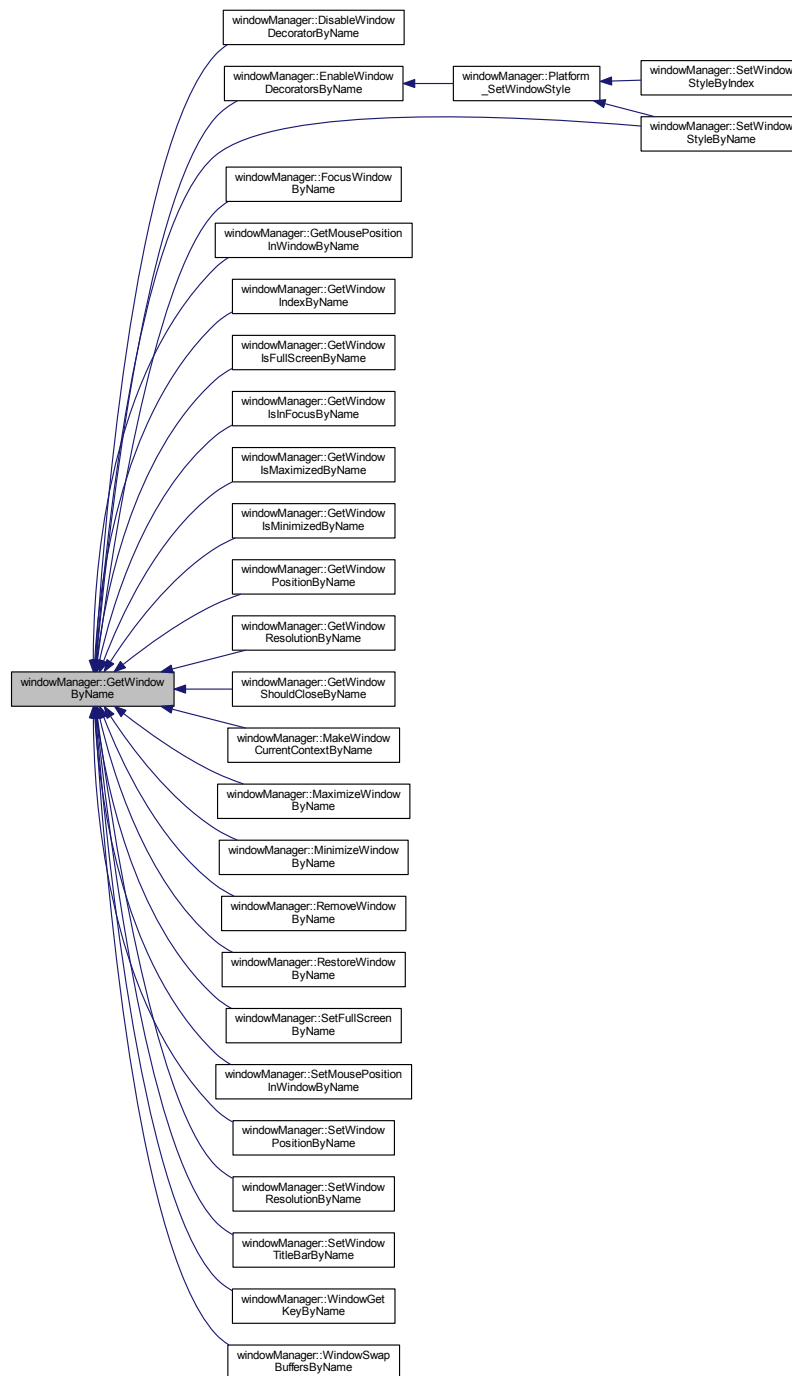
Definition at line 3090 of file TinyWindow.h.

Referenced by DisableWindowDecoratorByName(), EnableWindowDecoratorsByName(), FocusWindowByName(), GetMousePositionInWindowByName(), GetWindowIndexByName(), GetWindowIsFullScreenByName(), GetWindowIsInFocusByName(), GetWindowIsMaximizedByName(), GetWindowIsMinimizedByName(), GetWindowPositionByName(), GetWindowResolutionByName(), GetWindowShouldCloseByName(), MakeWindowCurrentContextByName(), MaximizeWindowByName(), MinimizeWindowByName(), RemoveWindowByName(), RestoreWindowByName(), SetFullScreenByName(), SetMousePositionInWindowByName(), SetWindowPositionByName(), SetWindowResolutionByName(), SetWindowStyleByName(), SetWindowTitleBarByName(), WindowGetKeyByName(), and WindowSwapBuffersByName().

```

3091     {
3092         for( auto window : instance->windowList )
3093         {
3094             if ( !strcmp( window->name, windowName ) )
3095             {
3096                 return window;
3097             }
3098         }
3099         return nullptr;
3100     }
3101 
```

Here is the caller graph for this function:



**4.2.3.22** `static unsigned int windowManager::GetWindowIndexByName ( const char * windowName )` `[inline]`,  
`[static]`

Get window index by name

Definition at line 1309 of file TinyWindow.h.

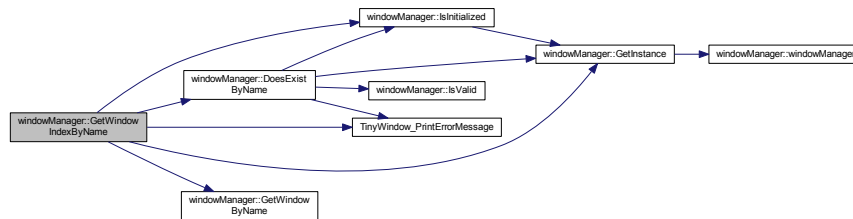
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `windowManager::window_t::iD`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1310     {
1311         if ( GetInstance()->IsInitialized() )
1312         {
1313             if ( DoesExistByName( windowName ) )
1314             {
1315                 return GetWindowByName( windowName )->iD;
1316             }
1317             TinyWindow_PrintErrorMessage(
1318 tinyWindowError_t::WINDOW_NOT_FOUND);
1319             return false;
1320         }
1321         TinyWindow_PrintErrorMessage(
1322 tinyWindowError_t::NOT_INITIALIZED );
1323         return false;
1324     }

```

Here is the call graph for this function:



**4.2.3.23** `static bool windowManager::GetWindowsFullScreenByIndex ( unsigned int windowIndex ) [inline], [static]`

Return whether the given window is in full screen mode

Definition at line 1083 of file `TinyWindow.h`.

References `windowManager::window_t::currentState`, `DoesExistByIndex()`, `FULLSCREEN`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

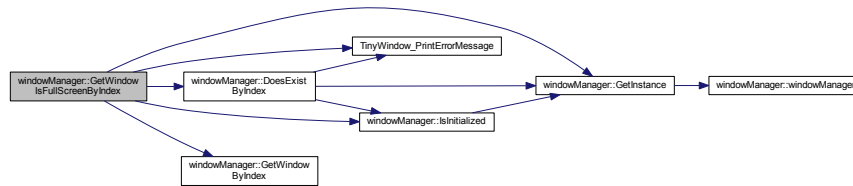
```

1084     {
1085         if ( GetInstance()->IsInitialized() )
1086         {
1087             if ( DoesExistByIndex( windowIndex ) )
1088             {
1089                 return (GetWindowByIndex(windowIndex)->currentState ==
1090 tinyWindowState_t::FULLSCREEN);
1091             }
1092             return false;
1093         }
1094         TinyWindow_PrintErrorMessage(
1095 tinyWindowError_t::NOT_INITIALIZED);
1096         return false;
1097     }

```



Here is the call graph for this function:



**4.2.3.24** `static bool WindowManager::GetWindowIsFullScreenByName ( const char * windowName )` [inline], [static]

Return whether the given window is in full screen mode

Definition at line 1066 of file TinyWindow.h.

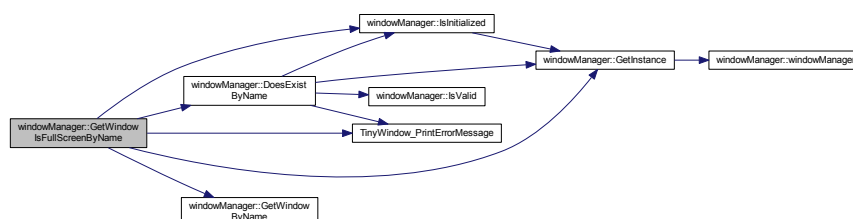
References `windowManager::window_t::currentState`, `DoesExistByName()`, `FULLSCREEN`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

```

1067     {
1068         if ( GetInstance()->IsInitialized() )
1069         {
1070             if ( DoesExistByName( windowName ) )
1071             {
1072                 return ( GetWindowByName( windowName )->currentState ==
1073 tinyWindowState_t::FULLSCREEN );
1074             }
1075             return false;
1076         }
1077         TinyWindow_PrintErrorMessage(
1078 tinyWindowError_t::NOT_INITIALIZED);
1078         return false;
1079     }

```

Here is the call graph for this function:



#### 4.2.3.25 static bool windowManager::GetWindowIsInFocusByIndex ( unsigned int *windowIndex* ) [inline],[static]

Get whether the window is in focus by index

Definition at line 1455 of file TinyWindow.h.

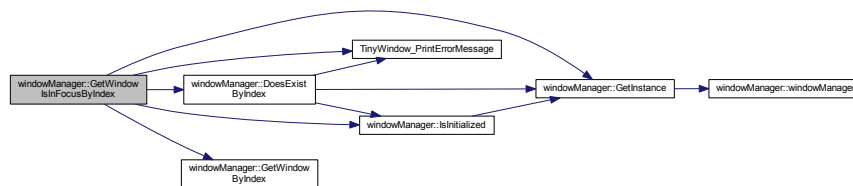
References [DoesExistByIndex\(\)](#), [GetInstance\(\)](#), [GetWindowByIndex\(\)](#), [windowManager::window\\_t::inFocus](#), [IsInitialized\(\)](#), [NOT\\_INITIALIZED](#), [TinyWindow\\_PrintErrorMessage\(\)](#), and [WINDOW\\_NOT\\_FOUND](#).

```

1456     {
1457         if ( GetInstance()->IsInitialized() )
1458         {
1459             if ( DoesExistByIndex( windowIndex ) )
1460             {
1461                 return GetWindowByIndex( windowIndex )->inFocus;
1462             }
1463             TinyWindow_PrintErrorMessage(
1464 tinyWindowError_t::WINDOW_NOT_FOUND);
1464             return false;
1465         }
1466         TinyWindow_PrintErrorMessage(
1467 tinyWindowError_t::NOT_INITIALIZED );
1468         return false;
1469     }

```

Here is the call graph for this function:



#### 4.2.3.26 static bool windowManager::GetWindowIsInFocusByName ( const char \* *windowName* ) [inline],[static]

Get whether the window is in focus by name

Definition at line 1438 of file TinyWindow.h.

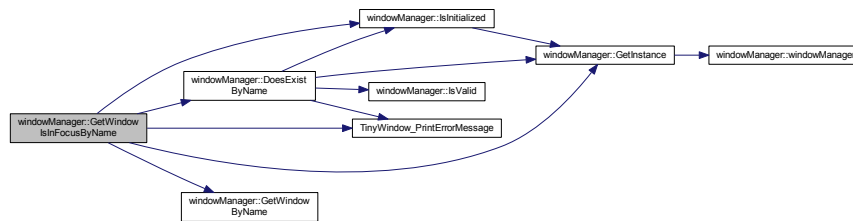
References [DoesExistByName\(\)](#), [GetInstance\(\)](#), [GetWindowByName\(\)](#), [windowManager::window\\_t::inFocus](#), [IsInitialized\(\)](#), [NOT\\_INITIALIZED](#), [TinyWindow\\_PrintErrorMessage\(\)](#), and [WINDOW\\_NOT\\_FOUND](#).

```

1439     {
1440         if ( GetInstance()->IsInitialized() )
1441         {
1442             if ( DoesExistByName( windowName ) )
1443             {
1444                 return GetWindowByName( windowName )->inFocus;
1445             }
1446             TinyWindow_PrintErrorMessage(
1447 tinyWindowError_t::WINDOW_NOT_FOUND);
1447             return false;
1448         }
1449         TinyWindow_PrintErrorMessage(
1450 tinyWindowError_t::NOT_INITIALIZED );
1450         return false;
1451     }

```

Here is the call graph for this function:



**4.2.3.27** `static bool WindowManager::GetWindowIsMaximizedByIndex ( unsigned int windowIndex ) [inline], [static]`

Return whether the given window is currently maximized

Definition at line 1235 of file TinyWindow.h.

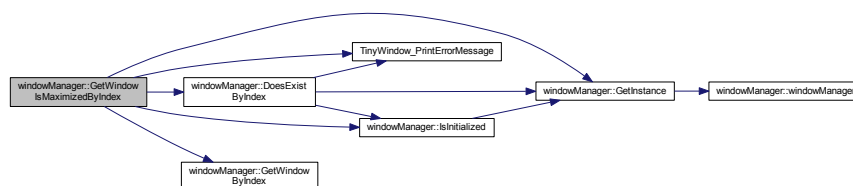
References `windowManager::window_t::currentState`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `MAXIMIZED`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1236     {
1237         if ( GetInstance()->IsInitialized() )
1238         {
1239             if ( DoesExistByIndex( windowIndex ) )
1240             {
1241                 return (GetWindowByIndex(windowIndex)->currentState ==
tinyWindowState_t::MAXIMIZED);
1242             }
1243             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1244             return false;
1245         }
1246         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1247         return false;
1248     }

```

Here is the call graph for this function:



**4.2.3.28** `static bool windowManager::GetWindowIsMaximizedByName ( const char * windowName )` `[inline]`,  
`[static]`

Return whether the current window is currently maximized

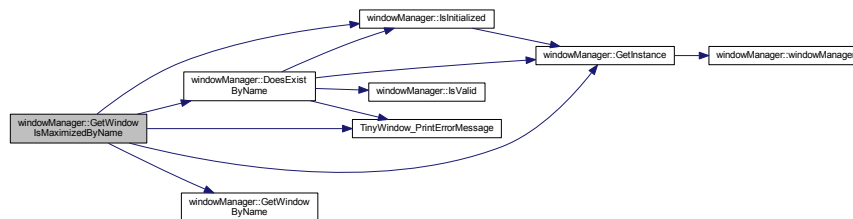
Definition at line 1217 of file TinyWindow.h.

References `windowManager::window_t::currentState`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `MAXIMIZED`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1218     {
1219         if ( GetInstance()->IsInitialized() )
1220         {
1221             if ( DoesExistByName( windowName ) )
1222             {
1223                 return (GetWindowByName(windowName)->currentState ==
tinyWindowState_t::MAXIMIZED);
1224             }
1225             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1226             return false;
1227         }
1228         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1229         return false;
1230     }
1231 }
```

Here is the call graph for this function:



**4.2.3.29** `static bool windowManager::GetWindowIsMinimizedByIndex ( unsigned int windowIndex )` `[inline]`,  
`[static]`

Returns whether the given window is minimized

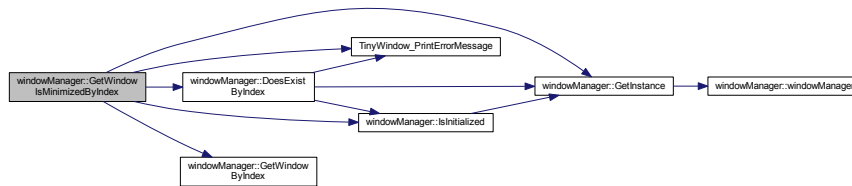
Definition at line 1161 of file TinyWindow.h.

References `windowManager::window_t::currentState`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `MINIMIZED`, `NOT_INITIALIZED`, `TINYWINDOW_ERROR`, and `TinyWindow_PrintErrorMessage()`.

```

1162     {
1163         if ( GetInstance()->IsInitialized() )
1164         {
1165             if ( DoesExistByIndex( windowIndex ) )
1166             {
1167                 return (GetWindowByIndex(windowIndex)->currentState ==
tinyWindowState_t::MINIMIZED);
1168             }
1169             return (bool)tinyWindowError_t::TINYWINDOW_ERROR;
1170         }
1171         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1172         return (bool)tinyWindowError_t::TINYWINDOW_ERROR;
1173     }
```

Here is the call graph for this function:



#### 4.2.3.30 static bool windowManager::GetWindowIsMinimizedByName ( const char \* *windowName* ) [inline], [static]

Returns whether the given window is minimized

Definition at line 1145 of file TinyWindow.h.

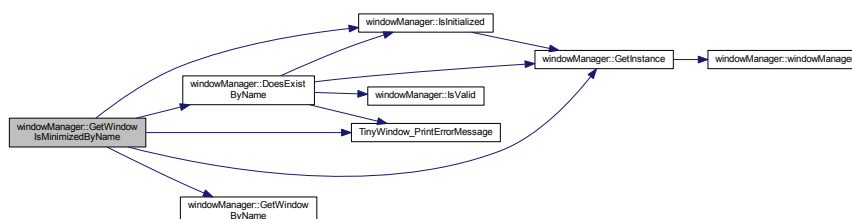
References windowManager::window\_t::currentState, DoesExistByName(), GetInstance(), GetWindowByName(), IsInitialized(), MINIMIZED, NOT\_INITIALIZED, TINYWINDOW\_ERROR, and TinyWindow\_PrintErrorMessage().

```

1146     {
1147         if ( GetInstance()->IsInitialized() )
1148         {
1149             if ( DoesExistByName( windowName ) )
1150             {
1151                 return (GetWindowByName(windowName)->currentState ==
tinyWindowState_t::MINIMIZED);
1152             }
1153             return (bool)tinyWindowError_t::TINYWINDOW_ERROR;
1154         }
1155         TinyWindow_PrintErrorMessage (
tinyWindowError_t::NOT_INITIALIZED );
1156         return (bool)tinyWindowError_t::TINYWINDOW_ERROR;
1157     }

```

Here is the call graph for this function:



**4.2.3.31** `static const char* WindowManager::GetWindowNameByIndex ( unsigned int windowIndex ) [inline], [static]`

Get window name by index

Definition at line 1292 of file TinyWindow.h.

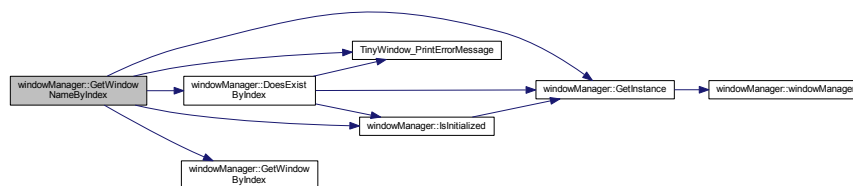
References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `WindowManager::window_t::name`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1293     {
1294         if ( GetInstance()->IsInitialized() )
1295         {
1296             if ( DoesExistByIndex( windowIndex ) )
1297             {
1298                 return GetWindowByIndex( windowIndex )->name;
1299             }
1300             TinyWindow_PrintErrorMessage(
1301 tinyWindowError_t::WINDOW_NOT_FOUND);
1302             return nullptr;
1303         }
1304         TinyWindow_PrintErrorMessage(
1305 tinyWindowError_t::NOT_INITIALIZED );
1306         return nullptr;
1307     }

```

Here is the call graph for this function:



**4.2.3.32** `static bool WindowManager::GetWindowPositionByIndex ( unsigned int windowIndex, unsigned int & x, unsigned int & y ) [inline], [static]`

Return the Position of the given window relative to screen co-ordinates by setting X and Y

Definition at line 689 of file TinyWindow.h.

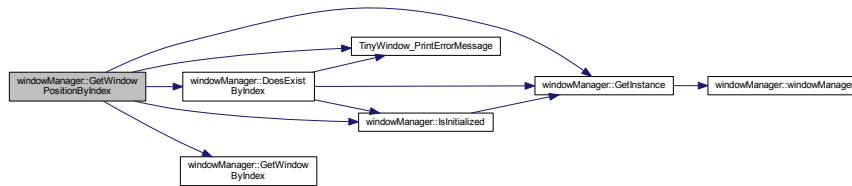
References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `WindowManager::window_t::position`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

690     {
691         if ( GetInstance()->IsInitialized() )
692         {
693             if ( DoesExistByIndex( windowIndex ) )
694             {
695                 x = GetWindowByIndex( windowIndex )->position[ 0 ];
696                 y = GetWindowByIndex( windowIndex )->position[ 1 ];
697                 return true;
698             }
699             TinyWindow_PrintErrorMessage(
700 tinyWindowError_t::WINDOW_NOT_FOUND);
701             return false;
702         }
703         TinyWindow_PrintErrorMessage(
704 tinyWindowError_t::NOT_INITIALIZED );
705         return false;
706     }

```

Here is the call graph for this function:



#### 4.2.3.33 static unsigned int\* WindowManager::GetWindowPositionByIndex ( unsigned int *windowIndex* ) [inline], [static]

Return the Position of the given window relative to screen co-ordinates as an array

Definition at line 727 of file TinyWindow.h.

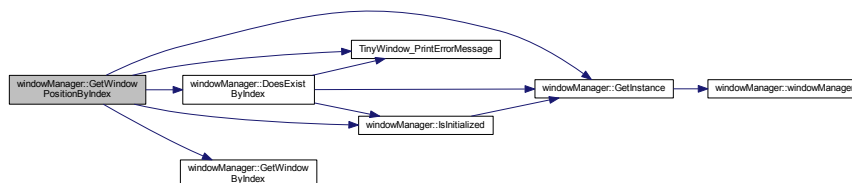
References DoesExistByIndex(), GetInstance(), GetWindowByIndex(), IsInitialized(), NOT\_INITIALIZED, WindowManager::window\_t::position, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

728 {
729     if ( GetInstance()->IsInitialized() )
730     {
731         if ( DoesExistByIndex( windowIndex ) )
732         {
733             return GetWindowByIndex( windowIndex )->position;
734         }
735         TinyWindow_PrintErrorMessage(
736         tinyWindowError_t::WINDOW_NOT_FOUND);
737         return nullptr;
738     }
739     TinyWindow_PrintErrorMessage(
740     tinyWindowError_t::NOT_INITIALIZED );
741     return nullptr;
742 }

```

Here is the call graph for this function:



#### 4.2.3.34 static bool WindowManager::GetWindowPositionByName ( const char \* *windowName*, unsigned int & x, unsigned int & y ) [inline],[static]

Return the Position of the given window relative to screen co-ordinates by setting X and Y

Definition at line 670 of file TinyWindow.h.

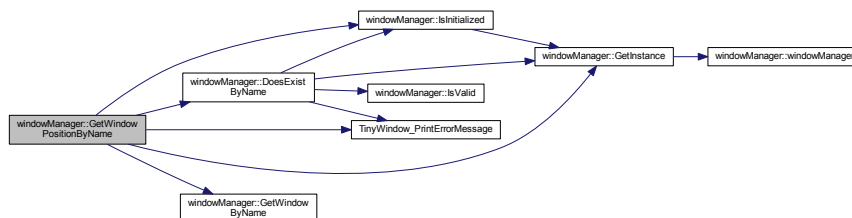
References DoesExistByName(), GetInstance(), GetWindowByName(), IsInitialized(), NOT\_INITIALIZED, WindowManager::window\_t::position, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

671     {
672         if ( GetInstance()->IsInitialized() )
673         {
674             if ( DoesExistByName( windowName ) )
675             {
676                 x = GetWindowByName( windowName )->position[ 0 ];
677                 y = GetWindowByName( windowName )->position[ 1 ];
678                 return true;
679             }
680             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
681             return false;
682         }
683         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
684         return false;
685     }

```

Here is the call graph for this function:



**4.2.3.35** `static unsigned int* windowManager::GetWindowPositionByName ( const char * windowName )` [inline], [static]

Return the Position of the given window relative to screen co-ordinates as an array

Definition at line 709 of file TinyWindow.h.

References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `windowManager::window_t::position`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

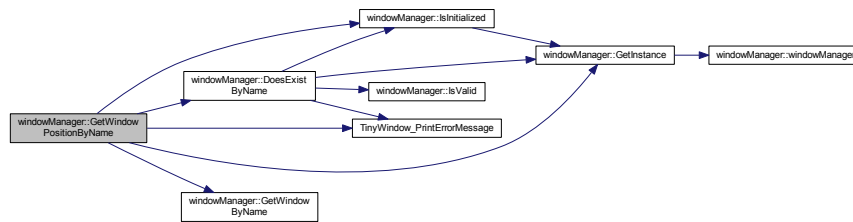
```

710     {
711         if ( GetInstance()->IsInitialized() )
712         {
713             if ( DoesExistByName( windowName ) )
714             {
715                 return GetWindowByName( windowName )->position;
716             }
717             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
718             return nullptr;
719         }
720         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
721         return nullptr;
722     }
723 }

```



Here is the call graph for this function:



#### 4.2.3.36 static bool WindowManager::GetWindowResolutionByIndex ( unsigned int *windowIndex*, unsigned int & *width*, unsigned int & *height* ) [inline], [static]

Return the Resolution of the given window by setting width and height

Definition at line 565 of file TinyWindow.h.

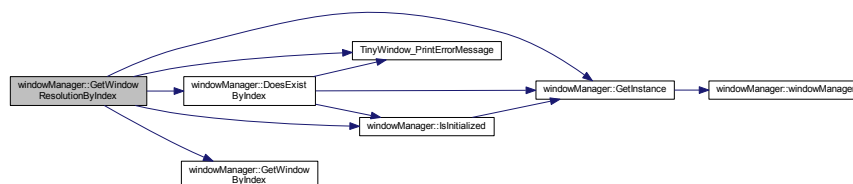
References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `windowManager::window_t::resolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

566     {
567         if ( GetInstance()->IsInitialized() )
568         {
569             if ( DoesExistByIndex( windowIndex ) )
570             {
571                 width = GetWindowByIndex( windowIndex )->
572 resolution[ 0 ];
573                 height = GetWindowByIndex( windowIndex )->
574 resolution[ 1 ];
575                 return true;
576             }
577             TinyWindow_PrintErrorMessage(
578 tinyWindowError_t::WINDOW_NOT_FOUND);
579             return false;
580         }
581         TinyWindow_PrintErrorMessage(
582 tinyWindowError_t::NOT_INITIALIZED );
583         return false;
584     }

```

Here is the call graph for this function:



#### 4.2.3.37 static unsigned int\* windowManager::GetWindowResolutionByIndex ( unsigned int *windowIndex* ) [inline], [static]

Return the Resolution of the Given Window as an array of doubles

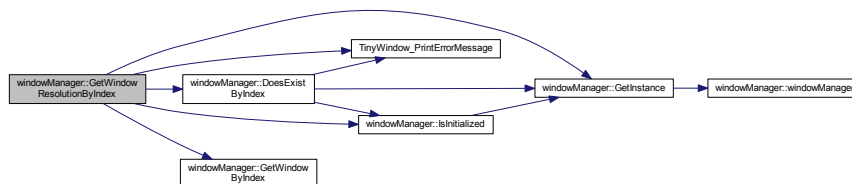
Definition at line 605 of file TinyWindow.h.

References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `windowManager::window_t::resolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

606     {
607         if ( GetInstance()->IsInitialized() )
608         {
609             if ( DoesExistByIndex( windowIndex ) )
610             {
611                 return GetWindowByIndex( windowIndex )->
resolution;
612             }
613             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
614             return nullptr;
615         }
616         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
617         return nullptr;
618     }
619 
```

Here is the call graph for this function:



#### 4.2.3.38 static bool windowManager::GetWindowResolutionByName ( const char \* *windowName*, unsigned int & *width*, unsigned int & *height* ) [inline], [static]

Return the Resolution of the given window by setting width and height

Definition at line 546 of file TinyWindow.h.

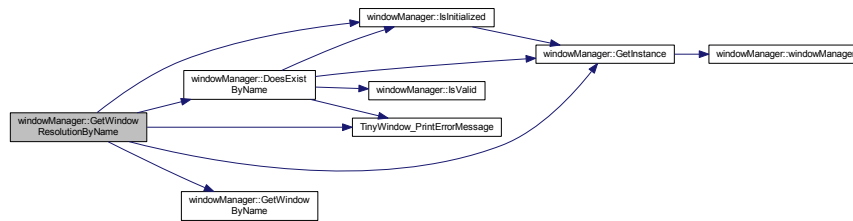
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `windowManager::window_t::resolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

547     {
548         if ( GetInstance()->IsInitialized() )
549         {
550             if ( DoesExistByName( windowName ) )
551             {
552                 width = GetWindowByName( windowName )->resolution[ 0 ];
553                 height = GetWindowByName( windowName )->
resolution[ 1 ];
554                 return false;
555             }
556             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
557             return false;
558         }
559         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
560         return false;
561     }

```

Here is the call graph for this function:



**4.2.3.39** `static unsigned int* WindowManager::GetWindowResolutionByName ( const char * windowName )` `[inline]`, `[static]`

Return the Resolution of the given Window as an array of doubles

Definition at line 587 of file TinyWindow.h.

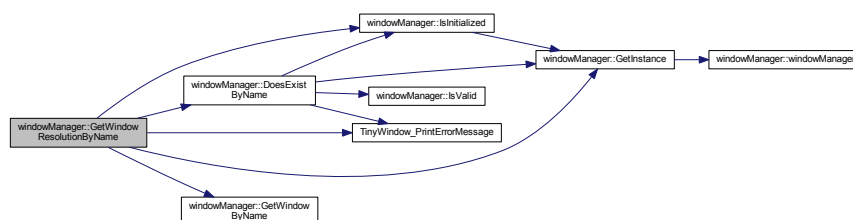
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `windowManager::window_t::resolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

588 {
589     if ( GetInstance()->IsInitialized() )
590     {
591         if ( DoesExistByName( windowName ) )
592         {
593             return GetWindowByName( windowName )->resolution;
594         }
595         TinyWindow_PrintErrorMessage(
596         tinyWindowError_t::WINDOW_NOT_FOUND);
596         return nullptr;
597     }
598
599     TinyWindow_PrintErrorMessage(
600     tinyWindowError_t::NOT_INITIALIZED );
600     return nullptr;
601 }

```

Here is the call graph for this function:



**4.2.3.40** `static bool WindowManager::GetWindowShouldCloseByIndex ( unsigned int windowIndex )` `[inline]`,  
`[static]`

Return whether the given window should be closing

Definition at line 965 of file TinyWindow.h.

References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `WindowManager::window_t::shouldClose`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

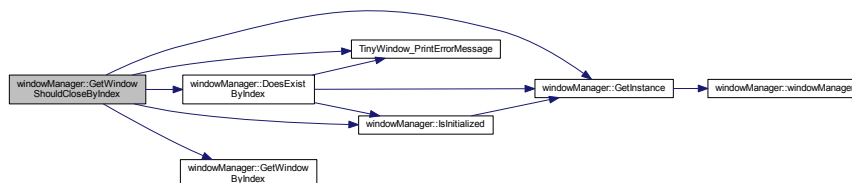
Referenced by `main()`.

```

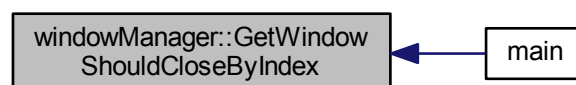
966     {
967         if ( GetInstance()->IsInitialized() )
968         {
969             if ( DoesExistByIndex( windowIndex ) )
970             {
971                 return GetWindowByIndex( windowIndex )->
shouldClose;
972             }
973             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
974             return false;
975         }
976
977         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
978         return false;
979     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.41** `static bool WindowManager::GetWindowShouldCloseByName ( const char * windowName ) [inline], [static]`

Return whether the given window should be closing

Definition at line 947 of file TinyWindow.h.

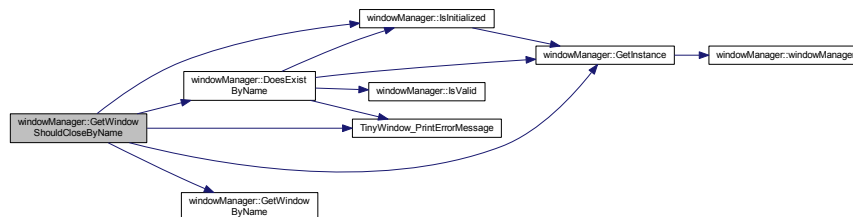
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `WindowManager::window_t::shouldClose`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

948     {
949         if ( GetInstance()->IsInitialized() )
950         {
951             if ( DoesExistByName( windowName ) )
952             {
953                 return GetWindowByName( windowName )->shouldClose;
954             }
955             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
956             return false;
957         }
958
959         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
960         return false;
961     }

```

Here is the call graph for this function:



**4.2.3.42** `static bool WindowManager::Initialize ( void ) [inline], [static]`

Initialize the window manager

Definition at line 1552 of file TinyWindow.h.

References `GetInstance()`, `instance`, `isInitialized`, `screenResolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOWS_CANNOT_INITIALIZE`.

Referenced by `main()`.

```

1553     {
1554         GetInstance()->isInitialized = false;
1555         #if defined( _WIN32 ) || defined( _WIN64 )
1556             CreateTerminal();
1557             RECT desktop;
1558
1559             HWND desktopHandle = GetDesktopWindow();
1560
1561             if (desktopHandle)
1562             {
1563                 GetWindowRect( desktopHandle, &desktop);

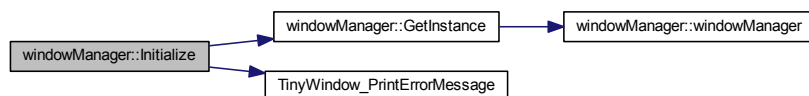
```

```

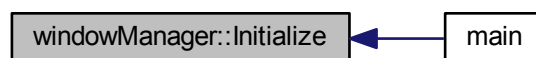
1564
1565         instance->screenResolution[0] = desktop.right;
1566         instance->screenResolution[1] = desktop.bottom;
1567         instance->isInitialized = true;
1568         return true;
1569     }
1570
1571     TinyWindow_PrintErrorMessage(
1572         tinyWindowError_t::WINDOWS_CANNOT_INITIALIZE);
1573     return false;
1574 #elif defined(__linux__)
1575     instance->currentDisplay = XOpenDisplay(0);
1576     if (!instance->currentDisplay)
1577     {
1578         TinyWindow_PrintErrorMessage(
1579             tinyWindowError_t::LINUX_CANNOT_CONNECT_X_SERVER);
1580         return false;
1581     }
1582     instance->screenResolution[0] = WidthOfScreen(
1583         XScreenOfDisplay(instance->currentDisplay,
1584             DefaultScreen(instance->currentDisplay)));
1585
1586     instance->screenResolution[1] = HeightOfScreen(
1587         XScreenOfDisplay(instance->currentDisplay,
1588             DefaultScreen(instance->currentDisplay)));
1589
1590     instance->isInitialized = true;
1591     return true;
1592 #endif
1593     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.3.43 static bool windowManager::IsInitialized ( void ) [inline],[static]

Return whether the window manager has been initialized

Definition at line 1598 of file TinyWindow.h.

References `GetInstance()`, and `isInitialized`.

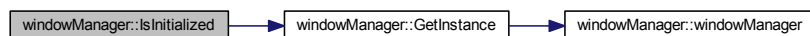
Referenced by AddWindow(), DisableWindowDecoratorByIndex(), DisableWindowDecoratorByName(), DoesExistByIndex(), DoesExistByName(), EnableWindowDecoratorsByIndex(), EnableWindowDecoratorsByName(), FocusWindowByIndex(), FocusWindowByName(), GetMousePositionInScreen(), GetMousePositionInWindowByIndex(), GetMousePositionInWindowByName(), GetNumWindows(), GetScreenResolution(), GetWindowIndexByName(), GetWindowsFullScreenByIndex(), GetWindowsFullScreenByName(), GetWindowsInFocusByIndex(), GetWindowsInFocusByName(), GetWindowsMaximizedByIndex(), GetWindowsMaximizedByName(), GetWindowsMinimizedByIndex(), GetWindowsMinimizedByName(), GetWindowNameByIndex(), GetWindowPositionByIndex(), GetWindowPositionByName(), GetWindowResolutionByIndex(), GetWindowResolutionByName(), GetWindowShouldCloseByIndex(), GetWindowShouldCloseByName(), MakeWindowCurrentContextByIndex(), MakeWindowCurrentContextByName(), MaximizeWindowByIndex(), MaximizeWindowByName(), MinimizeWindowByIndex(), MinimizeWindowByName(), PollForEvents(), RemoveWindowByIndex(), RemoveWindowByName(), RestoreWindowByIndex(), RestoreWindowByName(), SetFullScreenByIndex(), SetFullScreenByName(), SetMousePositionInScreen(), SetMousePositionInWindowByIndex(), SetMousePositionInWindowByName(), SetWindowOnDestroyedByIndex(), SetWindowOnDestroyedByName(), SetWindowOnFocusByIndex(), SetWindowOnFocusByName(), SetWindowOnKeyEventByIndex(), SetWindowOnKeyEventByName(), SetWindowOnMaximizedByIndex(), SetWindowOnMaximizedByName(), SetWindowOnMinimizedByIndex(), SetWindowOnMinimizedByName(), SetWindowOnMouseEventByIndex(), SetWindowOnMouseEventByName(), SetWindowOnMouseMoveByIndex(), SetWindowOnMouseMoveByName(), SetWindowOnMouseWheelEventByIndex(), SetWindowOnMouseWheelEventByName(), SetWindowOnMovedByIndex(), SetWindowOnMovedByName(), SetWindowOnResizeByIndex(), SetWindowOnResizeByName(), SetWindowPositionByIndex(), SetWindowPositionByName(), SetWindowResolutionByIndex(), SetWindowResolutionByName(), SetWindowStyleByIndex(), SetWindowStyleByName(), SetWindowTitleBarByIndex(), SetWindowTitleBarByName(), ShutDown(), WaitForEvents(), WindowGetKeyByIndex(), WindowGetKeyByName(), WindowSwapBuffersByIndex(), and WindowSwapBuffersByName().

```

1599     {
1600         return GetInstance()->isInitialized;
1601     }

```

Here is the call graph for this function:



#### 4.2.3.44 static bool windowManager::IsValid ( const char \* *stringParameter* ) [inline],[static],[private]

Definition at line 2421 of file TinyWindow.h.

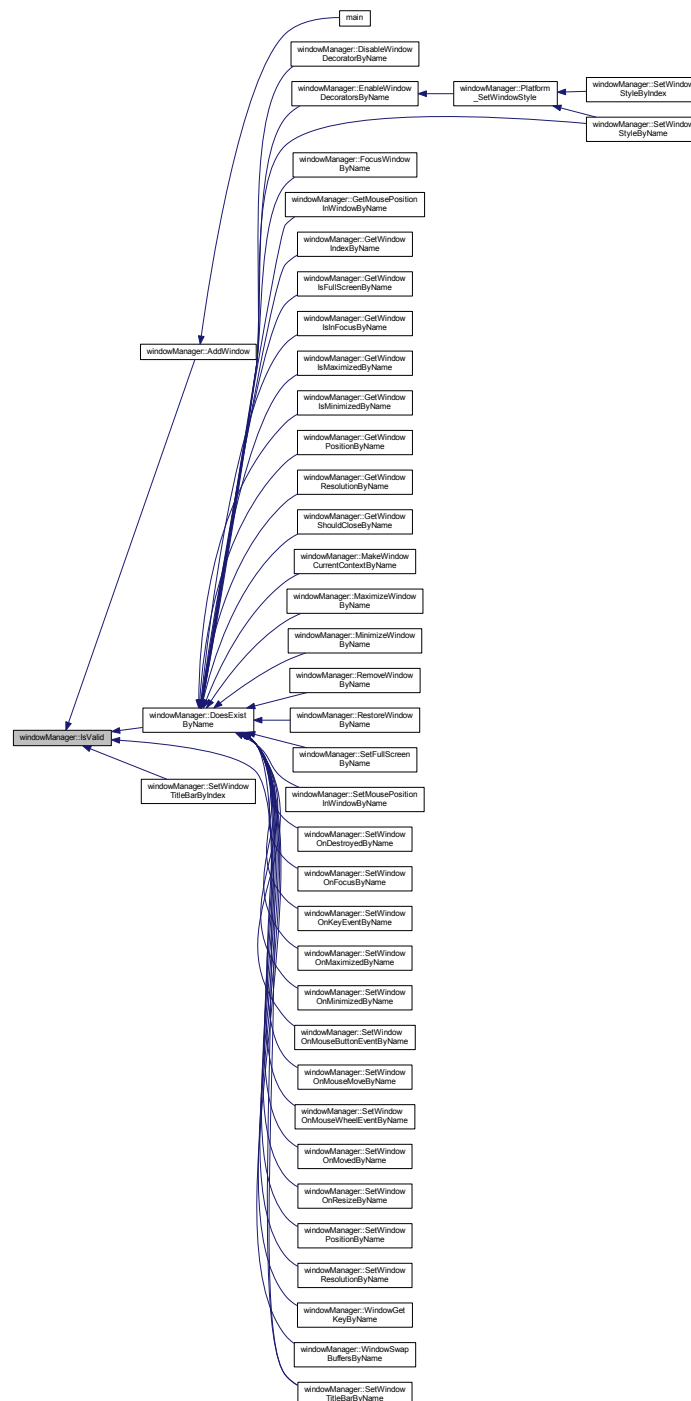
Referenced by AddWindow(), DoesExistByName(), SetWindowTitleBarByIndex(), and SetWindowTitleBarByName().

```

2422     {
2423         return ( stringParameter != nullptr );
2424     }

```

Here is the caller graph for this function:



**4.2.3.45** `static bool windowManager::MakeWindowCurrentContextByIndex ( unsigned int windowIndex ) [inline], [static]`

Make the given window be the current OpenGL Context to be drawn to

Definition at line 1045 of file TinyWindow.h.



References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_MakeCurrentContext()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

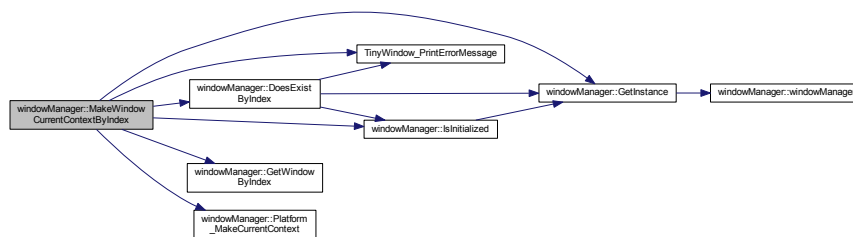
Referenced by `main()`.

```

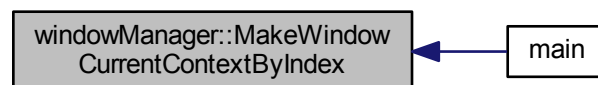
1046     {
1047         if (GetInstance()->IsInitialized())
1048         {
1049             if (DoesExistByIndex(windowIndex))
1050             {
1051                 window_t* window = GetWindowByIndex(windowIndex);
1052                 Platform_MakeCurrentContext(window);
1053                 return true;
1054             }
1055             TinyWindow_PrintErrorMessage(
1056                 tinyWindowError_t::WINDOW_NOT_FOUND);
1057             return false;
1058         }
1059         TinyWindow_PrintErrorMessage(
1060             tinyWindowError_t::NOT_INITIALIZED );
1061         return false;
1062     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.46** `static bool windowManager::MakeWindowCurrentContextByName ( const char * windowName ) [inline], [static]`

Make the given window be the current OpenGL Context to be drawn to

Definition at line 1025 of file `TinyWindow.h`.

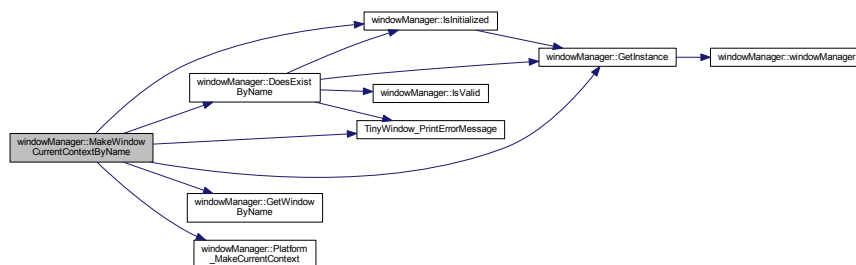
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_MakeCurrentContext()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1026     {
1027         if ( GetInstance()->IsInitialized() )
1028         {
1029             if ( DoesExistByName( windowName ) )
1030             {
1031                 window_t* window = GetWindowByName(windowName);
1032                 Platform_MakeCurrentContext(window);
1033                 return true;
1034             }
1035             TinyWindow_PrintErrorMessage(
1036 tinyWindowError_t::WINDOW_NOT_FOUND);
1037             return false;
1038         }
1039         TinyWindow_PrintErrorMessage(
1040 tinyWindowError_t::NOT_INITIALIZED );
1041         return false;
1042     }

```

Here is the call graph for this function:



**4.2.3.47** static bool windowManager::MaximizeWindowByIndex ( unsigned int *windowIndex*, bool *newState* ) [inline],  
[static]

Toggle the maximization state of the current window

Definition at line 1272 of file TinyWindow.h.

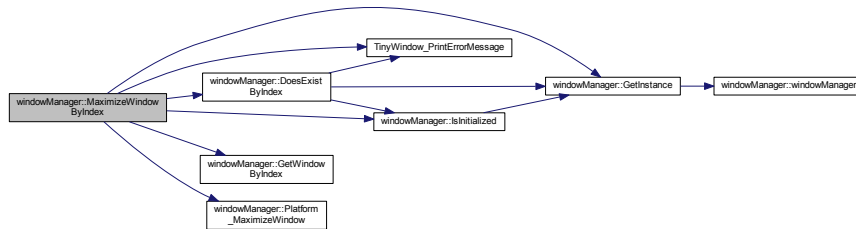
References windowManager::window\_t::AtomDesktopGeometry, DoesExistByIndex(), GetInstance(), GetWindowByIndex(), IsInitialized(), NOT\_INITIALIZED, Platform\_MaximizeWindow(), TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1273     {
1274         if (GetInstance()->IsInitialized())
1275         {
1276             if (DoesExistByIndex(windowIndex))
1277             {
1278                 window_t* window = GetWindowByIndex(windowIndex);
1279                 Platform_MaximizeWindow(window, newState);
1280                 return true;
1281             }
1282             TinyWindow_PrintErrorMessage(
1283 tinyWindowError_t::WINDOW_NOT_FOUND);
1284             return false;
1285         }
1286         TinyWindow_PrintErrorMessage(
1287 tinyWindowError_t::NOT_INITIALIZED);
1288         return false;
1289     }

```

Here is the call graph for this function:



**4.2.3.48** `static bool windowManager::MaximizeWindowByName ( const char * windowName, bool newState )` `[inline]`, `[static]`

Toggle the maximization state of the current window

Definition at line 1253 of file TinyWindow.h.

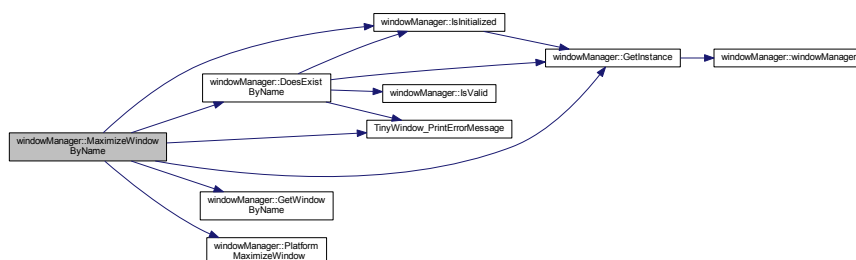
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_MaximizeWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1254     {
1255         if ( GetInstance()->IsInitialized() )
1256         {
1257             if ( DoesExistByName( windowName ) )
1258             {
1259                 window_t* window = GetWindowByName(windowName);
1260                 Platform_MaximizeWindow(window, newState);
1261                 return true;
1262             }
1263             TinyWindow_PrintErrorMessage(
1264 tinyWindowError_t::WINDOW_NOT_FOUND);
1265             return false;
1266         }
1267         TinyWindow_PrintErrorMessage(
1268 tinyWindowError_t::NOT_INITIALIZED );
1269         return false;
1270     }

```

Here is the call graph for this function:



**4.2.3.49** `static bool windowManager::MinimizeWindowByIndex ( unsigned int windowIndex, bool newState )` `[inline]`,  
`[static]`

Toggle the minimization state of the window

Definition at line 1197 of file TinyWindow.h.

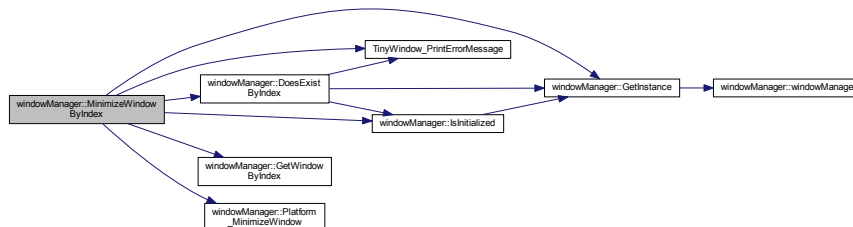
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_MinimizeWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1198     {
1199         if ( GetInstance()->IsInitialized() )
1200         {
1201             if ( DoesExistByIndex( windowIndex ) )
1202             {
1203                 window_t* window = GetWindowByIndex(windowIndex);
1204                 Platform_MinimizeWindow(window, newState);
1205                 return true;
1206             }
1207             TinyWindow_PrintErrorMessage(
1208                 tinyWindowError_t::WINDOW_NOT_FOUND);
1209             return false;
1210         }
1211         TinyWindow_PrintErrorMessage(
1212             tinyWindowError_t::NOT_INITIALIZED );
1213         return false;
1214     }

```

Here is the call graph for this function:



**4.2.3.50** `static bool windowManager::MinimizeWindowByName ( const char * windowName, bool newState )` `[inline]`,  
`[static]`

Toggle the minimization state of the given window

Definition at line 1178 of file TinyWindow.h.

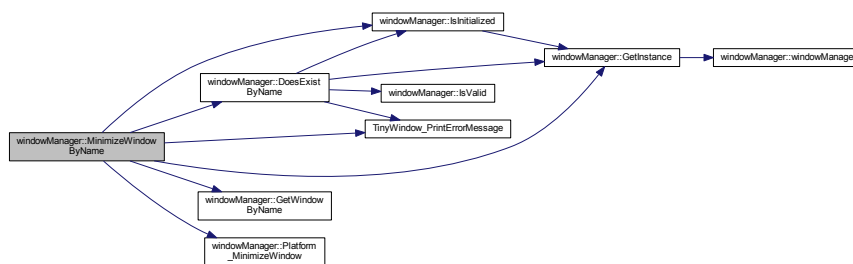
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_MinimizeWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1179     {
1180         if ( GetInstance()->IsInitialized() )
1181         {
1182             if ( DoesExistByName( windowName ) )
1183             {
1184                 window_t* window = GetWindowByName(windowName);
1185                 Platform_MinimizeWindow(window, newState);
1186                 return true;
1187             }
1188             TinyWindow_PrintErrorMessage(
1189 tinyWindowError_t::WINDOW_NOT_FOUND);
1189             return false;
1190         }
1191         TinyWindow_PrintErrorMessage(
1192 tinyWindowError_t::NOT_INITIALIZED);
1192         return false;
1193     }

```

Here is the call graph for this function:



#### 4.2.3.51 static void windowManager::Platform\_DisableWindowDecorators ( window\_t \* window, unsigned int decorators ) [inline], [static], [private]

Definition at line 2890 of file TinyWindow.h.

References windowManager::window\_t::currentWindowStyle, DECORATOR\_BORDER, DECORATOR\_CLOSEBUTTON, DECORATOR\_ICON, DECORATOR\_MAXIMIZEBUTTON, DECORATOR\_MINIMIZEBUTTON, DECORATOR\_SIZEABLEBORDER, and DECORATOR\_TITLEBAR.

Referenced by DisableWindowDecoratorByIndex(), and DisableWindowDecoratorByName().

```

2891     {
2892 #if defined( _WIN32 ) || defined( _WIN64 )
2893         if (decorators & DECORATOR_BORDER)
2894         {
2895             window->currentWindowStyle &= ~WS_BORDER;
2896         }
2897
2898         if (decorators & DECORATOR_TITLEBAR)
2899         {
2900             window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
2901         }
2902
2903         if (decorators & DECORATOR_ICON)
2904         {
2905             window->currentWindowStyle &= ~WS_ICONIC;
2906         }
2907
2908         if (decorators & DECORATOR_CLOSEBUTTON)
2909         {
2910             window->currentWindowStyle &= ~WS_SYSMENU;
2911         }
2912
2913         if (decorators & DECORATOR_MINIMIZEBUTTON)
2914         {

```

```

2915         window->currentWindowStyle &= ~WS_MINIMIZEBOX;
2916     }
2917
2918     if (decorators & DECORATOR_MAXIMIZEBUTTON)
2919     {
2920         window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
2921     }
2922
2923     if (decorators & DECORATOR_SIZEABLEBORDER)
2924     {
2925         window->currentWindowStyle &= ~WS_SIZEBOX;
2926     }
2927
2928     SetWindowLongPtr(window->windowHandle, GWL_STYLE,
2929         window->currentWindowStyle | WS_VISIBLE);
2930 #elif defined(__linux__)
2931     if (decorators & DECORATOR_CLOSEBUTTON)
2932     {
2933         //I hate doing this but it is necessary to keep functionality going.
2934         bool minimizeEnabled = false;
2935         bool maximizeEnabled = false;
2936
2937         if (decorators & DECORATOR_MAXIMIZEBUTTON)
2938         {
2939             maximizeEnabled = true;
2940         }
2941
2942         if (decorators & DECORATOR_MINIMIZEBUTTON)
2943         {
2944             minimizeEnabled = true;
2945         }
2946
2947         window->currentWindowStyle &= ~LINUX_DECORATOR_CLOSE;
2948
2949         if (maximizeEnabled)
2950         {
2951             window->currentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
2952         }
2953
2954         if (minimizeEnabled)
2955         {
2956             window->currentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
2957         }
2958
2959         window->decorators = 1;
2960     }
2961
2962     if (decorators & DECORATOR_MINIMIZEBUTTON)
2963     {
2964         window->currentWindowStyle &= ~LINUX_DECORATOR_MINIMIZE;
2965         window->decorators = 1;
2966     }
2967
2968     if (decorators & DECORATOR_MAXIMIZEBUTTON)
2969     {
2970         bool minimizeEnabled = false;
2971
2972         if (decorators & DECORATOR_MINIMIZEBUTTON)
2973         {
2974             minimizeEnabled = true;
2975         }
2976
2977         window->currentWindowStyle &= ~LINUX_DECORATOR_MAXIMIZE;
2978
2979         if (minimizeEnabled)
2980         {
2981             window->currentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
2982         }
2983
2984         window->decorators = 1;
2985     }
2986
2987     if (decorators & DECORATOR_ICON)
2988     {
2989         //Linux ( at least cinnamon ) does not have icons in the window. only in the taskbar icon
2990     }
2991
2992     //just need to set it to 1 to enable all decorators that include title bar
2993     if (decorators & DECORATOR_TITLEBAR)
2994     {
2995         window->decorators = LINUX_DECORATOR_BORDER;
2996     }
2997
2998     if (decorators & DECORATOR_BORDER)
2999     {
3000         window->decorators = 0;
3001     }

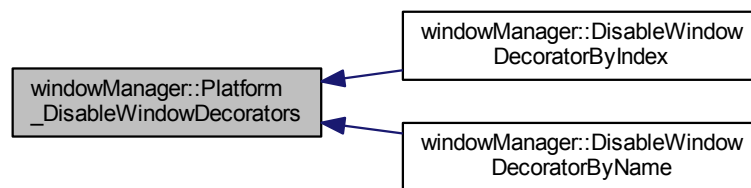
```

```

3002
3003     if (decorators & DECORATOR_SIZEABLEBORDER)
3004     {
3005         window->decorators = 0;
3006     }
3007
3008     long hints[5] = { LINUX_FUNCTION | LINUX_DECORATOR, window->
currentWindowStyle, window->decorators, 0, 0 };
3009
3010     XChangeProperty(instance->currentDisplay, window->windowHandle, window->AtomHints, XA_ATOM,
3011 32,
3012         PropModeReplace, (unsigned char*)hints, 5);
3013
3014     XMapWindow(instance->currentDisplay, window->windowHandle);
3015 #endif
3016 }

```

Here is the caller graph for this function:



#### 4.2.3.52 static void windowManager::Platform\_EnableWindowDecorators ( window\_t \* window, unsigned int decorators ) [inline], [static], [private]

Definition at line 2799 of file TinyWindow.h.

References windowManager::window\_t::currentWindowStyle, DECORATOR\_BORDER, DECORATOR\_CLOSEBUTTON, DECORATOR\_ICON, DECORATOR\_MAXIMIZEBUTTON, DECORATOR\_MINIMIZEBUTTON, DECORATOR\_SIZEABLEBORDER, and DECORATOR\_TITLEBAR.

Referenced by EnableWindowDecoratorsByIndex(), and EnableWindowDecoratorsByName().

```

2800 {
2801 #if defined( _WIN32 ) || defined( _WIN64 )
2802     window->currentWindowStyle = WS_VISIBLE | WS_CLIPSIBLINGS;
2803
2804     if (decorators & DECORATOR_BORDER)
2805     {
2806         window->currentWindowStyle |= WS_BORDER;
2807     }
2808
2809     if (decorators & DECORATOR_TITLEBAR)
2810     {
2811         window->currentWindowStyle |= WS_CAPTION;
2812     }
2813
2814     if (decorators & DECORATOR_ICON)
2815     {
2816         window->currentWindowStyle |= WS_ICONIC;
2817     }
2818
2819     if (decorators & DECORATOR_CLOSEBUTTON)
2820     {
2821         window->currentWindowStyle |= WS_SYSMENU;
2822     }
2823 }

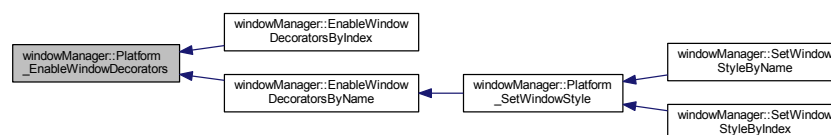
```

```

2824         if (decorators & DECORATOR_MINIMIZEBUTTON)
2825         {
2826             window->currentWindowStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
2827         }
2828
2829         if (decorators & DECORATOR_MAXIMIZEBUTTON)
2830         {
2831             window->currentWindowStyle |= WS_MAXIMIZEBOX | WS_SYSMENU;
2832         }
2833
2834         if (decorators & DECORATOR_SIZEABLEBORDER)
2835         {
2836             window->currentWindowStyle |= WS_SIZEBOX;
2837         }
2838
2839         SetWindowLongPtr(window->windowHandle, GWL_STYLE,
2840             window->currentWindowStyle);
2841 #elif defined(__linux__)
2842         if (decorators & DECORATOR_CLOSEBUTTON)
2843         {
2844             window->currentWindowStyle |= LINUX_DECORATOR_CLOSE;
2845             window->decorators = 1;
2846         }
2847
2848         if (decorators & DECORATOR_MINIMIZEBUTTON)
2849         {
2850             window->currentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
2851             window->decorators = 1;
2852         }
2853
2854         if (decorators & DECORATOR_MAXIMIZEBUTTON)
2855         {
2856             window->currentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
2857             window->decorators = 1;
2858         }
2859
2860         if (decorators & DECORATOR_ICON)
2861         {
2862             //Linux ( at least cinnamon ) does not have icons in the window. only in the task bar icon
2863         }
2864
2865         //just need to set it to 1 to enable all decorators that include title bar
2866         if (decorators & DECORATOR_TITLEBAR)
2867         {
2868             window->decorators = 1;
2869         }
2870
2871         if (decorators & DECORATOR_BORDER)
2872         {
2873             window->decorators = 1;
2874         }
2875
2876         if (decorators & DECORATOR_SIZEABLEBORDER)
2877         {
2878             window->decorators = 1;
2879         }
2880
2881         long hints[5] = { LINUX_FUNCTION | LINUX_DECORATOR, window->
currentWindowStyle, window->decorators, 0, 0 };
2882
2883         XChangeProperty(instance->currentDisplay, window->windowHandle, window->AtomHints, XA_ATOM,
32,
2884             PropModeReplace, (unsigned char*)hints, 5);
2885
2886         XMapWindow(instance->currentDisplay, window->windowHandle);
2887 #endif
2888     }

```

Here is the caller graph for this function:





**4.2.3.53** `static void WindowManager::Platform_FocusWindow ( window_t * window, bool newState ) [inline], [static], [private]`

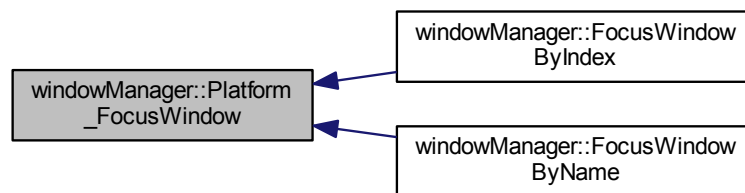
Definition at line 2686 of file TinyWindow.h.

Referenced by FocusWindowByIndex(), and FocusWindowByName().

```

2687     {
2688         if (newState)
2689         {
2690             #if defined( _WIN32 ) || defined( _WIN64 )
2691                 SetFocus(window->windowHandle);
2692             #elif defined(__linux__)
2693                 XMapWindow(instance->currentDisplay, window->windowHandle);
2694             #endif
2695         }
2696         else
2697         {
2698             #if defined( _WIN32 ) || defined( _WIN64 )
2699                 SetFocus(nullptr);
2700             #elif defined(__linux__)
2701                 XUnmapWindow(instance->currentDisplay, window->windowHandle);
2702             #endif
2703         }
2704     }
2705 }
```

Here is the caller graph for this function:



**4.2.3.54** `static bool WindowManager::Platform_InitializeGL ( window_t * window ) [inline], [static], [private]`

Definition at line 2455 of file TinyWindow.h.

References `windowManager::window_t::contextCreated`, `INVALID_CONTEXT`, and `TinyWindow_PrintErrorMessage()`.

```

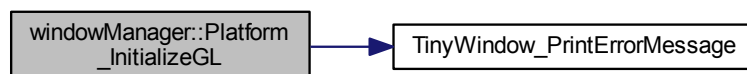
2456     {
2457         #if defined( _WIN32 ) || defined( _WIN64 )
2458             window->deviceContextHandle = GetDC(window->windowHandle);
2459             InitializePixelFormat(window);
2460             window->glRenderingContextHandle = wglCreateContext(window->deviceContextHandle);
2461             wglMakeCurrent(window->deviceContextHandle, window->glRenderingContextHandle);
2462
2463             window->contextCreated = (window->glRenderingContextHandle != nullptr);
2464
2465             if (window->contextCreated)
2466             {
2467                 return true;
2468             }
2469         }
```

```

2469
2470     TinyWindow_PrintErrorMessage(
2471         tinyWindowError_t::INVALID_CONTEXT);
2472     return false;
2473 #elif defined(__linux__)
2474     if (!window->context)
2475     {
2476         window->context = glXCreateContext(
2477             instance->currentDisplay,
2478             window->visualInfo,
2479             0,
2480             true);
2481
2482         if (window->context)
2483         {
2484             glXMakeCurrent(instance->currentDisplay,
2485                 window->windowHandle,
2486                 window->context);
2487
2488             XWindowAttributes l_Attributes;
2489
2490             XGetWindowAttributes(instance->currentDisplay,
2491                 window->windowHandle, &l_Attributes);
2492             window->position[0] = l_Attributes.x;
2493             window->position[1] = l_Attributes.y;
2494
2495             window->contextCreated = true;
2496             return true;
2497         }
2498         return false;
2499     }
2500     else
2501     {
2502         TinyWindow_PrintErrorMessage(
2503             tinyWindowError_t::EXISTING_CONTEXT);
2504         return false;
2505     }
2506 #endif
2507 }

```

Here is the call graph for this function:



**4.2.3.55** static void windowManager::Platform\_InitializeWindow ( window\_t \* window ) [inline], [static], [private]

Definition at line 2446 of file TinyWindow.h.

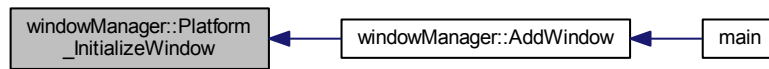
Referenced by AddWindow().

```

2447     {
2448 #if defined( _WIN32 ) || defined( _WIN64 )
2449         Windows_InitializeWindow( window );
2450 #elif defined(__linux__)
2451         Linux_InitializeWindow( window );
2452 #endif
2453     }

```

Here is the caller graph for this function:



**4.2.3.56** `static void windowManager::Platform_MakeCurrentContext ( window_t * window ) [inline], [static], [private]`

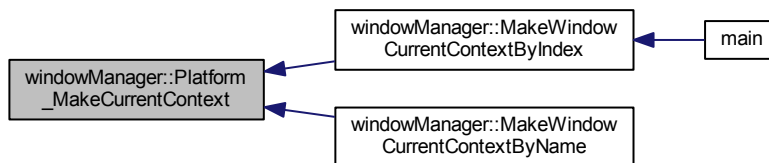
Definition at line 2567 of file TinyWindow.h.

Referenced by `MakeWindowCurrentContextByIndex()`, and `MakeWindowCurrentContextByName()`.

```

2568     {
2569     #if defined( _WIN32 ) || defined( _WIN64 )
2570         wglMakeCurrent( window->deviceContextHandle,
2571             window->glRenderingContextHandle );
2572     #elif defined( __linux__ )
2573         glXMakeCurrent( instance->currentDisplay, window->windowHandle,
2574             window->context );
2575     #endif
2576     }
  
```

Here is the caller graph for this function:



**4.2.3.57** `static void windowManager::Platform_MaximizeWindow ( window_t * window, bool newState ) [inline], [static], [private]`

Definition at line 2628 of file TinyWindow.h.

References `windowManager::window_t::currentState`, `MAXIMIZED`, and `NORMAL`.

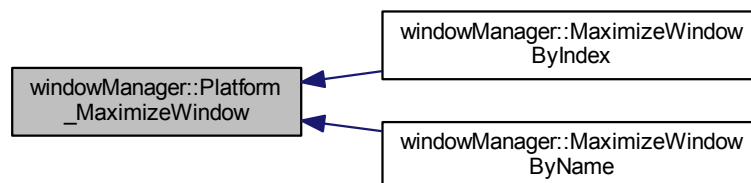
Referenced by `MaximizeWindowByIndex()`, and `MaximizeWindowByName()`.

```

2629     {
2630         if (newState)
2631         {
2632             window->currentState = tinyWindowState_t::MAXIMIZED;
2633 #if defined( _WIN32 ) || defined( _WIN64 )
2634             ShowWindow(window->windowHandle, SW_MAXIMIZE);
2635 #elif defined(__linux__)
2636             XEvent currentEvent;
2637             memset(&currentEvent, 0, sizeof(currentEvent));
2638
2639             currentEvent.xany.type = ClientMessage;
2640             currentEvent.xclient.message_type = window->AtomState;
2641             currentEvent.xclient.format = 32;
2642             currentEvent.xclient.window = window->windowHandle;
2643             currentEvent.xclient.data.l[0] = (window->currentState ==
tinyWindowState_t::MAXIMIZED);
2644             currentEvent.xclient.data.l[1] = window->AtomMaxVert;
2645             currentEvent.xclient.data.l[2] = window->AtomMaxHorz;
2646
2647             XSendEvent(instance->currentDisplay,
2648                 XDefaultRootWindow(instance->currentDisplay),
2649                 0, SubstructureNotifyMask, &currentEvent);
2650 #endif
2651         }
2652         else
2653         {
2654             window->currentState = tinyWindowState_t::NORMAL;
2655 #if defined( _WIN32 ) || defined( _WIN64 )
2656             ShowWindow(window->windowHandle, SW_RESTORE);
2657 #elif defined(__linux__)
2658             XEvent currentEvent;
2659             memset(&currentEvent, 0, sizeof(currentEvent));
2660
2661             currentEvent.xany.type = ClientMessage;
2662             currentEvent.xclient.message_type = window->AtomState;
2663             currentEvent.xclient.format = 32;
2664             currentEvent.xclient.window = window->windowHandle;
2665             currentEvent.xclient.data.l[0] = (window->currentState ==
tinyWindowState_t::MAXIMIZED);
2666             currentEvent.xclient.data.l[1] = window->AtomMaxVert;
2667             currentEvent.xclient.data.l[2] = window->AtomMaxHorz;
2668
2669             XSendEvent(instance->currentDisplay,
2670                 XDefaultRootWindow(instance->currentDisplay),
2671                 0, SubstructureNotifyMask, &currentEvent);
2672 #endif
2673         }
2674     }
2675 }

```

Here is the caller graph for this function:



**4.2.3.58** `static void windowManager::Platform_MinimizeWindow ( window_t * window, bool newState )` [inline], [static], [private]

Definition at line 2603 of file TinyWindow.h.

References `windowManager::window_t::currentState`, `MINIMIZED`, and `NORMAL`.

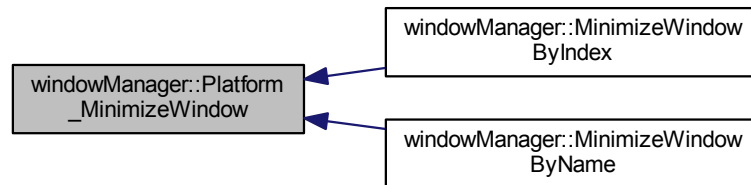
Referenced by `MinimizeWindowByIndex()`, and `MinimizeWindowByName()`.

```

2604     {
2605         if (newState)
2606         {
2607             window->currentState = tinyWindowState_t::MINIMIZED;
2608
2609 #if defined( _WIN32 ) || defined( _WIN64 )
2610             ShowWindow(window->windowHandle, SW_MINIMIZE);
2611 #elif defined(__linux__)
2612             XIconifyWindow(instance->currentDisplay,
2613                             window->windowHandle, 0);
2614 #endif
2615         }
2616     else
2617     {
2618         window->currentState = tinyWindowState_t::NORMAL;
2619 #if defined( _WIN32 ) || defined( _WIN64 )
2620         ShowWindow(window->windowHandle, SW_RESTORE);
2621 #elif defined(__linux__)
2622         XMapWindow(instance->currentDisplay, window->windowHandle);
2623 #endif
2624     }
2625 }
2626

```

Here is the caller graph for this function:



**4.2.3.59** static void windowManager::Platform\_RestoreWindow ( window\_t\* window ) [inline], [static], [private]

Definition at line 2707 of file TinyWindow.h.

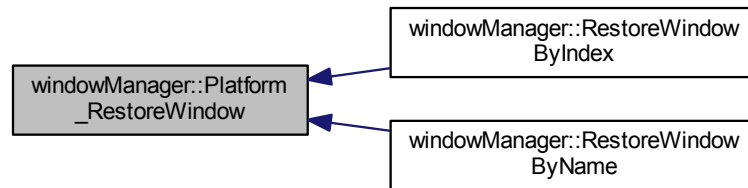
Referenced by RestoreWindowByIndex(), and RestoreWindowByName().

```

2708     {
2709 #if defined( _WIN32 ) || defined( _WIN64 )
2710         ShowWindow(window->windowHandle, SW_RESTORE);
2711 #elif defined(__linux__)
2712         XMapWindow(instance->currentDisplay, window->windowHandle);
2713 #endif
2714     }

```

Here is the caller graph for this function:



**4.2.3.60** `static void windowManager::Platform_SetFullScreen ( window_t * window )` `[inline], [static], [private]`

Definition at line 2578 of file TinyWindow.h.

References `GetScreenResolution()`.

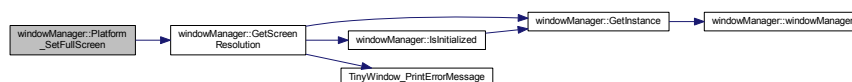
Referenced by `SetFullScreenByIndex()`, and `SetFullScreenByName()`.

```

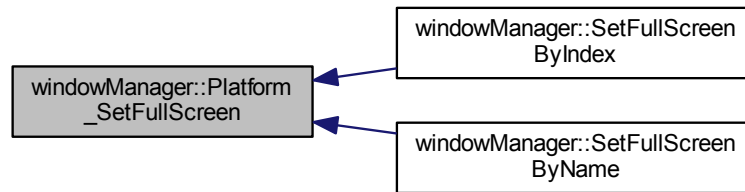
2579     {
2580 #if defined( _WIN32 ) || defined( _WIN64 )
2581         SetWindowLongPtr(window->windowHandle, GWL_STYLE,
2582             WS_SYSMENU | WS_POPUP | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_VISIBLE);
2583
2584         MoveWindow(window->windowHandle, 0, 0,
2585             windowManager::GetScreenResolution()[0],
2586             windowManager::GetScreenResolution()[1], true);
2587 #elif defined(__linux__)
2588         XEvent currentEvent;
2589         memset(&currentEvent, 0, sizeof(currentEvent));
2590
2591         currentEvent.xany.type = ClientMessage;
2592         currentEvent.xclient.message_type = window->AtomState;
2593         currentEvent.xclient.format = 32;
2594         currentEvent.xclient.window = window->windowHandle;
2595         currentEvent.xclient.data.l[0] = window->currentState ==
2596             tinyWindowState_t::FULLSCREEN;
2597         currentEvent.xclient.data.l[1] = window->AtomFullScreen;
2598
2599         XSendEvent(instance->currentDisplay,
2600             XDefaultRootWindow(instance->currentDisplay),
2601             0, SubstructureNotifyMask, &currentEvent);
2602 #endif
2603     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.61** `static void windowManager::Platform_SetMousePositionInWindow ( window_t * window, unsigned int x, unsigned int y )` `[inline]`, `[static]`, `[private]`

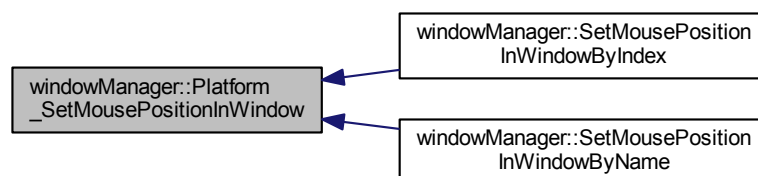
Definition at line 2540 of file `TinyWindow.h`.

Referenced by `SetMousePositionInWindowByIndex()`, and `SetMousePositionInWindowByName()`.

```

2541     {
2542     #if defined( _WIN32 ) || defined( _WIN64 )
2543         POINT mousePoint;
2544         mousePoint.x = x;
2545         mousePoint.y = y;
2546         ScreenToClient(window->windowHandle, &mousePoint);
2547         SetCursorPos(mousePoint.x, mousePoint.y);
2548     #elif defined(__linux__)
2549         XWarpPointer(
2550             windowManager::GetDisplay(),
2551             window->windowHandle, window->windowHandle,
2552             window->position[0], window->position[1],
2553             window->resolution[0], window->resolution[1],
2554             x, y);
2555     #endif
2556     }
  
```

Here is the caller graph for this function:



**4.2.3.62** `static void windowManager::Platform_SetWindowPosition ( window_t * window, unsigned int x, unsigned int y )`  
`[inline],[static],[private]`

Definition at line 2522 of file TinyWindow.h.

References `windowManager::window_t::resolution`.

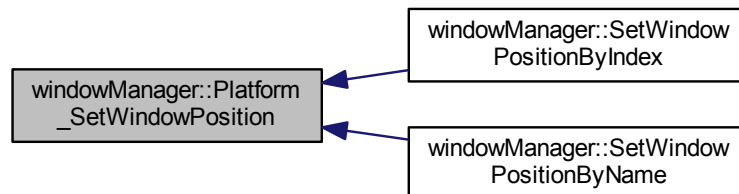
Referenced by `SetWindowPositionByIndex()`, and `SetWindowPositionByName()`.

```

2523     {
2524     #if defined( _WIN32 ) || defined( _WIN64 )
2525         SetWindowPos(window->windowHandle, HWND_TOP, x, y,
2526             window->resolution[0], window->resolution[1],
2527             SWP_SHOWWINDOW | SWP_NOSIZE);
2528     #elif defined(__linux__)
2529         XWindowChanges windowChanges;
2530
2531         windowChanges.x = x;
2532         windowChanges.y = y;
2533
2534         XConfigureWindow(
2535             instance->currentDisplay,
2536             window->windowHandle, CWX | CWY, &windowChanges);
2537     #endif
2538     }

```

Here is the caller graph for this function:



**4.2.3.63** `static void windowManager::Platform_SetWindowResolution ( window_t * window )` `[inline],[static],[private]`

Definition at line 2509 of file TinyWindow.h.

References `windowManager::window_t::position`, and `windowManager::window_t::resolution`.

Referenced by `SetWindowResolutionByIndex()`, and `SetWindowResolutionByName()`.

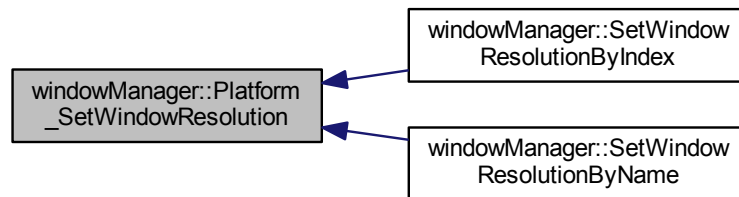
```

2510     {
2511     #if defined( _WIN32 ) || defined( _WIN64 )
2512         SetWindowPos(window->windowHandle, HWND_TOP,
2513             window->position[0], window->position[1],
2514             window->resolution[0], window->resolution[1],
2515             SWP_SHOWWINDOW | SWP_NOMOVE);
2516     #elif defined(__linux__)
2517         XResizeWindow(instance->currentDisplay,
2518             window->windowHandle, window->resolution[0], window->resolution[1]);
2519     #endif
2520     }

```



Here is the caller graph for this function:



**4.2.3.64** `static void windowManager::Platform_SetWindowStyle ( window_t * window, tinyWindowStyle_t windowStyle )`  
`[inline], [static], [private]`

Definition at line 2716 of file TinyWindow.h.

References BARE, DECORATOR\_BORDER, DECORATOR\_CLOSEBUTTON, DECORATOR\_MAXIMIZEBUTTON, DECORATOR\_MINIMIZEBUTTON, DECORATOR\_TITLEBAR, DEFAULT, EnableWindowDecoratorsByName(), INVALID\_WINDOWSTYLE, windowManager::window\_t::name, POPUP, and TinyWindow\_PrintErrorMessage().

Referenced by SetWindowStyleByIndex(), and SetWindowStyleByName().

```

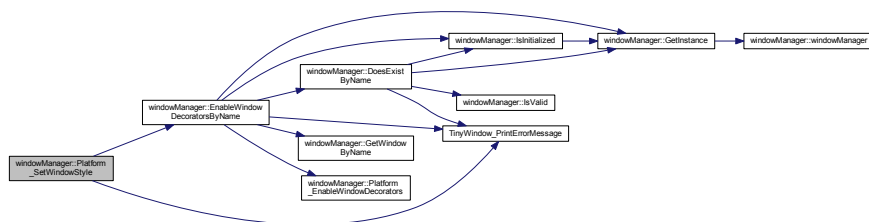
2717     {
2718     #if defined( _WIN32 ) || defined( _WIN64 )
2719         switch (windowStyle)
2720         {
2721             case tinyWindowStyle_t::DEFAULT:
2722             {
2723                 EnableWindowDecoratorsByName(window->name,
2724 DECORATOR_TITLEBAR | DECORATOR_BORDER |
2725                 DECORATOR_CLOSEBUTTON |
2726 DECORATOR_MINIMIZEBUTTON | DECORATOR_MAXIMIZEBUTTON);
2727                 break;
2728             }
2729             case tinyWindowStyle_t::POPUP:
2730             {
2731                 EnableWindowDecoratorsByName(window->name, 0);
2732                 break;
2733             }
2734             case tinyWindowStyle_t::BARE:
2735             {
2736                 EnableWindowDecoratorsByName(window->name,
2737 DECORATOR_TITLEBAR | DECORATOR_BORDER);
2738                 break;
2739             }
2740             default:
2741             {
2742                 TinyWindow_PrintErrorMessage(
2743 tinyWindowError_t::INVALID_WINDOWSTYLE);
2744                 break;
2745             }
2746         }
2747     #elif defined(__linux__)
2748         switch (windowStyle)
2749         {
2750             case tinyWindowStyle_t::DEFAULT:
2751             {
2752                 window->decorators = (1L << 2);
  
```

```

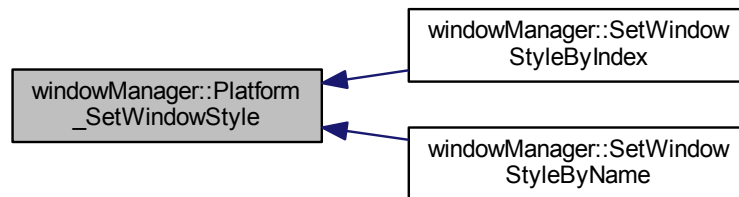
2753         window->currentWindowStyle = LINUX_DECORATOR_MOVE | LINUX_DECORATOR_CLOSE |
2754             LINUX_DECORATOR_MAXIMIZE | LINUX_DECORATOR_MINIMIZE;
2755         long Hints[5] = { LINUX_FUNCTION | LINUX_DECORATOR, window->
currentWindowStyle, window->decorators, 0, 0 };
2756
2757         XChangeProperty(instance->currentDisplay, window->windowHandle, window->AtomHints,
XA_ATOM, 32, PropModeReplace,
2758             (unsigned char*)Hints, 5);
2759
2760         XMapWindow(instance->currentDisplay, window->windowHandle);
2761         break;
2762     }
2763
2764     case tinyWindowStyle_t::BARE:
2765     {
2766         window->decorators = (1L << 2);
2767         window->currentWindowStyle = (1L << 2);
2768         long Hints[5] = { LINUX_FUNCTION | LINUX_DECORATOR, window->
currentWindowStyle, window->decorators, 0, 0 };
2769
2770         XChangeProperty(instance->currentDisplay, window->windowHandle, window->AtomHints,
XA_ATOM, 32, PropModeReplace,
2771             (unsigned char*)Hints, 5);
2772
2773         XMapWindow(instance->currentDisplay, window->windowHandle);
2774         break;
2775     }
2776
2777     case tinyWindowStyle_t::POPUP:
2778     {
2779         window->decorators = 0;
2780         window->currentWindowStyle = (1L << 2);
2781         long Hints[5] = { LINUX_FUNCTION | LINUX_DECORATOR, window->
currentWindowStyle, window->decorators, 0, 0 };
2782
2783         XChangeProperty(instance->currentDisplay, window->windowHandle, window->AtomHints,
XA_ATOM, 32, PropModeReplace,
2784             (unsigned char*)Hints, 5);
2785
2786         XMapWindow(instance->currentDisplay, window->windowHandle);
2787         break;
2788     }
2789
2790     default:
2791     {
2792         TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_WINDOWSTYLE);
2793         break;
2794     }
2795 }
2796 #endif
2797 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.65** `static void windowManager::Platform_SetWindowTitleBar ( window_t * window, const char * newTitle )`  
`[inline], [static], [private]`

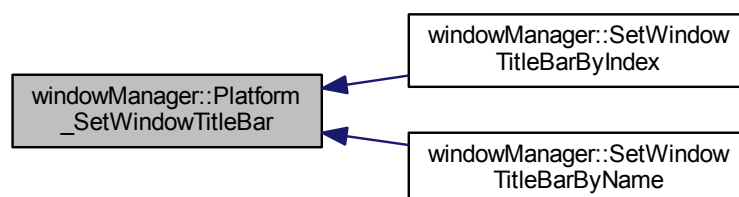
Definition at line 2677 of file TinyWindow.h.

Referenced by SetWindowTitleBarByIndex(), and SetWindowTitleBarByName().

```

2678     {
2679     #if defined( _WIN32 ) || defined( _WIN64 )
2680         SetWindowText( window->windowHandle, newTitle);
2681     #elif defined( __linux__ )
2682         XStoreName( instance->currentDisplay, window->windowHandle, newTitle);
2683     #endif
2684     }
  
```

Here is the caller graph for this function:



**4.2.3.66** `static void windowManager::Platform_SwapBuffers ( window_t * window )` `[inline], [static], [private]`

Definition at line 2558 of file TinyWindow.h.

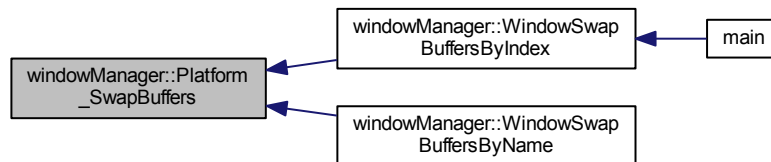
Referenced by WindowSwapBuffersByIndex(), and WindowSwapBuffersByName().

```

2559     {
2560     #if defined( _WIN32 ) || defined( _WIN64 )
2561         SwapBuffers(window->deviceContextHandle);
2562     #elif defined(__linux__)
2563         glXSwapBuffers(instance->currentDisplay, window->windowHandle);
2564     #endif
2565     }

```

Here is the caller graph for this function:



#### 4.2.3.67 static void windowManager::PollForEvents ( void ) [inline],[static]

Ask the window manager to poll for events

Definition at line 1606 of file TinyWindow.h.

References `GetInstance()`, `instance`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

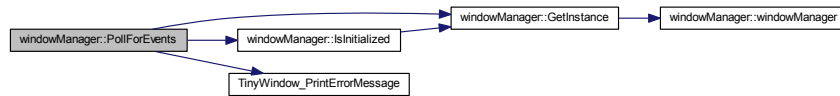
Referenced by `main()`.

```

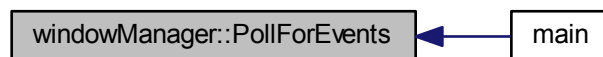
1607     {
1608         if ( GetInstance()->IsInitialized() )
1609         {
1610             #if defined( _WIN32 ) || defined( _WIN64 )
1611                 //only process events if there are any to process
1612                 if (PeekMessage(&instance->message, 0, 0, 0, PM_REMOVE))
1613                 {
1614                     TranslateMessage(&instance->message);
1615                     DispatchMessage(&instance->message);
1616                 }
1617             #elif defined(__linux__)
1618                 //if there are any events to process
1619                 if (XEventsQueued(instance->currentDisplay, QueuedAfterReading))
1620                 {
1621                     XNextEvent(instance->currentDisplay, &instance->currentEvent);
1622                     XEvent currentEvent = instance->currentEvent;
1623                     Linux_ProcessEvents(currentEvent);
1624                 }
1625             #endif
1626         }
1627     }
1628 }
1629
1630 else
1631 {
1632     TinyWindow_PrintErrorMessage(
        tinyWindowError_t::NOT_INITIALIZED );
1633 }
1634 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.3.68 static bool WindowManager::RemoveWindowByIndex ( unsigned int *windowIndex* ) [inline],[static]

Remove window from the manager by index

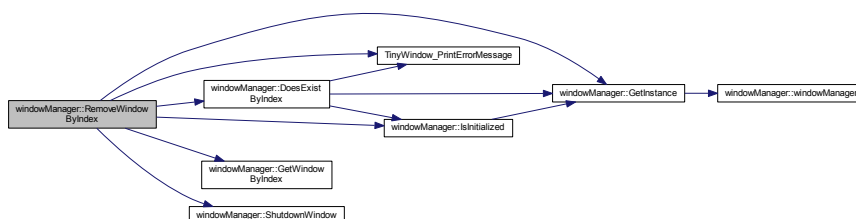
Definition at line 1685 of file TinyWindow.h.

References `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `ShutdownWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1686     {
1687         if ( GetInstance()->IsInitialized() )
1688         {
1689             if ( DoesExistByIndex( windowIndex ) )
1690             {
1691                 ShutdownWindow( GetWindowByIndex( windowIndex ) );
1692                 return true;
1693             }
1694             TinyWindow_PrintErrorMessage(
1695 tinyWindowError_t::WINDOW_NOT_FOUND);
1696             return false;
1697         }
1698         TinyWindow_PrintErrorMessage(
1699 tinyWindowError_t::NOT_INITIALIZED );
1700         return false;
1701     }
  
```

Here is the call graph for this function:



#### 4.2.3.69 static bool windowManager::RemoveWindowByName ( const char \* *windowName* ) [inline],[static]

Remove window from the manager by name

Definition at line 1667 of file TinyWindow.h.

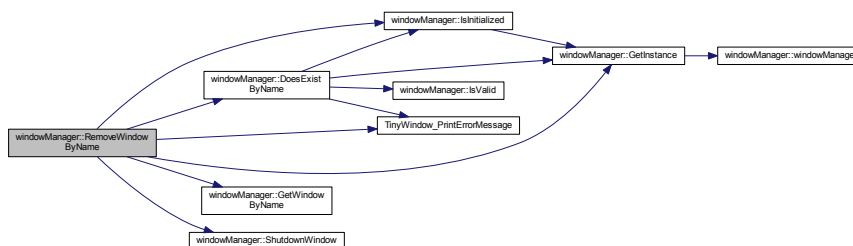
References `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `ShutdownWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1668     {
1669         if ( GetInstance()->IsInitialized() )
1670         {
1671             if ( DoesExistByName( windowName ) )
1672             {
1673                 ShutdownWindow( GetWindowByName( windowName ) );
1674                 return true;
1675             }
1676             TinyWindow_PrintErrorMessage(
1677 tinyWindowError_t::WINDOW_NOT_FOUND);
1678             return false;
1679         }
1680         TinyWindow_PrintErrorMessage(
1681 tinyWindowError_t::NOT_INITIALIZED );
1682         return false;
1683     }

```

Here is the call graph for this function:



#### 4.2.3.70 static bool windowManager::RestoreWindowByIndex ( unsigned int *windowIndex* ) [inline],[static]

Restore the window by index

Definition at line 1532 of file TinyWindow.h.

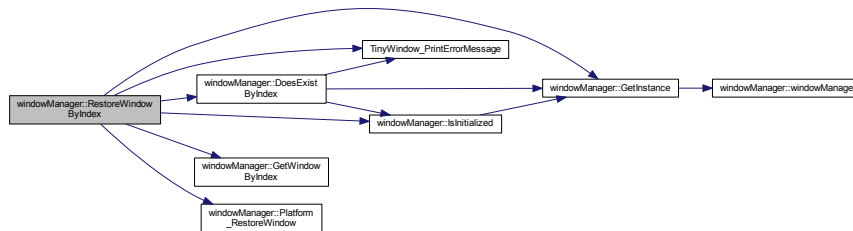
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_RestoreWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1533     {
1534         if ( GetInstance()->IsInitialized() )
1535         {
1536             if ( DoesExistByIndex( windowIndex ) )
1537             {
1538                 window_t* window = GetWindowByIndex( windowIndex );
1539                 Platform_RestoreWindow( window );
1540                 return true;
1541             }
1542             TinyWindow_PrintErrorMessage(
1543 tinyWindowError_t::WINDOW_NOT_FOUND);
1544             return false;
1545         }
1546         TinyWindow_PrintErrorMessage(
1547 tinyWindowError_t::NOT_INITIALIZED);
1548         return false;
1549     }

```

Here is the call graph for this function:



#### 4.2.3.71 static bool WindowManager::RestoreWindowByName ( const char \* *windowName* ) [inline],[static]

Restore the window by name

Definition at line 1513 of file TinyWindow.h.

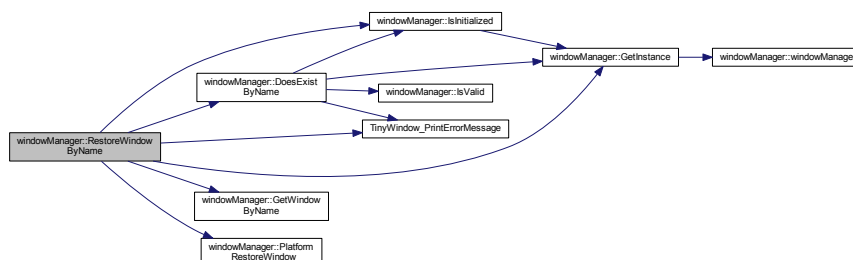
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_RestoreWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1514     {
1515         if ( GetInstance() -> IsInitialized() )
1516         {
1517             if ( DoesExistByName( windowName ) )
1518             {
1519                 window_t* window = GetWindowByName(windowName);
1520                 Platform_RestoreWindow(window);
1521                 return true;
1522             }
1523             TinyWindow_PrintErrorMessage(
1524                 tinyWindowError_t::WINDOW_NOT_FOUND);
1525             return false;
1526         }
1527         TinyWindow_PrintErrorMessage(
1528             tinyWindowError_t::NOT_INITIALIZED );
1529         return false;
1530     }

```

Here is the call graph for this function:



**4.2.3.72** `static bool windowManager::SetFullScreenByIndex ( unsigned int windowIndex, bool newState ) [inline],  
[static]`

Definition at line 1123 of file TinyWindow.h.

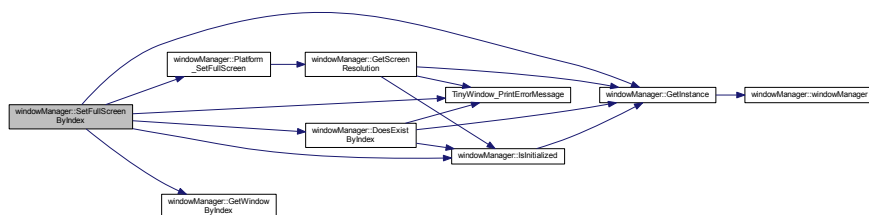
References `windowManager::window_t::AtomDesktopGeometry`, `windowManager::window_t::currentState`, `DoesExistByIndex()`, `FULLSCREEN`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NORMAL`, `NOT_INITIALIZED`, `Platform_SetFullScreen()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1124     {
1125         if ( GetInstance()->IsInitialized() )
1126         {
1127             if ( DoesExistByIndex( windowIndex ) )
1128             {
1129                 window_t* window = GetWindowByIndex(windowIndex);
1130                 window->currentState = (newState == true) ?
1131                 tinyWindowState_t::FULLSCREEN :
1132                 tinyWindowState_t::NORMAL;
1133                 Platform_SetFullScreen(window);
1134                 return true;
1135             }
1136             TinyWindow_PrintErrorMessage(
1137             tinyWindowError_t::WINDOW_NOT_FOUND);
1138             return false;
1139         }
1140         TinyWindow_PrintErrorMessage(
1141         tinyWindowError_t::NOT_INITIALIZED );
1142         return false;
1143     }

```

Here is the call graph for this function:



**4.2.3.73** `static bool windowManager::SetFullScreenByName ( const char * windowName, bool newState ) [inline],  
[static]`

Toggle the given window's full screen mode

Definition at line 1101 of file TinyWindow.h.

References `windowManager::window_t::AtomDesktopGeometry`, `windowManager::window_t::currentState`, `DoesExistByName()`, `FULLSCREEN`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NORMAL`, `NOT_INITIALIZED`, `Platform_SetFullScreen()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1102     {
1103         if ( GetInstance()->IsInitialized() )
1104         {
1105             if ( DoesExistByName( windowName ) )
1106             {
1107                 window_t* window = GetWindowByName(windowName);
1108                 window->currentState = (newState == true) ?
1109

```

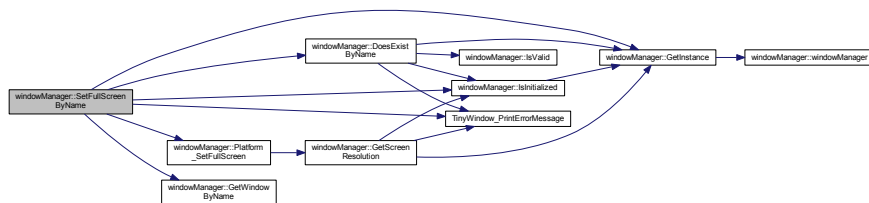


```

        tinyWindowState_t::FULLSCREEN :
        tinyWindowState_t::NORMAL;
1110
1111         Platform_SetFullScreen (window);
1112         return true;
1113     }
1114     TinyWindow_PrintErrorMessage (
        tinyWindowError_t::WINDOW_NOT_FOUND);
1115     return false;
1116 }
1117 TinyWindow_PrintErrorMessage (
        tinyWindowError_t::NOT_INITIALIZED );
1118 return false;
1119 }

```

Here is the call graph for this function:



#### 4.2.3.74 static bool WindowManager::SetMousePositionInScreen ( unsigned int x, unsigned int y ) [inline], [static]

Set the position of the mouse cursor relative to screen co-ordinates

Definition at line 467 of file TinyWindow.h.

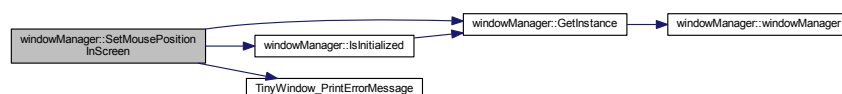
References `GetInstance()`, `instance`, `IsInitialized()`, `NOT_INITIALIZED`, `screenMousePosition`, and `TinyWindow_PrintErrorMessage()`.

```

468     {
469         if ( GetInstance()->IsInitialized() )
470         {
471             instance->screenMousePosition[0] = x;
472             instance->screenMousePosition[1] = y;
473
474             #if defined( _WIN32 ) || defined( _WIN64 )
475                 SetCursorPos(x, y);
476             #elif defined(__linux__)
477                 XWarpPointer(instance->currentDisplay, None,
478                             XDefaultRootWindow(instance->currentDisplay), 0, 0,
479                             GetScreenResolution()[0],
480                             GetScreenResolution()[1],
481                             x, y);
482             #endif
483             return true;
484         }
485         TinyWindow_PrintErrorMessage (
            tinyWindowError_t::NOT_INITIALIZED );
486         return false;
487     }

```

Here is the call graph for this function:



#### 4.2.3.75 static bool windowManager::SetMousePositionInWindowByIndex ( unsigned int *windowIndex*, unsigned int *x*, unsigned int *y* ) [inline],[static]

Set the mouse Position of the given window's co-ordinates

Definition at line 888 of file TinyWindow.h.

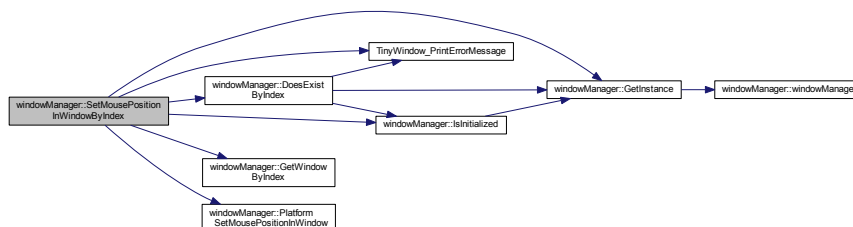
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `windowManager::window_t::mousePosition`, `NOT_INITIALIZED`, `Platform_SetMousePositionInWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

889 {
890     if ( GetInstance()->IsInitialized() )
891     {
892         if ( DoesExistByIndex( windowIndex ) )
893         {
894             GetWindowByIndex( windowIndex )->mousePosition[ 0 ] = x;
895             GetWindowByIndex( windowIndex )->mousePosition[ 1 ] = y;
896             window_t* window = GetWindowByIndex(windowIndex);
897
898             Platform_SetMousePositionInWindow(window, x, y);
899             return true;
900         }
901         TinyWindow_PrintErrorMessage(
902         tinyWindowError_t::WINDOW_NOT_FOUND);
903         return false;
904     }
905     TinyWindow_PrintErrorMessage(
906     tinyWindowError_t::NOT_INITIALIZED );
907     return false;
908 }

```

Here is the call graph for this function:



#### 4.2.3.76 static bool windowManager::SetMousePositionInWindowByName ( const char \* *windowName*, unsigned int *x*, unsigned int *y* ) [inline],[static]

Set the mouse Position of the given window's co-ordinates

Definition at line 866 of file TinyWindow.h.

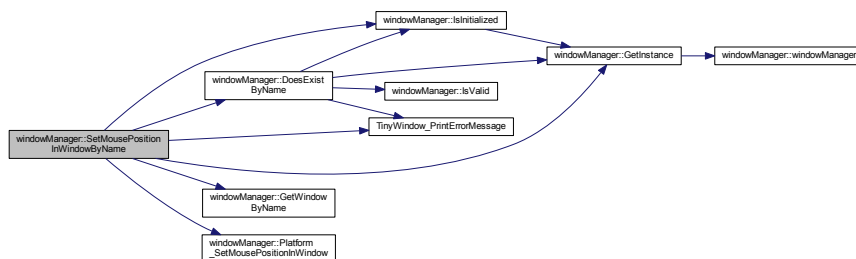
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `windowManager::window_t::mousePosition`, `NOT_INITIALIZED`, `Platform_SetMousePositionInWindow()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

867     {
868         if ( GetInstance()->IsInitialized() )
869         {
870             if ( DoesExistByName( windowName ) )
871             {
872                 GetWindowByName( windowName )->mousePosition[ 0 ] = x;
873                 GetWindowByName( windowName )->mousePosition[ 1 ] = y;
874                 window_t* window = GetWindowByName(windowName);
875                 Platform_SetMousePositionInWindow(window, x, y);
876                 return true;
877             }
878             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
879             return false;
880         }
881
882         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
883         return false;
884     }

```

Here is the call graph for this function:



#### 4.2.3.77 static bool windowManager::SetWindowIconByIndex ( void ) [inline],[static]

Set the window icon by index (currently not functional)

Definition at line 1408 of file TinyWindow.h.

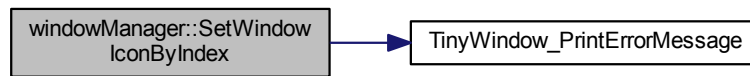
References FUNCTION\_NOT\_IMPLEMENTED, and TinyWindow\_PrintErrorMessage().

```

1409     {
1410         TinyWindow_PrintErrorMessage(
tinyWindowError_t::FUNCTION_NOT_IMPLEMENTED);
1411         return false;
1412         /*if ( GetInstance()->IsInitialized() )
1413         {
1414             if (IsValid(icon))
1415             {
1416                 if (DoesExistByIndex(windowIndex))
1417                 {
1418                     #if defined( _WIN32 ) || defined( _WIN64 )
1419                         Windows_SetWindowIcon(GetWindowByIndex(windowIndex), icon, width, height);
1420                     #elif defined(__linux__)
1421                         Linux_SetWindowIcon(GetWindowByIndex(windowIndex), icon, width, height);
1422                     #endif
1423                     return true;
1424                 }
1425                 TinyWindow_PrintErrorMessage(tinyWindowError_t::TINYWINDOW_ERROR_WINDOW_NOT_FOUND);
1426                 return false;
1427             }
1428             TinyWindow_PrintErrorMessage(tinyWindowError_t::TINYWINDOW_ERROR_INVALID_ICON_PATH);
1429             return false;
1430         }
1431         TinyWindow_PrintErrorMessage( tinyWindowError_t::TINYWINDOW_ERROR_NOT_INITIALIZED );
1432         return false;*/
1433     }

```

Here is the call graph for this function:



#### 4.2.3.78 static bool windowManager::SetWindowIconByName ( void ) [inline],[static]

Set the window icon by name (currently not functional)

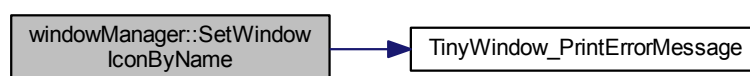
Definition at line 1377 of file TinyWindow.h.

References `FUNCTION_NOT_IMPLEMENTED`, and `TinyWindow_PrintErrorMessage()`.

```

1378     {
1379         TinyWindow_PrintErrorMessage (
1380             tinyWindowError_t::FUNCTION_NOT_IMPLEMENTED);
1381         return false;
1382         /* if ( GetInstance()->IsInitialized() )
1383         {
1384             if (IsValid(icon))
1385             {
1386                 if (DoesExistByName(windowName))
1387                 {
1388                     #if defined( _WIN32 ) || defined( _WIN64 )
1389                         //Windows_SetWindowIcon(GetWindowByName(windowName), icon, width, height);
1390                     #elif defined(__linux__)
1391                         //Linux_SetWindowIcon(); //GetWindowByName(windowName), icon, width, height);
1392                     #endif
1393                     return true;
1394                 }
1395                 TinyWindow_PrintErrorMessage(tinyWindowError_t::TINYWINDOW_ERROR_WINDOW_NOT_FOUND);
1396                 return false;
1397             }
1398             TinyWindow_PrintErrorMessage(tinyWindowError_t::TINYWINDOW_ERROR_INVALID_ICON_PATH);
1399             return false;
1400         }
1401         TinyWindow_PrintErrorMessage( tinyWindowError_t::TINYWINDOW_ERROR_NOT_INITIALIZED );
1402         return false;*/
1403     }
1404 }
  
```

Here is the call graph for this function:



#### 4.2.3.79 static bool WindowManager::SetWindowOnDestroyedByIndex ( unsigned int *windowIndex*, std::function< void(void)> *onDestroyed* ) [inline],[static]

Set the window on destroyed event callback by index

Definition at line 1986 of file TinyWindow.h.

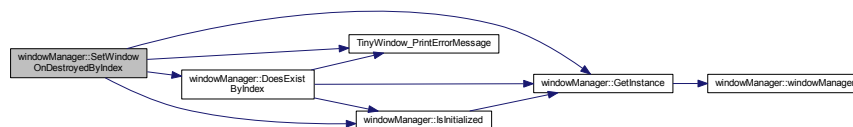
References `DoesExistByIndex()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1987     {
1988         if ( GetInstance()->IsInitialized() )
1989         {
1990             if ( DoesExistByIndex( windowIndex ) )
1991             {
1992                 if (onDestroyed != nullptr)
1993                 {
1994                     GetWindowByIndex(windowIndex)->
1995                     destroyedEvent = onDestroyed;
1996                     return true;
1997                 }
1998                 TinyWindow_PrintErrorMessage(
1999                 tinyWindowError_t::INVALID_CALLBACK);
2000                 return false;
2001             }
2002             TinyWindow_PrintErrorMessage(
2003             tinyWindowError_t::WINDOW_NOT_FOUND);
2004             return false;
2005         }
2006     }

```

Here is the call graph for this function:



#### 4.2.3.80 static bool WindowManager::SetWindowOnDestroyedByName ( const char \* *windowName*, std::function< void(void)> *onDestroyed* ) [inline],[static]

Set the window on destroyed event callback by name

Definition at line 1963 of file TinyWindow.h.

References `DoesExistByName()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1964     {
1965         if ( GetInstance()->IsInitialized() )
1966         {
1967             if ( DoesExistByName( windowName ) )
1968             {
1969                 if (onDestroyed != nullptr)
1970                 {
1971                     GetWindowByName(windowName)->destroyedEvent = onDestroyed;
1972                     return true;
1973                 }
1974             }
1975         }
1976     }

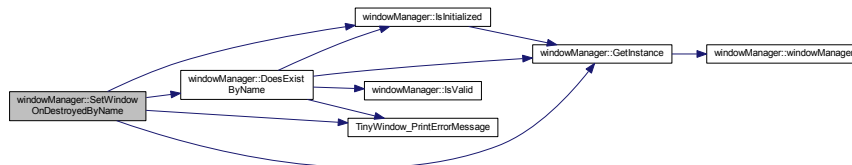
```

```

1974         TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
1975         return false;
1976     }
1977     TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1978     return false;
1979 }
1980 TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1981     return false;
1982 }

```

Here is the call graph for this function:



**4.2.3.81** static bool windowManager::SetWindowOnFocusByIndex ( unsigned int *windowIndex*, std::function< void(bool)> *onFocus* ) [inline], [static]

Set the window on focus event callback by index

Definition at line 2127 of file TinyWindow.h.

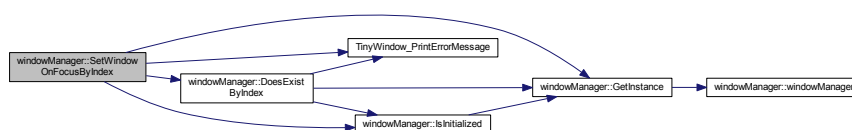
References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2128     {
2129         if ( GetInstance()->IsInitialized() )
2130         {
2131             if ( DoesExistByIndex( windowIndex ) )
2132             {
2133                 if (onFocus != nullptr)
2134                 {
2135                     GetWindowByIndex(windowIndex)->focusEvent = onFocus;
2136                     return true;
2137                 }
2138                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
2139                 return false;
2140             }
2141             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2142             return false;
2143         }
2144         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2145         return false;
2146     }

```

Here is the call graph for this function:



**4.2.3.82** static bool windowManager::SetWindowOnFocusByName ( const char \* *windowName*, std::function< void(bool)> *onFocus* ) [inline],[static]

Set the window on focus event callback by name

Definition at line 2104 of file TinyWindow.h.

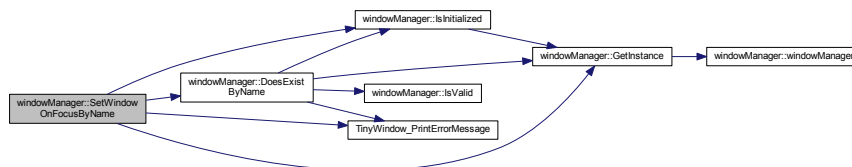
References DoesExistByName(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2105     {
2106         if ( GetInstance()->IsInitialized() )
2107         {
2108             if ( DoesExistByName( windowName ) )
2109             {
2110                 if (onFocus != nullptr)
2111                 {
2112                     GetWindowByName(windowName)->focusEvent = onFocus;
2113                     return true;
2114                 }
2115                 TinyWindow_PrintErrorMessage(
2116 tinyWindowError_t::INVALID_CALLBACK);
2117                 return false;
2118             }
2119             TinyWindow_PrintErrorMessage(
2120 tinyWindowError_t::WINDOW_NOT_FOUND);
2121             return false;
2122         }
2123         TinyWindow_PrintErrorMessage(
2124 tinyWindowError_t::NOT_INITIALIZED );
2125         return false;
2126     }

```

Here is the call graph for this function:



**4.2.3.83** static bool windowManager::SetWindowOnKeyEventByIndex ( unsigned int *windowIndex*, std::function< void(unsigned int, tinyWindowKeyState\_t)> *onKey* ) [inline],[static]

Set the window on key event callback by index

Definition at line 1845 of file TinyWindow.h.

References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1846     {
1847         if ( GetInstance()->IsInitialized() )
1848         {
1849             if ( DoesExistByIndex( windowIndex ) )
1850             {
1851                 if (onKey != nullptr)
1852                 {
1853                     GetWindowByIndex(windowIndex)->keyEvent = onKey;
1854                     return true;
1855                 }
1856             }
1857             TinyWindow_PrintErrorMessage(
1858 tinyWindowError_t::INVALID_CALLBACK);
1859             return false;
1860         }
1861         TinyWindow_PrintErrorMessage(
1862 tinyWindowError_t::WINDOW_NOT_FOUND);
1863         return false;
1864     }

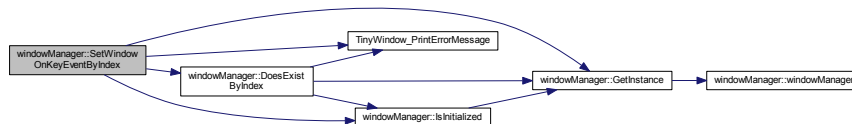
```

```

1855         }
1856         TinyWindow_PrintErrorMessage(
1857     tinyWindowError_t::INVALID_CALLBACK);
1858         return false;
1859     }
1860     TinyWindow_PrintErrorMessage(
1861     tinyWindowError_t::WINDOW_NOT_FOUND);
1862     return false;
1863 }
1864 TinyWindow_PrintErrorMessage(
1865     tinyWindowError_t::NOT_INITIALIZED );
1866     return false;
1867 }

```

Here is the call graph for this function:



**4.2.3.84** static bool windowManager::SetWindowOnKeyEventByName ( const char \* *windowName*, std::function< void(unsigned int, tinyWindowKeyState\_t)> *onKey* ) [inline], [static]

Set the window on key event callback by name

Definition at line 1821 of file TinyWindow.h.

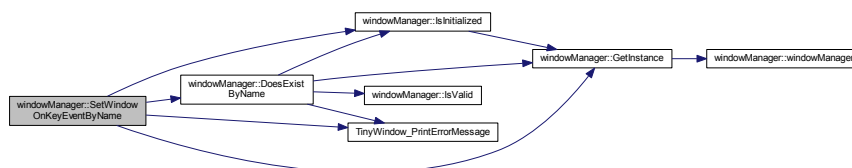
References DoesExistByName(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1822     {
1823         if ( GetInstance()->IsInitialized() )
1824         {
1825             if ( DoesExistByName( windowName ) )
1826             {
1827                 if (onKey != nullptr)
1828                 {
1829                     GetWindowByName(windowName)->keyEvent = onKey;
1830                     return true;
1831                 }
1832                 TinyWindow_PrintErrorMessage(
1833     tinyWindowError_t::INVALID_CALLBACK);
1834                 return false;
1835             }
1836             TinyWindow_PrintErrorMessage(
1837     tinyWindowError_t::WINDOW_NOT_FOUND);
1838             return false;
1839         }
1840         TinyWindow_PrintErrorMessage(
1841     tinyWindowError_t::NOT_INITIALIZED );
1842         return false;
1843     }

```

Here is the call graph for this function:





**4.2.3.85** static bool WindowManager::SetWindowOnMaximizedByIndex ( unsigned int *windowIndex*, std::function< void(void)> *onMaximized* ) [inline],[static]

Set the window on maximized event callback by index

Definition at line 2033 of file TinyWindow.h.

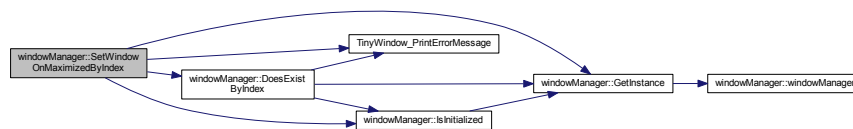
References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2034     {
2035         if ( GetInstance()->IsInitialized() )
2036         {
2037             if ( DoesExistByIndex( windowIndex ) )
2038             {
2039                 if (onMaximized != nullptr)
2040                 {
2041                     GetWindowByIndex(windowIndex)->
2042                     maximizedEvent = onMaximized;
2043                     return true;
2044                 }
2045                 TinyWindow_PrintErrorMessage(
2046                 tinyWindowError_t::INVALID_CALLBACK);
2047                 return false;
2048             }
2049             TinyWindow_PrintErrorMessage(
2050             tinyWindowError_t::WINDOW_NOT_FOUND);
2051             return false;
2052         }
2053     }

```

Here is the call graph for this function:



**4.2.3.86** static bool WindowManager::SetWindowOnMaximizedByName ( const char \* *windowName*, std::function< void(void)> *onMaximized* ) [inline],[static]

Set the window on maximized event callback by name

Definition at line 2010 of file TinyWindow.h.

References DoesExistByName(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2011     {
2012         if ( GetInstance()->IsInitialized() )
2013         {
2014             if ( DoesExistByName( windowName ) )
2015             {
2016                 if (onMaximized != nullptr)
2017                 {
2018                     GetWindowByName(windowName)->maximizedEvent = onMaximized;
2019                     return true;
2020                 }
2021             }
2022         }
2023     }

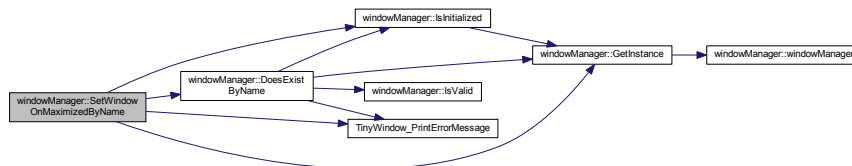
```

```

2021         TinyWindow_PrintErrorMessage(
2022         tinyWindowError_t::INVALID_CALLBACK);
2023     }
2024     TinyWindow_PrintErrorMessage(
2025     tinyWindowError_t::WINDOW_NOT_FOUND);
2026     return false;
2027 }
2028 TinyWindow_PrintErrorMessage(
2029 tinyWindowError_t::NOT_INITIALIZED );
2030 return false;
2031 }

```

Here is the call graph for this function:



**4.2.3.87** static bool windowManager::SetWindowOnMinimizedByIndex ( unsigned int *windowIndex*, std::function< void(void)> *onMinimized* ) [inline],[static]

Set the window on minimized event callback by index

Definition at line 2080 of file TinyWindow.h.

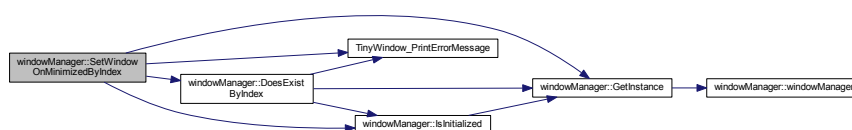
References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2081     {
2082         if ( GetInstance() -> IsInitialized() )
2083         {
2084             if ( DoesExistByIndex( windowIndex ) )
2085             {
2086                 if ( onMinimized != nullptr )
2087                 {
2088                     GetWindowByIndex( windowIndex ) ->
2089                     minimizedEvent = onMinimized;
2090                     return true;
2091                 }
2092                 TinyWindow_PrintErrorMessage(
2093                 tinyWindowError_t::INVALID_CALLBACK);
2094                 return false;
2095             }
2096             TinyWindow_PrintErrorMessage(
2097             tinyWindowError_t::WINDOW_NOT_FOUND);
2098             return false;
2099         }
2100         TinyWindow_PrintErrorMessage(
2101         tinyWindowError_t::NOT_INITIALIZED );
2102         return false;
2103     }

```

Here is the call graph for this function:



**4.2.3.88** `static bool WindowManager::SetWindowOnMinimizedByName ( const char * windowName, std::function< void(void)> onMinimized ) [inline],[static]`

Set the window on minimized event callback by name

Definition at line 2057 of file TinyWindow.h.

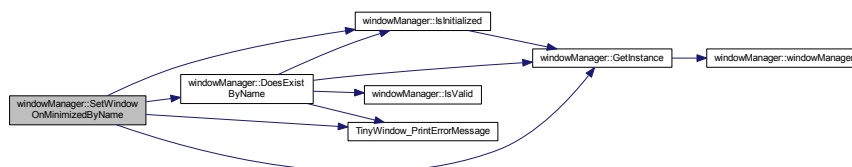
References `DoesExistByName()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

2058     {
2059         if ( GetInstance()->IsInitialized() )
2060         {
2061             if ( DoesExistByName( windowName ) )
2062             {
2063                 if (onMinimized != nullptr)
2064                 {
2065                     GetWindowByName(windowName)->minimizedEvent = onMinimized;
2066                     return true;
2067                 }
2068                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
2069                 return false;
2070             }
2071             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2072             return false;
2073         }
2074         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2075         return false;
2076     }

```

Here is the call graph for this function:



**4.2.3.89** `static bool WindowManager::SetWindowOnMouseButtonEventByIndex ( unsigned int windowIndex, std::function< void(tinyWindowMouseButton_t, tinyWindowButtonState_t)> onMouseButton ) [inline],[static]`

Set the window on mouse button event callback by index

Definition at line 1892 of file TinyWindow.h.

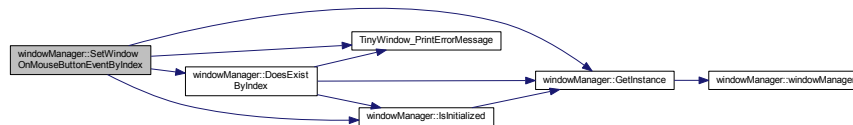
References `DoesExistByIndex()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1893     {
1894         if ( GetInstance()->IsInitialized() )
1895         {
1896             if ( DoesExistByIndex( windowIndex ) )
1897             {
1898                 if (onMouseButton != nullptr)
1899                 {
1900                     GetWindowByIndex(windowIndex)->
mouseButtonEvent = onMouseButton;
1901                     return true;
1902                 }
1903                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
1904                 return false;
1905             }
1906             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1907             return false;
1908         }
1909         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1910         return false;
1911     }

```

Here is the call graph for this function:



**4.2.3.90** static bool windowManager::SetWindowOnMouseButtonEventByName ( const char \* *windowName*, std::function< void(tinyWindowMouseButton\_t, tinyWindowButtonState\_t)> *onMouseButton* ) [inline], [static]

Set the window on mouse button event callback by name

Definition at line 1869 of file TinyWindow.h.

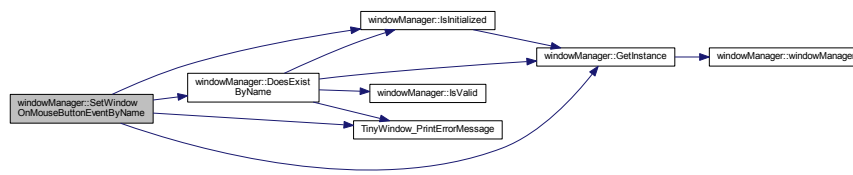
References DoesExistByName(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1870     {
1871         if ( GetInstance()->IsInitialized() )
1872         {
1873             if ( DoesExistByName( windowName ) )
1874             {
1875                 if (onMouseButton != nullptr)
1876                 {
1877                     GetWindowByName(windowName)->mouseButtonEvent =
onMouseButton;
1878                     return true;
1879                 }
1880                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
1881                 return false;
1882             }
1883             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1884             return false;
1885         }
1886         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1887         return false;
1888     }

```

Here is the call graph for this function:



**4.2.3.91** static bool `windowManager::SetWindowOnMouseMoveByIndex ( unsigned int windowIndex, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> onMouseMove )` `[inline]`, `[static]`

Set the window on mouse move event callback by index

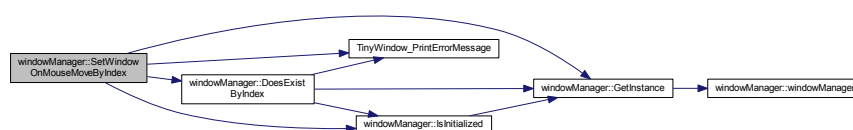
Definition at line 2269 of file `TinyWindow.h`.

References `DoesExistByIndex()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

2270     {
2271         if ( GetInstance()->IsInitialized() )
2272         {
2273             if ( DoesExistByIndex( windowIndex ) )
2274             {
2275                 if (onMouseMove != nullptr)
2276                 {
2277                     GetWindowByIndex(windowIndex)->
mouseMoveEvent = onMouseMove;
2278                     return true;
2279                 }
2280                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
2281                 return false;
2282             }
2283             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2284             return false;
2285         }
2286         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2287         return false;
2288     }
  
```

Here is the call graph for this function:



**4.2.3.92** static bool windowManager::SetWindowOnMouseMoveByName ( const char \* *windowName*, std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> *onMouseMove* ) [inline],[static]

Set the window on mouse move event callback by name

Definition at line 2246 of file TinyWindow.h.

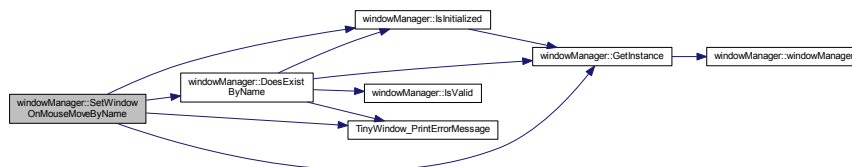
References DoesExistByName(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2247     {
2248         if ( GetInstance()->IsInitialized() )
2249         {
2250             if ( DoesExistByName( windowName ) )
2251             {
2252                 if (onMouseMove != nullptr)
2253                 {
2254                     GetWindowByName(windowName)->mouseMoveEvent = onMouseMove;
2255                     return true;
2256                 }
2257                 TinyWindow_PrintErrorMessage(
2258 tinyWindowError_t::INVALID_CALLBACK);
2259                 return false;
2260             }
2261             TinyWindow_PrintErrorMessage(
2262 tinyWindowError_t::WINDOW_NOT_FOUND);
2263             return false;
2264         }
2265         TinyWindow_PrintErrorMessage(
2266 tinyWindowError_t::NOT_INITIALIZED );
2267         return false;
2268     }

```

Here is the call graph for this function:



**4.2.3.93** static bool windowManager::SetWindowOnMouseWheelEventByIndex ( unsigned int *windowIndex*, std::function< void(tinyWindowMouseScroll\_t)> *onMouseWheel* ) [inline],[static]

Set the window on mouse wheel event callback by index

Definition at line 1939 of file TinyWindow.h.

References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1940     {
1941         if ( GetInstance()->IsInitialized() )
1942         {
1943             if ( DoesExistByIndex( windowIndex ) )
1944             {
1945                 if (onMouseWheel != nullptr)
1946                 {
1947                     GetWindowByIndex(windowIndex)->
mouseWheelEvent = onMouseWheel;

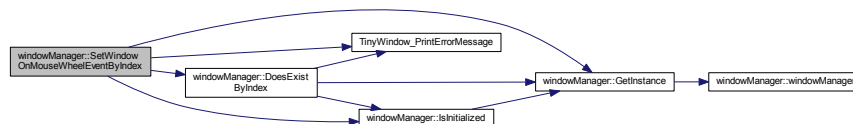
```

```

1948         return true;
1949     }
1950     TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
1951     return false;
1952 }
1953 TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1954     return false;
1955 }
1956 TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1957     return false;
1958 }

```

Here is the call graph for this function:



#### 4.2.3.94 static bool WindowManager::SetWindowOnMouseWheelEventByName ( const char \* *windowName*, std::function< void(tinyWindowMouseScroll\_t)> *onMouseWheel* ) [inline], [static]

Set the window on mouse wheel event callback by name

Definition at line 1916 of file TinyWindow.h.

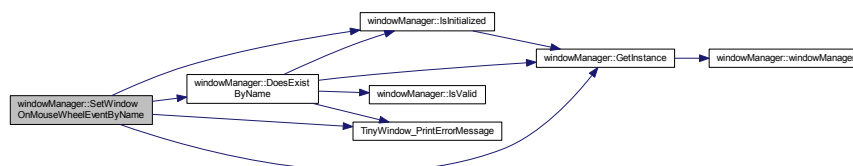
References `DoesExistByName()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1917     {
1918         if ( GetInstance() -> IsInitialized() )
1919         {
1920             if ( DoesExistByName( windowName ) )
1921             {
1922                 if (onMouseWheel != nullptr)
1923                 {
1924                     GetWindowByName(windowName)->mouseWheelEvent =
onMouseWheel;
1925                     return true;
1926                 }
1927                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
1928                 return false;
1929             }
1930             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
1931             return false;
1932         }
1933         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
1934         return false;
1935     }

```

Here is the call graph for this function:



**4.2.3.95** `static bool WindowManager::SetWindowOnMovedByIndex ( unsigned int windowIndex, std::function< void(unsigned int, unsigned int)> onMoved ) [inline],[static]`

Set the window on moved event callback by index

Definition at line 2175 of file TinyWindow.h.

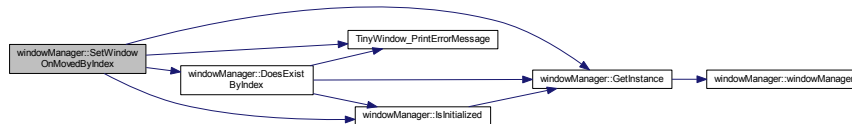
References `DoesExistByIndex()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

2176     {
2177         if ( GetInstance()->IsInitialized() )
2178         {
2179             if ( DoesExistByIndex( windowIndex ) )
2180             {
2181                 if (onMoved != nullptr)
2182                 {
2183                     GetWindowByIndex(windowIndex)->movedEvent = onMoved;
2184                     return true;
2185                 }
2186                 TinyWindow_PrintErrorMessage(
2187 tinyWindowError_t::INVALID_CALLBACK);
2188                 return false;
2189             }
2190             TinyWindow_PrintErrorMessage(
2191 tinyWindowError_t::WINDOW_NOT_FOUND);
2192             return false;
2193         }
2194         TinyWindow_PrintErrorMessage(
2195 tinyWindowError_t::NOT_INITIALIZED );
2196         return false;
2197     }

```

Here is the call graph for this function:



**4.2.3.96** `static bool WindowManager::SetWindowOnMovedByName ( const char * windowName, std::function< void(unsigned int, unsigned int)> onMoved ) [inline],[static]`

Set the window on moved event callback by name

Definition at line 2151 of file TinyWindow.h.

References `DoesExistByName()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

2152     {
2153         if ( GetInstance()->IsInitialized() )
2154         {
2155             if ( DoesExistByName( windowName ) )
2156             {
2157                 if (onMoved != nullptr)
2158                 {
2159                     GetWindowByName(windowName)->movedEvent = onMoved;
2160                     return true;
2161                 }
2162                 TinyWindow_PrintErrorMessage(

```

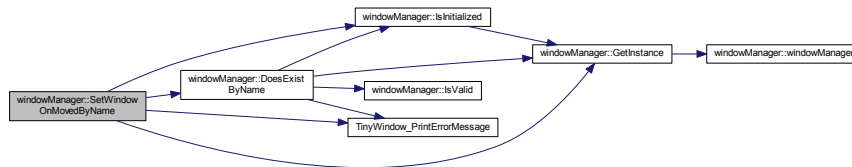


```

        tinyWindowError_t::INVALID_CALLBACK);
2163         return false;
2164     }
2165
2166     TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2167     return false;
2168 }
2169 TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2170     return false;
2171 }

```

Here is the call graph for this function:



**4.2.3.97** static bool WindowManager::SetWindowOnResizeByIndex ( unsigned int *windowIndex*, std::function< void(unsigned int, unsigned int)> *onResize* ) [inline],[static]

Set the window on resized event callback by index

Definition at line 2222 of file TinyWindow.h.

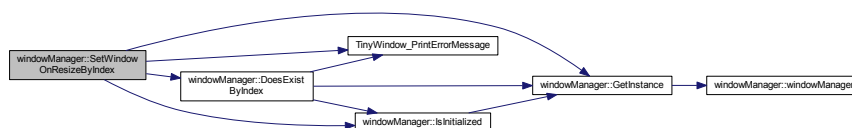
References DoesExistByIndex(), GetInstance(), INVALID\_CALLBACK, IsInitialized(), NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

2223     {
2224         if ( GetInstance()->IsInitialized() )
2225         {
2226             if ( DoesExistByIndex( windowIndex ) )
2227             {
2228                 if (onResize != nullptr)
2229                 {
2230                     GetWindowByIndex(windowIndex)->resizeEvent = onResize;
2231                     return true;
2232                 }
2233                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
2234                 return false;
2235             }
2236             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2237             return false;
2238         }
2239         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2240         return false;
2241     }

```

Here is the call graph for this function:



**4.2.3.98** `static bool windowManager::SetWindowOnResizeByName ( const char * windowName, std::function< void(unsigned int, unsigned int)> onResize ) [inline],[static]`

Set the window on resized event callback by name

Definition at line 2199 of file TinyWindow.h.

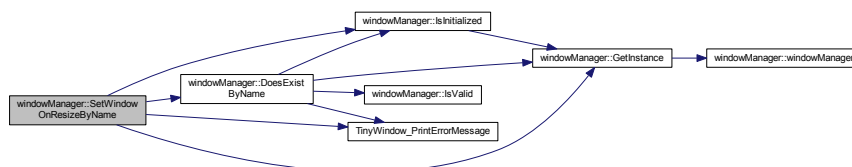
References `DoesExistByName()`, `GetInstance()`, `INVALID_CALLBACK`, `IsInitialized()`, `NOT_INITIALIZED`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

2200     {
2201         if ( GetInstance()->IsInitialized() )
2202         {
2203             if ( DoesExistByName( windowName ) )
2204             {
2205                 if (onResize != nullptr)
2206                 {
2207                     GetWindowByName(windowName)->resizeEvent = onResize;
2208                     return true;
2209                 }
2210                 TinyWindow_PrintErrorMessage(
tinyWindowError_t::INVALID_CALLBACK);
2211                 return false;
2212             }
2213             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
2214             return false;
2215         }
2216         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
2217         return false;
2218     }

```

Here is the call graph for this function:



**4.2.3.99** `static bool windowManager::SetWindowPositionByIndex ( unsigned int windowIndex, unsigned int x, unsigned int y ) [inline],[static]`

Set the position of the given window relative to screen co-ordinates

Definition at line 769 of file TinyWindow.h.

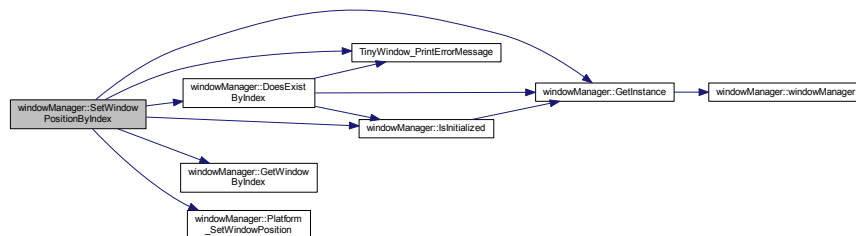
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SetWindowPosition()`, `windowManager::window_t::position`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

770 {
771     if ( GetInstance()->IsInitialized() )
772     {
773         if ( DoesExistByIndex( windowIndex ) )
774         {
775             GetWindowByIndex( windowIndex )->position[ 0 ] = x;
776             GetWindowByIndex( windowIndex )->position[ 1 ] = y;
777             window_t* window = GetWindowByIndex(windowIndex);
778             Platform_SetWindowPosition(window, x, y);
779             return true;
780         }
781
782         TinyWindow_PrintErrorMessage(
783         tinyWindowError_t::WINDOW_NOT_FOUND);
784     }
785
786     TinyWindow_PrintErrorMessage(
787     tinyWindowError_t::NOT_INITIALIZED );
788     return false;
789 }

```

Here is the call graph for this function:



#### 4.2.3.100 static bool windowManager::SetWindowPositionByName ( const char \* *windowName*, unsigned int x, unsigned int y ) [inline],[static]

Set the Position of the given window relative to screen co-ordinates

Definition at line 745 of file TinyWindow.h.

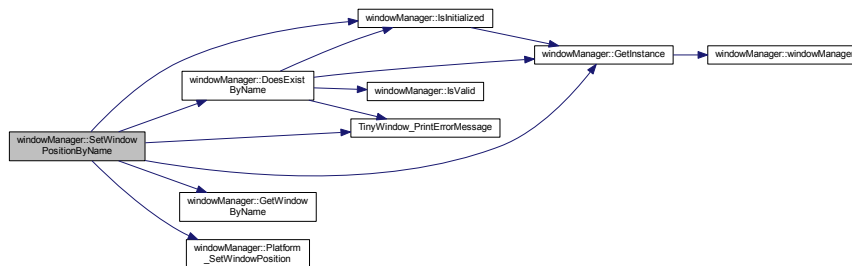
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SetWindowPosition()`, `windowManager::window_t::position`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

746 {
747     if ( GetInstance()->IsInitialized() )
748     {
749         if ( DoesExistByName( windowName ) )
750         {
751             GetWindowByName( windowName )->position[ 0 ] = x;
752             GetWindowByName( windowName )->position[ 1 ] = y;
753             window_t* window = GetWindowByName(windowName);
754
755             Platform_SetWindowPosition(window, x, y);
756             TinyWindow_PrintErrorMessage(
757             tinyWindowError_t::WINDOW_NOT_FOUND);
758             return true;
759         }
760         TinyWindow_PrintErrorMessage(
761         tinyWindowError_t::WINDOW_NOT_FOUND );
762     }
763
764     TinyWindow_PrintErrorMessage(
765     tinyWindowError_t::NOT_INITIALIZED );
766     return false;
767 }

```

Here is the call graph for this function:



**4.2.3.101** static bool windowManager::SetWindowResolutionByIndex ( unsigned int *windowIndex*, unsigned int *width*, unsigned int *height* ) [inline],[static]

Set the Size/Resolution of the given window

Definition at line 647 of file TinyWindow.h.

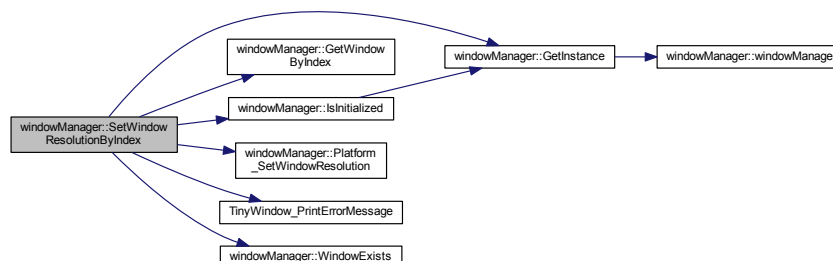
References windowManager::window\_t::AtomDesktopGeometry, GetInstance(), GetWindowByIndex(), IsInitialized(), NOT\_INITIALIZED, Platform\_SetWindowResolution(), windowManager::window\_t::resolution, TinyWindow\_PrintErrorMessage(), WINDOW\_NOT\_FOUND, and WindowExists().

```

648 {
649     if ( GetInstance()->IsInitialized() )
650     {
651         if ( WindowExists( windowIndex ) )
652         {
653             GetWindowByIndex( windowIndex )->resolution[ 0 ] = width;
654             GetWindowByIndex( windowIndex )->resolution[ 1 ] = height;
655             window_t* window = GetWindowByIndex(windowIndex);
656
657             Platform_SetWindowResolution(window);
658             return true;
659         }
660         TinyWindow_PrintErrorMessage(
661             tinyWindowError_t::WINDOW_NOT_FOUND);
662         return false;
663     }
664     TinyWindow_PrintErrorMessage(
665         tinyWindowError_t::NOT_INITIALIZED );
666     return false;
667 }

```

Here is the call graph for this function:



**4.2.3.102** `static bool WindowManager::SetWindowResolutionByName ( const char * windowName, unsigned int width, unsigned int height )` `[inline],[static]`

Set the Size/Resolution of the given window

Definition at line 624 of file TinyWindow.h.

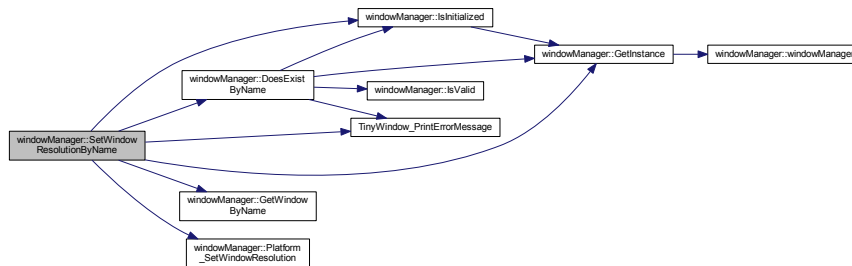
References `WindowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `INVALID_CONTEXT`, `IsInitialized()`, `Platform_SetWindowResolution()`, `WindowManager::window_t::resolution`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

625     {
626         if ( GetInstance()->IsInitialized() )
627         {
628             if ( DoesExistByName( windowName ) )
629             {
630                 GetWindowByName( windowName )->resolution[ 0 ] = width;
631                 GetWindowByName( windowName )->resolution[ 1 ] = height;
632                 window_t* window = GetWindowByName(windowName);
633
634                 Platform_SetWindowResolution(window);
635                 return true;
636             }
637             TinyWindow_PrintErrorMessage(
638                 tinyWindowError_t::WINDOW_NOT_FOUND);
639             return false;
640         }
641         TinyWindow_PrintErrorMessage(
642             tinyWindowError_t::INVALID_CONTEXT );
643         return false;
644     }

```

Here is the call graph for this function:



**4.2.3.103** `static bool WindowManager::SetWindowStyleByIndex ( unsigned int windowIndex, tinyWindowStyle_t windowStyle )` `[inline],[static]`

Set the window style preset by index

Definition at line 1723 of file TinyWindow.h.

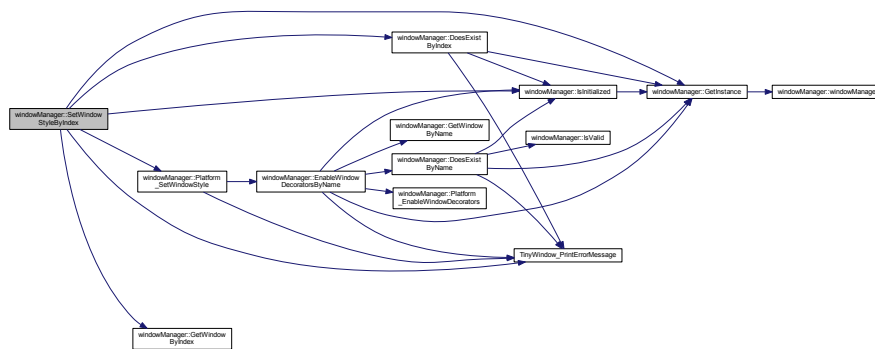
References `WindowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SetWindowStyle()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1724     {
1725         if ( GetInstance()->IsInitialized() )
1726         {
1727             if ( DoesExistByIndex( windowIndex ) )
1728             {
1729                 window_t* window = GetWindowByIndex(windowIndex);
1730                 Platform_SetWindowStyle(window, windowStyle);
1731                 return true;
1732             }
1733             TinyWindow_PrintErrorMessage(
1734 tinyWindowError_t::WINDOW_NOT_FOUND);
1735             return false;
1736         }
1737         TinyWindow_PrintErrorMessage(
1738 tinyWindowError_t::NOT_INITIALIZED );
1739         return false;
1740     }

```

Here is the call graph for this function:



#### 4.2.3.104 static bool windowManager::SetWindowStyleByName ( const char \* *windowName*, tinyWindowStyle\_t *windowStyle* ) [inline],[static]

Set the window style preset by name

Definition at line 1704 of file TinyWindow.h.

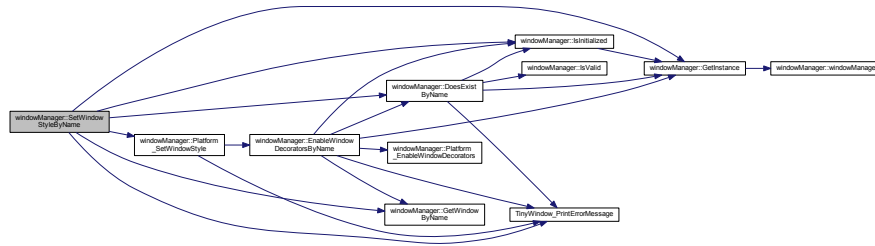
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SetWindowStyle()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1705     {
1706         if ( GetInstance()->IsInitialized() )
1707         {
1708             if ( DoesExistByName( windowName ) )
1709             {
1710                 window_t* window = GetWindowByName(windowName);
1711                 Platform_SetWindowStyle(window, windowStyle);
1712                 return true;
1713             }
1714             TinyWindow_PrintErrorMessage(
1715 tinyWindowError_t::WINDOW_NOT_FOUND);
1716             return false;
1717         }
1718         TinyWindow_PrintErrorMessage(
1719 tinyWindowError_t::NOT_INITIALIZED );
1720         return false;
1721     }

```

Here is the call graph for this function:



#### 4.2.3.105 static bool windowManager::SetWindowTitleBarByIndex ( unsigned int *windowIndex*, const char \* *newName* ) [inline],[static]

Set the window title bar by index

Definition at line 1351 of file TinyWindow.h.

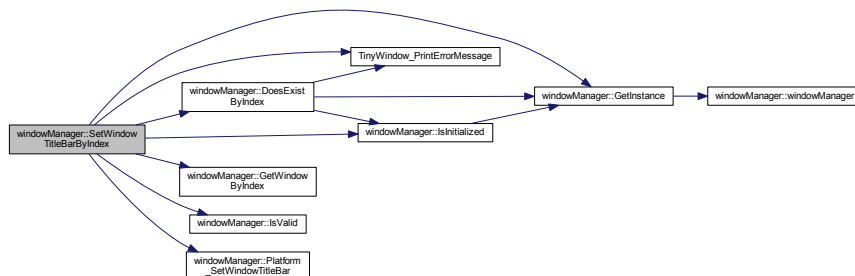
References windowManager::window\_t::AtomDesktopGeometry, DoesExistByIndex(), GetInstance(), GetWindowByIndex(), INVALID\_WINDOW\_NAME, IsInitialized(), IsValid(), NOT\_INITIALIZED, Platform\_SetWindowTitleBar(), TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

1352     {
1353         if ( GetInstance()->IsInitialized() )
1354         {
1355             if (IsValid(newName) )
1356             {
1357                 if (DoesExistByIndex(windowIndex))
1358                 {
1359                     window_t* window = GetWindowByIndex(windowIndex);
1360                     Platform_SetWindowTitleBar(window, newName);
1361                     return true;
1362                 }
1363                 TinyWindow_PrintErrorMessage(
1364 tinyWindowError_t::WINDOW_NOT_FOUND);
1365                 return false;
1366             }
1367             TinyWindow_PrintErrorMessage(
1368 tinyWindowError_t::INVALID_WINDOW_NAME);
1369             return false;
1370         }
1371         TinyWindow_PrintErrorMessage(
1372 tinyWindowError_t::NOT_INITIALIZED );
1373         return false;
1374     }

```

Here is the call graph for this function:



#### 4.2.3.106 static bool windowManager::SetWindowTitleBarByName ( const char \* *windowName*, const char \* *newTitle* ) [inline],[static]

Set the window title bar by name

Definition at line 1327 of file TinyWindow.h.

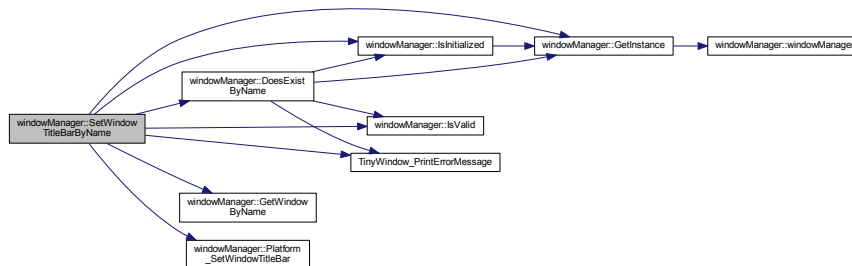
References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `INVALID_WINDOW_NAME`, `IsInitialized()`, `IsValid()`, `NOT_INITIALIZED`, `Platform_SetWindowTitleBar()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

```

1328     {
1329         if ( GetInstance()->IsInitialized() )
1330         {
1331             if (IsValid(newTitle))
1332             {
1333                 if (DoesExistByName(windowName))
1334                 {
1335                     window_t* window = GetWindowByName(windowName);
1336                     Platform_SetWindowTitleBar(window, newTitle);
1337                     return true;
1338                 }
1339                 TinyWindow_PrintErrorMessage(
1340 tinyWindowError_t::WINDOW_NOT_FOUND);
1341                 return false;
1342             }
1343             TinyWindow_PrintErrorMessage(
1344 tinyWindowError_t::INVALID_WINDOW_NAME);
1345             return false;
1346         }
1347         TinyWindow_PrintErrorMessage(
1348 tinyWindowError_t::NOT_INITIALIZED );
1349         return false;
1350     }

```

Here is the call graph for this function:



#### 4.2.3.107 static void windowManager::ShutDown ( void ) [inline],[static]

Use this to shutdown the window manager when your program is finished

Definition at line 368 of file TinyWindow.h.

References `GetInstance()`, `instance`, `IsInitialized()`, `isInitialized`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

Referenced by `main()`.

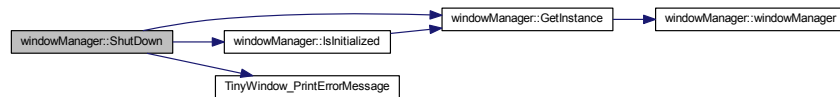


```

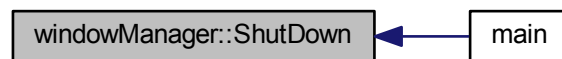
369     {
370         if (GetInstance()->IsInitialized())
371         {
372             for (auto CurrentWindow : instance->windowList)
373             {
374                 delete CurrentWindow;
375             }
376         }
377 #if defined( CURRENT_OS_LINUX )
378         XCloseDisplay(instance->currentDisplay);
379 #endif
380
381         instance->windowList.clear();
382         instance->isInitialized = false;
383         delete instance;
384     }
385     TinyWindow_PrintErrorMessage(
386         tinyWindowError_t::NOT_INITIALIZED);
387 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.108** `static void windowManager::ShutdownWindow ( window_t * window )` `[inline]`, `[static]`, `[private]`

Definition at line 3017 of file `TinyWindow.h`.

References `windowManager::window_t::name`.

Referenced by `RemoveWindowByIndex()`, and `RemoveWindowByName()`.

```

3018     {
3019 #if defined( _WIN32 ) || defined( _WIN64 )
3020         if (window->glRenderingContextHandle)
3021         {
3022             wglMakeCurrent(nullptr, nullptr);
3023             wglDeleteContext(window->glRenderingContextHandle);
3024         }
3025
3026         if (window->paletteHandle)
3027         {
3028             DeleteObject(window->paletteHandle);

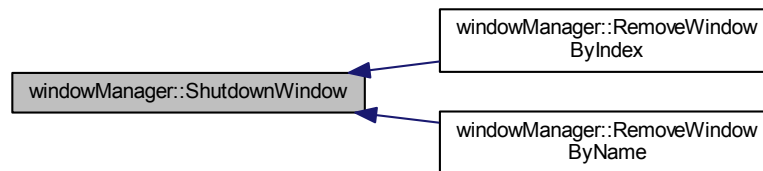
```

```

3029     }
3030     ReleaseDC(window->windowHandle, window->deviceContextHandle);
3031     UnregisterClass(window->name, window->instanceHandle);
3032
3033     FreeModule(window->instanceHandle);
3034
3035     window->deviceContextHandle = nullptr;
3036     window->windowHandle = nullptr;
3037     window->glRenderingContextHandle = nullptr;
3038 #elif defined(__linux__)
3039     if (window->currentState == tinyWindowState_t::FULLSCREEN)
3040     {
3041         RestoreWindowByName(window->name);
3042     }
3043
3044     glXDestroyContext(instance->currentDisplay, window->context);
3045     XUnmapWindow(instance->currentDisplay, window->windowHandle);
3046     XDestroyWindow(instance->currentDisplay, window->windowHandle);
3047     window->windowHandle = 0;
3048     window->context = 0;
3049 #endif
3050 }

```

Here is the caller graph for this function:



#### 4.2.3.109 static void windowManager::WaitForEvents ( void ) [inline],[static]

Ask the window manager to wait for events

Definition at line 1639 of file TinyWindow.h.

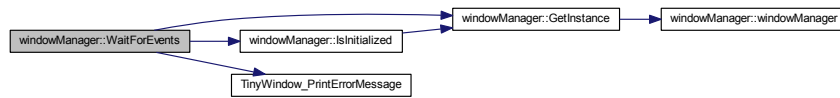
References `GetInstance()`, `instance`, `IsInitialized()`, `NOT_INITIALIZED`, and `TinyWindow_PrintErrorMessage()`.

```

1640     {
1641         if ( GetInstance()->IsInitialized() )
1642         {
1643 #if defined( _WIN32 ) || defined( _WIN64 )
1644             //process even if there aren't any to process
1645             GetMessage(&instance->message, 0, 0, 0);
1646             TranslateMessage (&instance->message);
1647             DispatchMessage (&instance->message);
1648 #elif defined(__linux__)
1649             //even if there aren't any events to process
1650             XNextEvent (instance->currentDisplay, &instance->currentEvent);
1651
1652             XEvent currentEvent = instance->currentEvent;
1653
1654             Linux_ProcessEvents (currentEvent);
1655 #endif
1656         }
1657         else
1658         {
1659             TinyWindow_PrintErrorMessage (
1660                 tinyWindowError_t::NOT_INITIALIZED );
1661         }
1662     }

```

Here is the call graph for this function:



**4.2.3.110** `static bool WindowManager::WindowExists ( unsigned int windowIndex )` `[inline],[static],`  
`[private]`

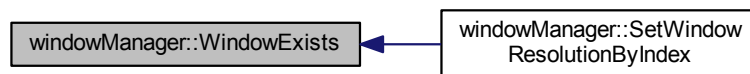
Definition at line 2426 of file TinyWindow.h.

Referenced by SetWindowResolutionByIndex().

```

2427     {
2428         return ( windowIndex <= instance->windowList.size() - 1 );
2429     }
  
```

Here is the caller graph for this function:



**4.2.3.111** `static tinyWindowKeyState_t WindowManager::WindowGetKeyByIndex ( unsigned int windowIndex, unsigned int key )` `[inline],[static]`

Returns the current state of the given key relative to the given window

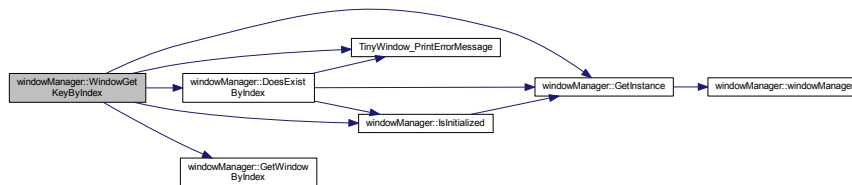
Definition at line 929 of file TinyWindow.h.

References BAD, DoesExistByIndex(), GetInstance(), GetWindowByIndex(), IsInitialized(), WindowManager::window\_t::keys, NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

930     {
931         if ( GetInstance()->IsInitialized() )
932         {
933             if ( DoesExistByIndex( windowIndex ) )
934             {
935                 return GetWindowByIndex( windowIndex )->keys[ key ];
936             }
937             TinyWindow_PrintErrorMessage(
tinyWindowError_t::WINDOW_NOT_FOUND);
938             return tinyWindowKeyState_t::BAD;
939         }
940         TinyWindow_PrintErrorMessage(
tinyWindowError_t::NOT_INITIALIZED );
941         return tinyWindowKeyState_t::BAD;
942     }
  
```

Here is the call graph for this function:



**4.2.3.112** static tinyWindowKeyState\_t windowManager::WindowGetKeyByName ( const char \* *windowName*, unsigned int *key* ) [inline],[static]

Returns the current state of the given key relative to the given window

Definition at line 911 of file TinyWindow.h.

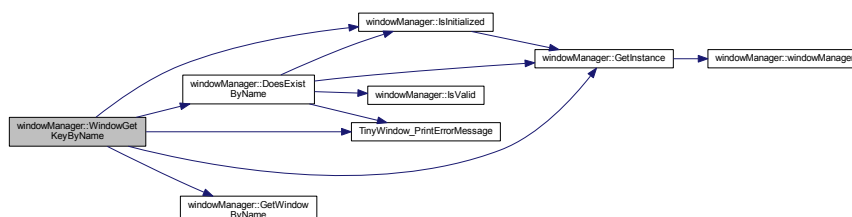
References BAD, DoesExistByName(), GetInstance(), GetWindowByName(), IsInitialized(), windowManager::window\_t::keys, NOT\_INITIALIZED, TinyWindow\_PrintErrorMessage(), and WINDOW\_NOT\_FOUND.

```

912 {
913     if ( GetInstance() -> IsInitialized() )
914     {
915         if ( DoesExistByName( windowName ) )
916         {
917             return GetWindowByName( windowName ) -> keys[ key ];
918         }
919         TinyWindow_PrintErrorMessage(
920             tinyWindowError_t::WINDOW_NOT_FOUND);
921         return tinyWindowKeyState_t::BAD;
922     }
923     TinyWindow_PrintErrorMessage(
924         tinyWindowError_t::NOT_INITIALIZED );
925     return tinyWindowKeyState_t::BAD;
926 }

```

Here is the call graph for this function:



**4.2.3.113** `static bool WindowManager::WindowSwapBuffersByIndex ( unsigned int windowIndex ) [inline],  
[static]`

Swap the draw buffers of the given window

Definition at line 1005 of file TinyWindow.h.

References `WindowManager::window_t::AtomDesktopGeometry`, `DoesExistByIndex()`, `GetInstance()`, `GetWindowByIndex()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SwapBuffers()`, `TinyWindow_PrintErrorMessage()`, and `WINDOW_NOT_FOUND`.

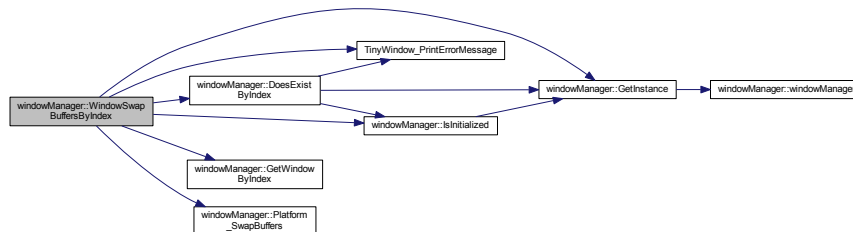
Referenced by `main()`.

```

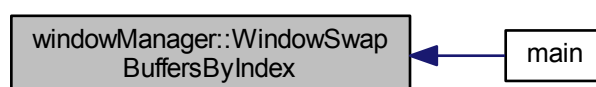
1006     {
1007         if ( GetInstance()->IsInitialized() )
1008         {
1009             if ( DoesExistByIndex( windowIndex ) )
1010             {
1011                 window_t* window = GetWindowByIndex(windowIndex);
1012                 Platform_SwapBuffers(window);
1013                 return true;
1014             }
1015             TinyWindow_PrintErrorMessage(
1016 tinyWindowError_t::WINDOW_NOT_FOUND);
1017             return false;
1018         }
1019         TinyWindow_PrintErrorMessage(
1020 tinyWindowError_t::NOT_INITIALIZED );
1021         return false;
1022     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



**4.2.3.114** `static bool windowManager::WindowSwapBuffersByName ( const char * windowName )` `[inline]`,  
`[static]`

Swap the draw buffers of the given window

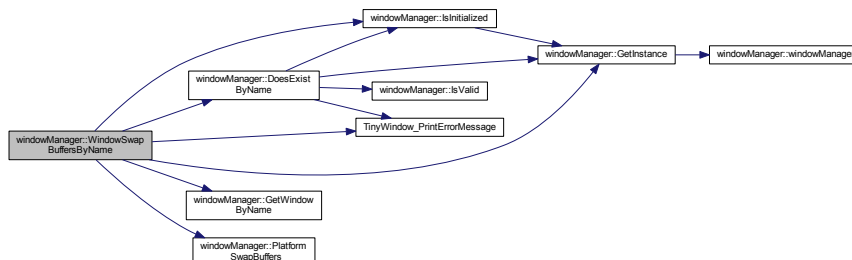
Definition at line 984 of file TinyWindow.h.

References `windowManager::window_t::AtomDesktopGeometry`, `DoesExistByName()`, `GetInstance()`, `GetWindowByName()`, `IsInitialized()`, `NOT_INITIALIZED`, `Platform_SwapBuffers()`, and `TinyWindow_PrintErrorMessage()`.

```

985     {
986         if ( GetInstance()->IsInitialized() )
987         {
988             if ( DoesExistByName( windowName ) )
989             {
990                 window_t* window = GetWindowByName(windowName);
991                 Platform_SwapBuffers(window);
992             }
993             return true;
994         }
995         TinyWindow_PrintErrorMessage(
996             tinyWindowError_t::NOT_INITIALIZED);
997         return false;
998     }
999     TinyWindow_PrintErrorMessage(
1000         tinyWindowError_t::NOT_INITIALIZED );
1001     return false;
1002 }
```

Here is the call graph for this function:



## 4.2.4 Member Data Documentation

**4.2.4.1** `windowManager * windowManager::instance = nullptr` `[static]`, `[private]`

Definition at line 2414 of file TinyWindow.h.

Referenced by `AddWindow()`, `GetInstance()`, `GetMousePositionInScreen()`, `GetScreenResolution()`, `Initialize()`, `PollForEvents()`, `SetMousePositionInScreen()`, `ShutDown()`, and `WaitForEvents()`.

**4.2.4.2** `bool windowManager::isInitialized` `[private]`

Definition at line 2419 of file TinyWindow.h.

Referenced by `Initialize()`, `IsInitialized()`, and `ShutDown()`.

#### 4.2.4.3 unsigned int windowManager::screenMousePosition[2] [private]

Definition at line 2417 of file TinyWindow.h.

Referenced by GetMousePositionInScreen(), and SetMousePositionInScreen().

#### 4.2.4.4 unsigned int windowManager::screenResolution[2] [private]

Definition at line 2416 of file TinyWindow.h.

Referenced by GetScreenResolution(), and Initialize().

#### 4.2.4.5 std::vector< window\_t\*> windowManager::windowList [private]

Definition at line 2413 of file TinyWindow.h.

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

- Include/[TinyWindow.h](#)





## Chapter 5

# File Documentation

### 5.1 Example/CMakeFiles/3.4.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

#### Macros

- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID ""`
- `#define ARCHITECTURE_ID ""`
- `#define DEC(n)`
- `#define HEX(n)`

#### Functions

- `int main (int argc, char *argv[])`

#### Variables

- `char const * info_compiler = "INFO" ":" "compiler[" "" "]"`
- `char const * info_platform = "INFO" ":" "platform[" "" "]"`
- `char const * info_arch = "INFO" ":" "arch[" "" "]"`
- `const char * info_language_dialect_default`

#### 5.1.1 Macro Definition Documentation

##### 5.1.1.1 `#define ARCHITECTURE_ID ""`

Definition at line 411 of file CMakeCXXCompilerId.cpp.

##### 5.1.1.2 `#define COMPILER_ID ""`

Definition at line 248 of file CMakeCXXCompilerId.cpp.

### 5.1.1.3 #define DEC( n )

**Value:**

```
( '0' + ((n) / 10000000) % 10 ), \
( '0' + ((n) / 1000000) % 10 ), \
( '0' + ((n) / 100000) % 10 ), \
( '0' + ((n) / 10000) % 10 ), \
( '0' + ((n) / 1000) % 10 ), \
( '0' + ((n) / 100) % 10 ), \
( '0' + ((n) / 10) % 10 ), \
( '0' + ((n) % 10) )
```

Definition at line 415 of file CMakeCXXCompilerId.cpp.

### 5.1.1.4 #define HEX( n )

**Value:**

```
( '0' + ((n) >> 28 & 0xF) ), \
( '0' + ((n) >> 24 & 0xF) ), \
( '0' + ((n) >> 20 & 0xF) ), \
( '0' + ((n) >> 16 & 0xF) ), \
( '0' + ((n) >> 12 & 0xF) ), \
( '0' + ((n) >> 8 & 0xF) ), \
( '0' + ((n) >> 4 & 0xF) ), \
( '0' + ((n) & 0xF) )
```

Definition at line 426 of file CMakeCXXCompilerId.cpp.

### 5.1.1.5 #define PLATFORM\_ID ""

Definition at line 361 of file CMakeCXXCompilerId.cpp.

### 5.1.1.6 #define STRINGIFY( X ) STRINGIFY\_HELPER(X)

Definition at line 265 of file CMakeCXXCompilerId.cpp.

### 5.1.1.7 #define STRINGIFY\_HELPER( X ) #X

Definition at line 264 of file CMakeCXXCompilerId.cpp.

## 5.1.2 Function Documentation

### 5.1.2.1 int main ( int argc, char \* argv[] )

Definition at line 494 of file CMakeCXXCompilerId.cpp.

References `info_compiler`, `info_language_dialect_default`, and `info_platform`.

```
495 {
496     int require = 0;
497     require += info_compiler[argc];
498     require += info_platform[argc];
499     #ifdef COMPILER_VERSION_MAJOR
500     require += info_version[argc];
501     #endif
502     #ifdef SIMULATE_ID
503     require += info_simulate[argc];
504     #endif
505     #ifdef SIMULATE_VERSION_MAJOR
506     require += info_simulate_version[argc];
507     #endif
508     require += info_language_dialect_default[argc];
509     (void)argv;
510     return require;
511 }
```

### 5.1.3 Variable Documentation

5.1.3.1 `char const* info_arch = "INFO" ":" "arch[" " ""]"`

Definition at line 477 of file CMakeCXXCompilerId.cpp.

5.1.3.2 `char const* info_compiler = "INFO" ":" "compiler[" " ""]"`

Definition at line 255 of file CMakeCXXCompilerId.cpp.

Referenced by `main()`.

5.1.3.3 `const char* info_language_dialect_default`

**Initial value:**

```
= "INFO" ":" "dialect_default["
```

```
    "98"
    "]"
```

Definition at line 482 of file CMakeCXXCompilerId.cpp.

Referenced by `main()`.

5.1.3.4 `char const* info_platform = "INFO" ":" "platform[" " ""]"`

Definition at line 476 of file CMakeCXXCompilerId.cpp.

Referenced by `main()`.

## 5.2 Example/CMakeFiles/feature\_tests.cxx File Reference

### Functions

- `int main` (int argc, char \*\*argv)

### Variables

- `const char features` []

## 5.2.1 Function Documentation

### 5.2.1.1 `int main ( int argc, char ** argv )`

Definition at line 377 of file `feature_tests.cxx`.

References `features`.

```
377 { (void)argv; return features[argc]; }
```

## 5.2.2 Variable Documentation

### 5.2.2.1 `const char features[]`

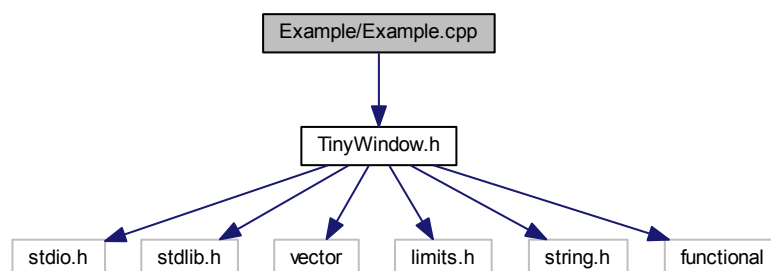
Definition at line 2 of file `feature_tests.cxx`.

Referenced by `main()`.

## 5.3 Example/Example.cpp File Reference

```
#include "TinyWindow.h"
```

Include dependency graph for `Example.cpp`:



## Functions

- void `handleKeyPresses` (unsigned int key, `tinyWindowKeyState_t` keyState)
- int `main` ()

### 5.3.1 Function Documentation

#### 5.3.1.1 void handleKeyPresses ( unsigned int key, tinyWindowKeyState\_t keyState )

Definition at line 3 of file Example.cpp.

References DOWN.

```
4 {  
5     if(keyState == tinyWindowKeyState_t::DOWN)  
6     {  
7         printf("%c \t", key);  
8     }  
9 }
```

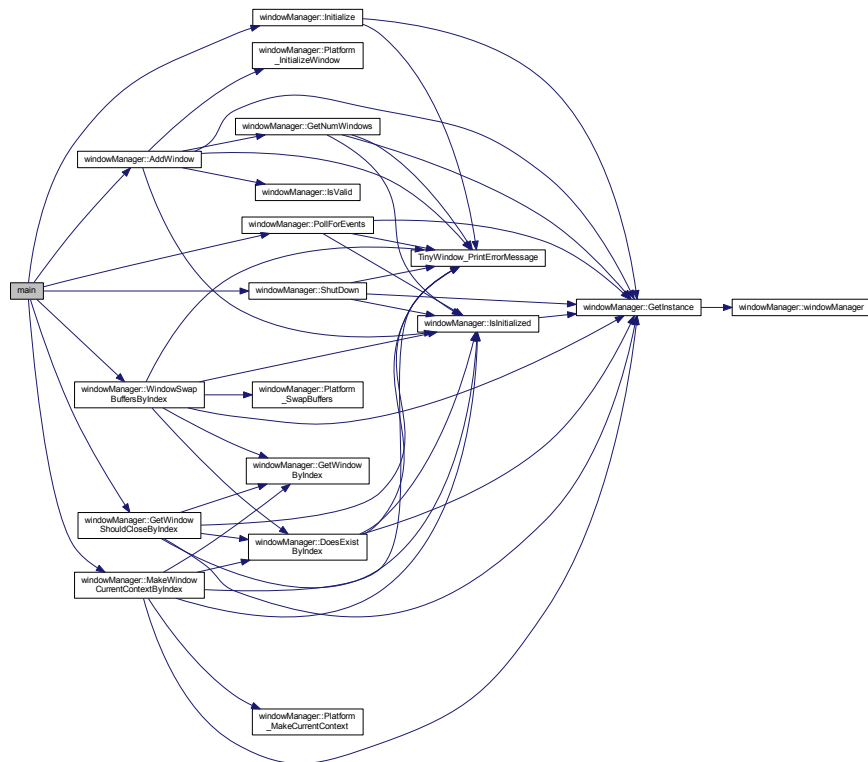
#### 5.3.1.2 int main ( )

Definition at line 11 of file Example.cpp.

References `windowManager::AddWindow()`, `windowManager::GetWindowShouldCloseByIndex()`, `windowManager::Initialize()`, `windowManager::MakeWindowCurrentContextByIndex()`, `windowManager::PollForEvents()`, `windowManager::ShutDown()`, and `windowManager::WindowSwapBuffersByIndex()`.

```
12 {  
13     windowManager::Initialize();  
14  
15     windowManager::AddWindow("Example");  
16     windowManager::SetWindowOnKeyEventByName("Example",  
17         handleKeyPresses);  
18  
19     glClearColor(0.25f, 0.25f, 0.25f, 1.0f);  
20     while (!windowManager::GetWindowShouldCloseByIndex(0))  
21     {  
22         windowManager::PollForEvents(); // or WaitForEvents  
23  
24         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
25         windowManager::MakeWindowCurrentContextByIndex(0);  
26         windowManager::WindowSwapBuffersByIndex(0);  
27     }  
28     windowManager::ShutDown();  
29     return 0;  
30 }
```

Here is the call graph for this function:



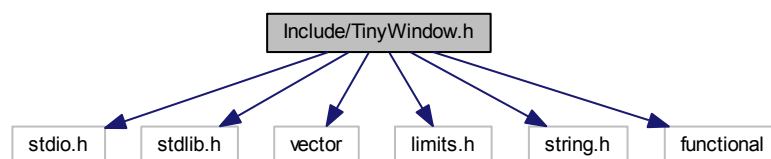
## 5.4 Include/TinyWindow.h File Reference

```

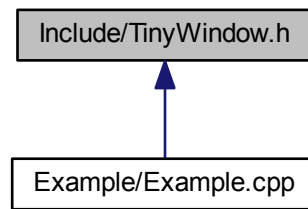
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <limits.h>
#include <string.h>
#include <functional>

```

Include dependency graph for TinyWindow.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [windowManager](#)
- struct [windowManager::window\\_t](#)

## Enumerations

- enum [tinyWindowKeyState\\_t](#) { [tinyWindowKeyState\\_t::UP](#) = 0, [tinyWindowKeyState\\_t::DOWN](#), [tinyWindowKeyState\\_t::BAD](#) = -1 }
- enum [tinyWindowKey\\_t](#) { [KEY\\_ERROR](#) = -1, [KEY\\_FIRST](#) = 256 + 1, [KEY\\_F1](#), [KEY\\_F2](#), [KEY\\_F3](#), [KEY\\_F4](#), [KEY\\_F5](#), [KEY\\_F6](#), [KEY\\_F7](#), [KEY\\_F8](#), [KEY\\_F9](#), [KEY\\_F10](#), [KEY\\_F11](#), [KEY\\_F12](#), [KEY\\_CAPSLOCK](#), [KEY\\_LEFTSHIFT](#), [KEY\\_RIGHTSHIFT](#), [KEY\\_LEFTCONTROL](#), [KEY\\_RIGHTCONTROL](#), [KEY\\_LEFTWINDOW](#), [KEY\\_RIGHTWINDOW](#), [KEY\\_LEFTALT](#), [KEY\\_RIGHTALT](#), [KEY\\_ENTER](#), [KEY\\_PRINTSCREEN](#), [KEY\\_SCROLLLOCK](#), [KEY\\_NUMLOCK](#), [KEY\\_PAUSE](#), [KEY\\_INSERT](#), [KEY\\_HOME](#), [KEY\\_END](#), [KEY\\_PAGEUP](#), [KEY\\_PAGEDOWN](#), [KEY\\_ARROW\\_DOWN](#), [KEY\\_ARROW\\_UP](#), [KEY\\_ARROW\\_LEFT](#), [KEY\\_ARROW\\_RIGHT](#), [KEY\\_KEYPAD\\_DIVIDE](#), [KEY\\_KEYPAD\\_MULTIPLY](#), [KEY\\_KEYPAD\\_SUBTRACT](#), [KEY\\_KEYPAD\\_ADD](#), [KEY\\_KEYPAD\\_ENTER](#), [KEY\\_KEYPAD\\_PERIOD](#), [KEY\\_KEYPAD\\_0](#), [KEY\\_KEYPAD\\_1](#), [KEY\\_KEYPAD\\_2](#), [KEY\\_KEYPAD\\_3](#), [KEY\\_KEYPAD\\_4](#), [KEY\\_KEYPAD\\_5](#), [KEY\\_KEYPAD\\_6](#), [KEY\\_KEYPAD\\_7](#), [KEY\\_KEYPAD\\_8](#), [KEY\\_KEYPAD\\_9](#), [KEY\\_BACKSPACE](#), [KEY\\_TAB](#), [KEY\\_DELETE](#), [KEY\\_ESCAPE](#), [KEY\\_LAST](#) = [KEY\\_ESCAPE](#) }
- enum [tinyWindowButtonState\\_t](#) { [tinyWindowButtonState\\_t::UP](#) = 0, [tinyWindowButtonState\\_t::DOWN](#) }
- enum [tinyWindowMouseButton\\_t](#) { [tinyWindowMouseButton\\_t::LEFT](#) = 0, [tinyWindowMouseButton\\_t::RIGHT](#), [tinyWindowMouseButton\\_t::MIDDLE](#), [tinyWindowMouseButton\\_t::LAST](#) }
- enum [tinyWindowMouseScroll\\_t](#) { [tinyWindowMouseScroll\\_t::DOWN](#) = 0, [tinyWindowMouseScroll\\_t::UP](#) }
- enum [tinyWindowStyle\\_t](#) { [tinyWindowStyle\\_t::BARE](#) = 1, [tinyWindowStyle\\_t::DEFAULT](#), [tinyWindowStyle\\_t::POPUP](#) }
- enum [tinyWindowState\\_t](#) { [tinyWindowState\\_t::NORMAL](#) = 0, [tinyWindowState\\_t::MAXIMIZED](#), [tinyWindowState\\_t::MINIMIZED](#), [tinyWindowState\\_t::FULLSCREEN](#) }
- enum [tinyWindowDecorator\\_t](#) { [DECORATOR\\_TITLEBAR](#) = 0x01, [DECORATOR\\_ICON](#) = 0x02, [DECORATOR\\_BORDER](#) = 0x04, [DECORATOR\\_MINIMIZEBUTTON](#) = 0x08, [DECORATOR\\_MAXIMIZEBUTTON](#) = 0x010, [DECORATOR\\_CLOSEBUTTON](#) = 0x020, [DECORATOR\\_SIZEABLEBORDER](#) = 0x40 }

- enum `tinyWindowError_t` {  
`tinyWindowError_t::TINYWINDOW_ERROR = -1`, `tinyWindowError_t::INVALID_WINDOW_NAME = 0`, `tinyWindowError_t::INVALID_ICON_PATH`, `tinyWindowError_t::INVALID_WINDOW_INDEX`,  
`tinyWindowError_t::INVALID_WINDOW_STATE`, `tinyWindowError_t::INVALID_RESOLUTION`, `tinyWindowError_t::INVALID_CONTEXT`, `tinyWindowError_t::EXISTING_CONTEXT`,  
`tinyWindowError_t::NOT_INITIALIZED`, `tinyWindowError_t::ALREADY_INITIALIZED`, `tinyWindowError_t::INVALID_TITLEBAR`, `tinyWindowError_t::INVALID_CALLBACK`,  
`tinyWindowError_t::WINDOW_NOT_FOUND`, `tinyWindowError_t::INVALID_WINDOWSTYLE`, `tinyWindowError_t::FUNCTION_NOT_IMPLEMENTED`, `tinyWindowError_t::LINUX_CANNOT_CONNECT_X_SERVER`,  
`tinyWindowError_t::LINUX_INVALID_VISUALINFO`, `tinyWindowError_t::LINUX_CANNOT_CREATE_WINDOW`, `tinyWindowError_t::LINUX_FUNCTION_NOT_IMPLEMENTED`, `tinyWindowError_t::WINDOWS_CANNOT_CREATE_WINDOW`,  
`tinyWindowError_t::WINDOWS_CANNOT_INITIALIZE`, `tinyWindowError_t::WINDOWS_FUNCTION_NOT_IMPLEMENTED` }

## Functions

- static void `TinyWindow_PrintErrorMessage` (const `tinyWindowError_t` errorNumber)

## Variables

- `_WIN32 _WIN64` const int `DEFAULT_WINDOW_WIDTH` = 1280
- const int `DEFAULT_WINDOW_HEIGHT` = 720
- const int `LINUX_FUNCTION` = 1
- const int `LINUX_DECORATOR` = 2

## 5.4.1 Enumeration Type Documentation

### 5.4.1.1 enum `tinyWindowButtonState_t` [strong]

#### Enumerator

**UP** The mouse button is currently up

**DOWN** The mouse button is currently down

Definition at line 106 of file `TinyWindow.h`.

```
107 {
108     UP = 0,
109     DOWN
110 };
```



## 5.4.1.2 enum tinyWindowDecorator\_t

Enumerator

**DECORATOR\_TITLEBAR** The title bar decoration of the window  
**DECORATOR\_ICON** The icon decoration of the window  
**DECORATOR\_BORDER** The border decoration of the window  
**DECORATOR\_MINIMIZEBUTTON** The minimize button decoration of the window  
**DECORATOR\_MAXIMIZEBUTTON** The maximize button decoration pf the window  
**DECORATOR\_CLOSEBUTTON** The close button decoration of the window  
**DECORATOR\_SIZEABLEBORDER** The sizable border decoration of the window

Definition at line 141 of file TinyWindow.h.

```

142 {
143     DECORATOR_TITLEBAR = 0x01,
144     DECORATOR_ICON = 0x02,
145     DECORATOR_BORDER = 0x04,
146     DECORATOR_MINIMIZEBUTTON = 0x08,
147     DECORATOR_MAXIMIZEBUTTON = 0x10,
148     DECORATOR_CLOSEBUTTON = 0x20,
149     DECORATOR_SIZEABLEBORDER = 0x40,
150 };

```

## 5.4.1.3 enum tinyWindowError\_t [strong]

Enumerator

**TINYWINDOW\_ERROR**  
**INVALID\_WINDOW\_NAME** If an invalid window name was given  
**INVALID\_ICON\_PATH** If an invalid icon path was given  
**INVALID\_WINDOW\_INDEX** If an invalid window index was given  
**INVALID\_WINDOW\_STATE** If an invalid window state was given  
**INVALID\_RESOLUTION** If an invalid window resolution was given  
**INVALID\_CONTEXT** If the OpenGL context for the window is invalid  
**EXISTING\_CONTEXT** If the window already has an OpenGL context  
**NOT\_INITIALIZED** If the window is being used without being initialized  
**ALREADY\_INITIALIZED** If the window was already initialized  
**INVALID\_TITLEBAR** If the Title-bar text given was invalid  
**INVALID\_CALLBACK** If the given event callback was invalid  
**WINDOW\_NOT\_FOUND** If the window was not found in the window manager  
**INVALID\_WINDOWSTYLE** If the window style gives is invalid  
**FUNCTION\_NOT\_IMPLEMENTED** If the function has not yet been implemented in the current version of the API  
**LINUX\_CANNOT\_CONNECT\_X\_SERVER** Linux: if cannot connect to an X11 server  
**LINUX\_INVALID\_VISUALINFO** Linux: if visual information given was invalid  
**LINUX\_CANNOT\_CREATE\_WINDOW** Linux: when X11 fails to create a new window  
**LINUX\_FUNCTION\_NOT\_IMPLEMENTED** Linux: when the function has not yet been implemented on the Linux in the current version of the API  
**WINDOWS\_CANNOT\_CREATE\_WINDOW** Windows: when Win32 cannot create a window

**WINDOWS\_CANNOT\_INITIALIZE** Windows: when Win32 cannot initialize

**WINDOWS\_FUNCTION\_NOT\_IMPLEMENTED** Windows: when a function has yet to be implemented on the Windows platform in the current version of the API

Definition at line 152 of file TinyWindow.h.

```

153 {
154     TINYWINDOW_ERROR = -1,
155     INVALID_WINDOW_NAME = 0,
156     INVALID_ICON_PATH,
157     INVALID_WINDOW_INDEX,
158     INVALID_WINDOW_STATE,
159     INVALID_RESOLUTION,
160     INVALID_CONTEXT,
161     EXISTING_CONTEXT,
162     NOT_INITIALIZED,
163     ALREADY_INITIALIZED,
164     INVALID_TITLEBAR,
165     INVALID_CALLBACK,
166     WINDOW_NOT_FOUND,
167     INVALID_WINDOWSTYLE,
168     FUNCTION_NOT_IMPLEMENTED,
169     LINUX_CANNOT_CONNECT_X_SERVER,
170     LINUX_INVALID_VISUALINFO,
171     LINUX_CANNOT_CREATE_WINDOW,
172     LINUX_FUNCTION_NOT_IMPLEMENTED,
173     WINDOWS_CANNOT_CREATE_WINDOW,
174     WINDOWS_CANNOT_INITIALIZE,
175     WINDOWS_FUNCTION_NOT_IMPLEMENTED,
176 };

```

#### 5.4.1.4 enum tinyWindowKey\_t

Enumerator

**KEY\_ERROR** The key pressed is considered invalid

**KEY\_FIRST** The first key that is not a char

**KEY\_F1** The F1 key

**KEY\_F2** The F2 key

**KEY\_F3** The F3 key

**KEY\_F4** The F4 key

**KEY\_F5** The F5 key

**KEY\_F6** The F6 key

**KEY\_F7** The F7 key

**KEY\_F8** The F8 key

**KEY\_F9** The F9 key

**KEY\_F10** The F10 key

**KEY\_F11** The F11 key

**KEY\_F12** The F12 key

**KEY\_CAPSLOCK** The CapsLock key

**KEY\_LEFTSHIFT** The left Shift key

**KEY\_RIGHTSHIFT** The right Shift key

**KEY\_LEFTCONTROL** The left Control key

**KEY\_RIGHTCONTROL** The right Control key

**KEY\_LEFTWINDOW** The left Window key

**KEY\_RIGHTWINDOW** The right Window key

**KEY\_LEFTALT** The left Alternate key  
**KEY\_RIGHTALT** The right Alternate key  
**KEY\_ENTER** The Enter/Return key  
**KEY\_PRINTSCREEN** The PrintScreen key  
**KEY\_SCROLLLOCK** The ScrollLock key  
**KEY\_NUMLOCK** The NumLock key  
**KEY\_PAUSE** The pause/break key  
**KEY\_INSERT** The insert key  
**KEY\_HOME** The Home key  
**KEY\_END** The End key  
**KEY\_PAGEUP** The PageUp key  
**KEY\_PAGEDOWN** The PageDown key  
**KEY\_ARROW\_DOWN** The ArrowDown key  
**KEY\_ARROW\_UP** The ArrowUp key  
**KEY\_ARROW\_LEFT** The ArrowLeft key  
**KEY\_ARROW\_RIGHT** The ArrowRight key  
**KEY\_KEYPAD\_DIVIDE** The KeyPad Divide key  
**KEY\_KEYPAD\_MULTIPLY** The Keypad Multiply key  
**KEY\_KEYPAD\_SUBTRACT** The Keypad Subtract key  
**KEY\_KEYPAD\_ADD** The Keypad Add key  
**KEY\_KEYPAD\_ENTER** The Keypad Enter key  
**KEY\_KEYPAD\_PERIOD** The Keypad Period/Decimal key  
**KEY\_KEYPAD\_0** The Keypad 0 key  
**KEY\_KEYPAD\_1** The Keypad 1 key  
**KEY\_KEYPAD\_2** The Keypad 2 key  
**KEY\_KEYPAD\_3** The Keypad 3 key  
**KEY\_KEYPAD\_4** The Keypad 4 key  
**KEY\_KEYPAD\_5** The Keypad 5 key  
**KEY\_KEYPAD\_6** The Keypad 6 key  
**KEY\_KEYPAD\_7** The Keypad 7 key  
**KEY\_KEYPAD\_8** The keypad 8 key  
**KEY\_KEYPAD\_9** The Keypad 9 key  
**KEY\_BACKSPACE** The Backspace key  
**KEY\_TAB** The Tab key  
**KEY\_DELETE** The Delete key  
**KEY\_ESCAPE** The Escape key  
**KEY\_LAST** The last key to be supported

Definition at line 44 of file TinyWindow.h.

```
45 {  
46     KEY_ERROR = -1,  
47     KEY_FIRST = 256 + 1,  
48     KEY_F1,  
49     KEY_F2,  
50     KEY_F3,  
51     KEY_F4,  
52     KEY_F5,  
53     KEY_F6,
```

```

54     KEY_F7,
55     KEY_F8,
56     KEY_F9,
57     KEY_F10,
58     KEY_F11,
59     KEY_F12,
60     KEY_CAPSLOCK,
61     KEY_LEFTSHIFT,
62     KEY_RIGHTSHIFT,
63     KEY_LEFTCONTROL,
64     KEY_RIGHTCONTROL,
65     KEY_LEFTWINDOW,
66     KEY_RIGHTWINDOW,
67     KEY_LEFTALT,
68     KEY_RIGHTALT,
69     KEY_ENTER,
70     KEY_PRINTSCREEN,
71     KEY_SCROLLLOCK,
72     KEY_NUMLOCK,
73     KEY_PAUSE,
74     KEY_INSERT,
75     KEY_HOME,
76     KEY_END,
77     KEY_PAGEUP,
78     KEY_PAGEDOWN,
79     KEY_ARROW_DOWN,
80     KEY_ARROW_UP,
81     KEY_ARROW_LEFT,
82     KEY_ARROW_RIGHT,
83     KEY_KEYPAD_DIVIDE,
84     KEY_KEYPAD_MULTIPLY,
85     KEY_KEYPAD_SUBTRACT,
86     KEY_KEYPAD_ADD,
87     KEY_KEYPAD_ENTER,
88     KEY_KEYPAD_PERIOD,
89     KEY_KEYPAD_0,
90     KEY_KEYPAD_1,
91     KEY_KEYPAD_2,
92     KEY_KEYPAD_3,
93     KEY_KEYPAD_4,
94     KEY_KEYPAD_5,
95     KEY_KEYPAD_6,
96     KEY_KEYPAD_7,
97     KEY_KEYPAD_8,
98     KEY_KEYPAD_9,
99     KEY_BACKSPACE,
100    KEY_TAB,
101    KEY_DELETE,
102    KEY_ESCAPE,
103    KEY_LAST = KEY_ESCAPE,
104 };

```

#### 5.4.1.5 enum tinyWindowKeyState\_t [strong]

##### Enumerator

- UP** The key is currently up
- DOWN** The key is currently down
- BAD** If get key state fails (could not name it ERROR)

Definition at line 37 of file TinyWindow.h.

```

38 {
39     UP = 0,
40     DOWN,
41     BAD = -1,
42 };

```

#### 5.4.1.6 enum tinyWindowMouseButton\_t [strong]

##### Enumerator

- LEFT** The left mouse button
- RIGHT** The right mouse button
- MIDDLE** The middle mouse button / ScrollWheel
- LAST** The last mouse button to be supported

Definition at line 112 of file TinyWindow.h.

```
113 {  
114     LEFT = 0,  
115     RIGHT,  
116     MIDDLE,  
117     LAST,  
118 };
```

#### 5.4.1.7 enum tinyWindowMouseScroll\_t [strong]

##### Enumerator

- DOWN** The mouse wheel up
- UP** The mouse wheel down

Definition at line 120 of file TinyWindow.h.

```
121 {  
122     DOWN = 0,  
123     UP  
124 };
```

#### 5.4.1.8 enum tinyWindowState\_t [strong]

##### Enumerator

- NORMAL** The window is in its default state
- MAXIMIZED** The window is currently maximized
- MINIMIZED** The window is currently minimized
- FULLSCREEN** The window is currently full screen

Definition at line 133 of file TinyWindow.h.

```
134 {  
135     NORMAL = 0,  
136     MAXIMIZED,  
137     MINIMIZED,  
138     FULLSCREEN,  
139 };
```

#### 5.4.1.9 enum tinyWindowStyle\_t [strong]

##### Enumerator

**BARE** The window has no decorators but the window border and title bar

**DEFAULT** The default window style for the respective platform

**POPUP** The window has no decorators

Definition at line 126 of file TinyWindow.h.

```
127 {
128     BARE = 1,
129     DEFAULT,
130     POPUP,
131 };
```

### 5.4.2 Function Documentation

#### 5.4.2.1 static void TinyWindow\_PrintErrorMessage ( const tinyWindowError\_t errorNumber ) [static]

Print out the error associated with the given error number

Definition at line 210 of file TinyWindow.h.

References ALREADY\_INITIALIZED, EXISTING\_CONTEXT, FUNCTION\_NOT\_IMPLEMENTED, INVALID\_CALLBACK, INVALID\_CONTEXT, INVALID\_ICON\_PATH, INVALID\_RESOLUTION, INVALID\_TITLEBAR, INVALID\_WINDOW\_INDEX, INVALID\_WINDOW\_NAME, INVALID\_WINDOW\_STATE, INVALID\_WINDOWSTYLE, LINUX\_CANNOT\_CONNECT\_X\_SERVER, LINUX\_CANNOT\_CREATE\_WINDOW, LINUX\_FUNCTION\_NOT\_IMPLEMENTED, LINUX\_INVALID\_VISUALINFO, NOT\_INITIALIZED, WINDOW\_NOT\_FOUND, WINDOWS\_CANNOT\_CREATE\_WINDOW, and WINDOWS\_FUNCTION\_NOT\_IMPLEMENTED.

Referenced by windowManager::AddWindow(), windowManager::DisableWindowDecoratorByIndex(), windowManager::DisableWindowDecoratorByName(), windowManager::DoesExistByIndex(), windowManager::DoesExistByName(), windowManager::EnableWindowDecoratorsByIndex(), windowManager::EnableWindowDecoratorsByName(), windowManager::FocusWindowByIndex(), windowManager::FocusWindowByName(), windowManager::GetMousePositionInScreen(), windowManager::GetMousePositionInWindowByIndex(), windowManager::GetMousePositionInWindowByName(), windowManager::GetNumWindows(), windowManager::GetScreenResolution(), windowManager::GetWindowIndexByName(), windowManager::GetWindowsFullScreenByIndex(), windowManager::GetWindowsFullScreenByName(), windowManager::GetWindowsInFocusByIndex(), windowManager::GetWindowsInFocusByName(), windowManager::GetWindowsMaximizedByIndex(), windowManager::GetWindowsMaximizedByName(), windowManager::GetWindowsMinimizedByIndex(), windowManager::GetWindowsMinimizedByName(), windowManager::GetWindowNameByIndex(), windowManager::GetWindowPositionByIndex(), windowManager::GetWindowPositionByName(), windowManager::GetWindowResolutionByIndex(), windowManager::GetWindowResolutionByName(), windowManager::GetWindowShouldCloseByIndex(), windowManager::GetWindowShouldCloseByName(), windowManager::Initialize(), windowManager::MakeWindowCurrentContextByIndex(), windowManager::MakeWindowCurrentContextByName(), windowManager::MaximizeWindowByIndex(), windowManager::MaximizeWindowByName(), windowManager::MinimizeWindowByIndex(), windowManager::MinimizeWindowByName(), windowManager::PlatformInitializeGL(), windowManager::Platform\_SetWindowStyle(), windowManager::PollForEvents(), windowManager::RemoveWindowByIndex(), windowManager::RemoveWindowByName(), windowManager::RestoreWindowByIndex(), windowManager::RestoreWindowByName(), windowManager::SetFullScreenByIndex(), windowManager::SetFullScreenByName(), windowManager::SetMousePositionInScreen(), windowManager::SetMousePositionInWindowByIndex(), windowManager::SetMousePositionInWindowByName(), windowManager::SetWindowIconByIndex(), windowManager::SetWindowIconByName(), windowManager::SetWindowOnDestroyedByIndex(),

windowManager::SetWindowOnDestroyedByName(), windowManager::SetWindowOnFocusByIndex(), windowManager::SetWindowOnFocusByName(), windowManager::SetWindowOnKeyEventByIndex(), windowManager::SetWindowOnKeyEventByName(), windowManager::SetWindowOnMaximizedByIndex(), windowManager::SetWindowOnMaximizedByName(), windowManager::SetWindowOnMinimizedByIndex(), windowManager::SetWindowOnMinimizedByName(), windowManager::SetWindowOnMouseButtonEventByIndex(), windowManager::SetWindowOnMouseButtonEventByName(), windowManager::SetWindowOnMouseMoveByIndex(), windowManager::SetWindowOnMouseMoveByName(), windowManager::SetWindowOnMouseWheelEventByIndex(), windowManager::SetWindowOnMouseWheelEventByName(), windowManager::SetWindowOnMovedByIndex(), windowManager::SetWindowOnMovedByName(), windowManager::SetWindowOnResizeByIndex(), windowManager::SetWindowOnResizeByName(), windowManager::SetWindowPositionByIndex(), windowManager::SetWindowPositionByName(), windowManager::SetWindowResolutionByIndex(), windowManager::SetWindowResolutionByName(), windowManager::SetWindowStyleByIndex(), windowManager::SetWindowStyleByName(), windowManager::SetWindowTitleBarByIndex(), windowManager::SetWindowTitleBarByName(), windowManager::ShutDown(), windowManager::WaitForEvents(), windowManager::WindowGetKeyByIndex(), windowManager::WindowGetKeyByName(), windowManager::WindowSwapBuffersByIndex(), and windowManager::WindowSwapBuffersByName().

```

211 {
212     switch ( errorNumber )
213     {
214         case tinyWindowError_t::INVALID_WINDOW_NAME:
215         {
216             printf( "Error: invalid window name \n" );
217             break;
218         }
219
220         case tinyWindowError_t::INVALID_ICON_PATH:
221         {
222             printf( "Error: invalid icon path \n" );
223             break;
224         }
225
226         case tinyWindowError_t::INVALID_WINDOW_INDEX:
227         {
228             printf( "Error: invalid window index \n" );
229             break;
230         }
231
232         case tinyWindowError_t::INVALID_WINDOW_STATE:
233         {
234             printf( "Error: invalid window state \n" );
235             break;
236         }
237
238         case tinyWindowError_t::INVALID_RESOLUTION:
239         {
240             printf( "Error: invalid resolution \n" );
241             break;
242         }
243
244         case tinyWindowError_t::INVALID_CONTEXT:
245         {
246             printf( "Error: Failed to create OpenGL context \n" );
247             break;
248         }
249
250         case tinyWindowError_t::EXISTING_CONTEXT:
251         {
252             printf( "Error: context already created \n" );
253             break;
254         }
255
256         case tinyWindowError_t::NOT_INITIALIZED:
257         {
258             printf( "Error: Window manager not initialized \n" );
259             break;
260         }
261
262         case tinyWindowError_t::ALREADY_INITIALIZED:
263         {
264             printf( "Error: window has already been initialized \n" );
265             break;
266         }
267
268         case tinyWindowError_t::INVALID_TITLEBAR:
269         {
270             printf( "Error: invalid title bar name ( cannot be null or nullptr ) \n" );
271             break;

```

```

272     }
273
274     case tinyWindowError_t::INVALID_CALLBACK:
275     {
276         printf( "Error: invalid event callback given \n" );
277         break;
278     }
279
280     case tinyWindowError_t::WINDOW_NOT_FOUND:
281     {
282         printf( "Error: window was not found \n" );
283         break;
284     }
285
286     case tinyWindowError_t::INVALID_WINDOWSTYLE:
287     {
288         printf( "Error: invalid window style given \n" );
289         break;
290     }
291
292     case tinyWindowError_t::FUNCTION_NOT_IMPLEMENTED:
293     {
294         printf( "Error: I'm sorry but this function has not been implemented yet :( \n" );
295         break;
296     }
297
298     case tinyWindowError_t::LINUX_CANNOT_CONNECT_X_SERVER
299 :
300     {
301         printf( "Error: cannot connect to X server \n" );
302         break;
303     }
304
305     case tinyWindowError_t::LINUX_INVALID_VISUALINFO:
306     {
307         printf( "Error: Invalid visual information given \n" );
308         break;
309     }
310
311     case tinyWindowError_t::LINUX_CANNOT_CREATE_WINDOW:
312     {
313         printf( "Error: failed to create window \n" );
314         break;
315     }
316
317     case tinyWindowError_t::LINUX_FUNCTION_NOT_IMPLEMENTED
318 :
319     {
320         printf( "Error: function not implemented on linux platform yet. sorry :( \n" );
321         break;
322     }
323
324     case tinyWindowError_t::WINDOWS_CANNOT_CREATE_WINDOW
325 :
326     {
327         printf( "Error: failed to create window \n" );
328         break;
329     }
330
331     case tinyWindowError_t::WINDOWS_FUNCTION_NOT_IMPLEMENTED
332 :
333     {
334         printf( "Error: function not implemented on Windows platform yet. sorry ;( \n" );
335         break;
336     }
337
338     default:
339     {
340         printf( "Error: unspecified Error \n" );
341         break;
342     }
343 }

```

## 5.4.3 Variable Documentation

### 5.4.3.1 const int DEFAULT\_WINDOW\_HEIGHT = 720

Definition at line 35 of file TinyWindow.h.



5.4.3.2 `_WIN32_WIN64` `const int DEFAULT_WINDOW_WIDTH = 1280`

Definition at line 34 of file TinyWindow.h.

5.4.3.3 `const int LINUX_DECORATOR = 2`

Definition at line 179 of file TinyWindow.h.

5.4.3.4 `const int LINUX_FUNCTION = 1`

Definition at line 178 of file TinyWindow.h.

## 5.5 README.md File Reference



# Index

- ~windowManager
  - windowManager, [22](#)
- ALREADY\_INITIALIZED
  - TinyWindow.h, [125](#)
- ARCHITECTURE\_ID
  - CMakeCXXCompilerId.cpp, [117](#)
- AddWindow
  - windowManager, [23](#)
- AtomActionClose
  - windowManager::window\_t, [9](#)
- AtomActionMaximizeHorz
  - windowManager::window\_t, [9](#)
- AtomActionMaximizeVert
  - windowManager::window\_t, [9](#)
- AtomActionMinimize
  - windowManager::window\_t, [9](#)
- AtomActionResize
  - windowManager::window\_t, [9](#)
- AtomActionShade
  - windowManager::window\_t, [10](#)
- AtomActive
  - windowManager::window\_t, [10](#)
- AtomAllowedActions
  - windowManager::window\_t, [10](#)
- AtomCardinal
  - windowManager::window\_t, [10](#)
- AtomClose
  - windowManager::window\_t, [10](#)
- AtomDemandsAttention
  - windowManager::window\_t, [10](#)
- AtomDesktopGeometry
  - windowManager::window\_t, [10](#)
- AtomFocused
  - windowManager::window\_t, [11](#)
- AtomFullScreen
  - windowManager::window\_t, [11](#)
- AtomHidden
  - windowManager::window\_t, [11](#)
- AtomHints
  - windowManager::window\_t, [11](#)
- AtomIcon
  - windowManager::window\_t, [11](#)
- AtomMaxHorz
  - windowManager::window\_t, [11](#)
- AtomMaxVert
  - windowManager::window\_t, [12](#)
- AtomState
  - windowManager::window\_t, [12](#)
- AtomWindowType
  - windowManager::window\_t, [12](#)
- AtomWindowTypeDesktop
  - windowManager::window\_t, [12](#)
- AtomWindowTypeNormal
  - windowManager::window\_t, [12](#)
- AtomWindowTypeSplash
  - windowManager::window\_t, [12](#)
- attributes
  - windowManager::window\_t, [12](#)
- BARE
  - TinyWindow.h, [130](#)
- BAD
  - TinyWindow.h, [128](#)
- CMakeCXXCompilerId.cpp
  - ARCHITECTURE\_ID, [117](#)
  - COMPILER\_ID, [117](#)
  - DEC, [117](#)
  - HEX, [118](#)
  - info\_arch, [119](#)
  - info\_compiler, [119](#)
  - info\_language\_dialect\_default, [119](#)
  - info\_platform, [119](#)
  - main, [118](#)
  - PLATFORM\_ID, [118](#)
  - STRINGIFY\_HELPER, [118](#)
  - STRINGIFY, [118](#)
- COMPILER\_ID
  - CMakeCXXCompilerId.cpp, [117](#)
- colorBits
  - windowManager::window\_t, [12](#)
- context
  - windowManager::window\_t, [13](#)
- contextCreated
  - windowManager::window\_t, [13](#)
- currentState
  - windowManager::window\_t, [13](#)
- currentWindowStyle
  - windowManager::window\_t, [13](#)
- DECORATOR\_BORDER
  - TinyWindow.h, [125](#)
- DECORATOR\_CLOSEBUTTON
  - TinyWindow.h, [125](#)
- DECORATOR\_ICON
  - TinyWindow.h, [125](#)
- DECORATOR\_MAXIMIZEBUTTON
  - TinyWindow.h, [125](#)
- DECORATOR\_MINIMIZEBUTTON

- TinyWindow.h, 125
- DECORATOR\_SIZEABLEBORDER
  - TinyWindow.h, 125
- DECORATOR\_TITLEBAR
  - TinyWindow.h, 125
- DEFAULT\_WINDOW\_HEIGHT
  - TinyWindow.h, 132
- DEFAULT\_WINDOW\_WIDTH
  - TinyWindow.h, 132
- DEFAULT
  - TinyWindow.h, 130
- DEC
  - CMakeCXXCompilerId.cpp, 117
- DOWN
  - TinyWindow.h, 124, 128, 129
- decorators
  - windowManager::window\_t, 13
- depthBits
  - windowManager::window\_t, 13
- destroyedEvent
  - windowManager::window\_t, 14
- DisableWindowDecoratorByIndex
  - windowManager, 24
- DisableWindowDecoratorByName
  - windowManager, 25
- DoesExistByIndex
  - windowManager, 25
- DoesExistByName
  - windowManager, 27
- EXISTING\_CONTEXT
  - TinyWindow.h, 125
- EnableWindowDecoratorsByIndex
  - windowManager, 29
- EnableWindowDecoratorsByName
  - windowManager, 30
- Example.cpp
  - handleKeyPresses, 121
  - main, 121
- Example/CMakeFiles/3.4.2/CompilerIdCXX/CMakeCXXCompilerId.cpp, 117
- Example/CMakeFiles/feature\_tests.cxx, 119
- Example/Example.cpp, 120
- FULLSCREEN
  - TinyWindow.h, 129
- FUNCTION\_NOT\_IMPLEMENTED
  - TinyWindow.h, 125
- feature\_tests.cxx
  - features, 120
  - main, 120
- features
  - feature\_tests.cxx, 120
- focusEvent
  - windowManager::window\_t, 14
- FocusWindowByIndex
  - windowManager, 31
- FocusWindowByName
  - windowManager, 32
- GetInstance
  - windowManager, 32
- GetMousePositionInScreen
  - windowManager, 33, 34
- GetMousePositionInWindowByIndex
  - windowManager, 34, 35
- GetMousePositionInWindowByName
  - windowManager, 36
- GetNumWindows
  - windowManager, 37
- GetScreenResolution
  - windowManager, 38, 39
- GetWindowByIndex
  - windowManager, 39
- GetWindowByName
  - windowManager, 41
- GetWindowIndexByName
  - windowManager, 43
- GetWindowIsFullScreenByIndex
  - windowManager, 44
- GetWindowIsFullScreenByName
  - windowManager, 45
- GetWindowIsInFocusByIndex
  - windowManager, 45
- GetWindowIsInFocusByName
  - windowManager, 46
- GetWindowIsMaximizedByIndex
  - windowManager, 47
- GetWindowIsMaximizedByName
  - windowManager, 47
- GetWindowIsMinimizedByIndex
  - windowManager, 48
- GetWindowIsMinimizedByName
  - windowManager, 49
- GetWindowNameByIndex
  - windowManager, 49
- GetWindowPositionByIndex
  - windowManager, 50, 51
- GetWindowPositionByName
  - windowManager, 51, 52
- GetWindowResolutionByIndex
  - windowManager, 53
- GetWindowResolutionByName
  - windowManager, 54, 55
- GetWindowShouldCloseByIndex
  - windowManager, 55
- GetWindowShouldCloseByName
  - windowManager, 56
- HEX
  - CMakeCXXCompilerId.cpp, 118
- handleKeyPresses
  - Example.cpp, 121
- INVALID\_CALLBACK
  - TinyWindow.h, 125
- INVALID\_CONTEXT
  - TinyWindow.h, 125
- INVALID\_ICON\_PATH

- TinyWindow.h, [125](#)
- INVALID\_RESOLUTION
  - TinyWindow.h, [125](#)
- INVALID\_TITLEBAR
  - TinyWindow.h, [125](#)
- INVALID\_WINDOW\_INDEX
  - TinyWindow.h, [125](#)
- INVALID\_WINDOW\_NAME
  - TinyWindow.h, [125](#)
- INVALID\_WINDOW\_STATE
  - TinyWindow.h, [125](#)
- INVALID\_WINDOWSTYLE
  - TinyWindow.h, [125](#)
- iD
  - windowManager::window\_t, [14](#)
- inFocus
  - windowManager::window\_t, [14](#)
- Include/TinyWindow.h, [122](#)
- info\_arch
  - CMakeCXXCompilerId.cpp, [119](#)
- info\_compiler
  - CMakeCXXCompilerId.cpp, [119](#)
- info\_language\_dialect\_default
  - CMakeCXXCompilerId.cpp, [119](#)
- info\_platform
  - CMakeCXXCompilerId.cpp, [119](#)
- Initialize
  - windowManager, [57](#)
- initialized
  - windowManager::window\_t, [14](#)
- instance
  - windowManager, [114](#)
- isCurrentContext
  - windowManager::window\_t, [14](#)
- IsInitialized
  - windowManager, [58](#)
- isInitialized
  - windowManager, [114](#)
- IsValid
  - windowManager, [59](#)
- KEY\_ARROW\_DOWN
  - TinyWindow.h, [127](#)
- KEY\_ARROW\_LEFT
  - TinyWindow.h, [127](#)
- KEY\_ARROW\_RIGHT
  - TinyWindow.h, [127](#)
- KEY\_ARROW\_UP
  - TinyWindow.h, [127](#)
- KEY\_BACKSPACE
  - TinyWindow.h, [127](#)
- KEY\_CAPSLOCK
  - TinyWindow.h, [126](#)
- KEY\_DELETE
  - TinyWindow.h, [127](#)
- KEY\_ENTER
  - TinyWindow.h, [127](#)
- KEY\_END
  - TinyWindow.h, [127](#)
- KEY\_ERROR
  - TinyWindow.h, [126](#)
- KEY\_ESCAPE
  - TinyWindow.h, [127](#)
- KEY\_F1
  - TinyWindow.h, [126](#)
- KEY\_F10
  - TinyWindow.h, [126](#)
- KEY\_F11
  - TinyWindow.h, [126](#)
- KEY\_F12
  - TinyWindow.h, [126](#)
- KEY\_F2
  - TinyWindow.h, [126](#)
- KEY\_F3
  - TinyWindow.h, [126](#)
- KEY\_F4
  - TinyWindow.h, [126](#)
- KEY\_F5
  - TinyWindow.h, [126](#)
- KEY\_F6
  - TinyWindow.h, [126](#)
- KEY\_F7
  - TinyWindow.h, [126](#)
- KEY\_F8
  - TinyWindow.h, [126](#)
- KEY\_F9
  - TinyWindow.h, [126](#)
- KEY\_FIRST
  - TinyWindow.h, [126](#)
- KEY\_HOME
  - TinyWindow.h, [127](#)
- KEY\_INSERT
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_0
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_1
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_2
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_3
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_4
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_5
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_6
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_7
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_8
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_9
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_ADD
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_DIVIDE
  - TinyWindow.h, [127](#)

- KEY\_KEYPAD\_ENTER
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_MULTIPLY
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_PERIOD
  - TinyWindow.h, [127](#)
- KEY\_KEYPAD\_SUBTRACT
  - TinyWindow.h, [127](#)
- KEY\_LAST
  - TinyWindow.h, [127](#)
- KEY\_LEFTALT
  - TinyWindow.h, [126](#)
- KEY\_LEFTCONTROL
  - TinyWindow.h, [126](#)
- KEY\_LEFTSHIFT
  - TinyWindow.h, [126](#)
- KEY\_LEFTWINDOW
  - TinyWindow.h, [126](#)
- KEY\_NUMLOCK
  - TinyWindow.h, [127](#)
- KEY\_PAGEDOWN
  - TinyWindow.h, [127](#)
- KEY\_PAGEUP
  - TinyWindow.h, [127](#)
- KEY\_PAUSE
  - TinyWindow.h, [127](#)
- KEY\_PRINTSCREEN
  - TinyWindow.h, [127](#)
- KEY\_RIGHTALT
  - TinyWindow.h, [127](#)
- KEY\_RIGHTCONTROL
  - TinyWindow.h, [126](#)
- KEY\_RIGHTSHIFT
  - TinyWindow.h, [126](#)
- KEY\_RIGHTWINDOW
  - TinyWindow.h, [126](#)
- KEY\_SCROLLLOCK
  - TinyWindow.h, [127](#)
- KEY\_TAB
  - TinyWindow.h, [127](#)
- keyEvent
  - windowManager::window\_t, [15](#)
- keys
  - windowManager::window\_t, [15](#)
- LAST
  - TinyWindow.h, [129](#)
- LEFT
  - TinyWindow.h, [129](#)
- LINUX\_CANNOT\_CONNECT\_X\_SERVER
  - TinyWindow.h, [125](#)
- LINUX\_CANNOT\_CREATE\_WINDOW
  - TinyWindow.h, [125](#)
- LINUX\_DECORATOR
  - TinyWindow.h, [133](#)
- LINUX\_FUNCTION\_NOT\_IMPLEMENTED
  - TinyWindow.h, [125](#)
- LINUX\_FUNCTION
  - TinyWindow.h, [133](#)
- LINUX\_INVALID\_VISUALINFO
  - TinyWindow.h, [125](#)
- MAXIMIZED
  - TinyWindow.h, [129](#)
- MIDDLE
  - TinyWindow.h, [129](#)
- MINIMIZED
  - TinyWindow.h, [129](#)
- main
  - CMakeCXXCompilerId.cpp, [118](#)
  - Example.cpp, [121](#)
  - feature\_tests.cxx, [120](#)
- MakeWindowCurrentContextByIndex
  - windowManager, [60](#)
- MakeWindowCurrentContextByName
  - windowManager, [61](#)
- MaximizeWindowByIndex
  - windowManager, [62](#)
- MaximizeWindowByName
  - windowManager, [63](#)
- maximizedEvent
  - windowManager::window\_t, [15](#)
- MinimizeWindowByIndex
  - windowManager, [63](#)
- MinimizeWindowByName
  - windowManager, [64](#)
- minimizedEvent
  - windowManager::window\_t, [15](#)
- mouseButton
  - windowManager::window\_t, [15](#)
- mouseButtonEvent
  - windowManager::window\_t, [15](#)
- mouseMoveEvent
  - windowManager::window\_t, [15](#)
- mousePosition
  - windowManager::window\_t, [16](#)
- mouseWheelEvent
  - windowManager::window\_t, [16](#)
- movedEvent
  - windowManager::window\_t, [16](#)
- NORMAL
  - TinyWindow.h, [129](#)
- NOT\_INITIALIZED
  - TinyWindow.h, [125](#)
- name
  - windowManager::window\_t, [16](#)
- PLATFORM\_ID
  - CMakeCXXCompilerId.cpp, [118](#)
- POPUP
  - TinyWindow.h, [130](#)
- Platform\_DisableWindowDecorators
  - windowManager, [65](#)
- Platform\_EnableWindowDecorators
  - windowManager, [67](#)
- Platform\_FocusWindow
  - windowManager, [68](#)

- Platform\_InitializeGL
  - windowManager, [69](#)
- Platform\_InitializeWindow
  - windowManager, [70](#)
- Platform\_MakeCurrentContext
  - windowManager, [71](#)
- Platform\_MaximizeWindow
  - windowManager, [71](#)
- Platform\_MinimizeWindow
  - windowManager, [72](#)
- Platform\_RestoreWindow
  - windowManager, [73](#)
- Platform\_SetFullScreen
  - windowManager, [74](#)
- Platform\_SetMousePositionInWindow
  - windowManager, [75](#)
- Platform\_SetWindowPosition
  - windowManager, [75](#)
- Platform\_SetWindowResolution
  - windowManager, [76](#)
- Platform\_SetWindowStyle
  - windowManager, [77](#)
- Platform\_SetWindowTitleBar
  - windowManager, [79](#)
- Platform\_SwapBuffers
  - windowManager, [79](#)
- PollForEvents
  - windowManager, [80](#)
- position
  - windowManager::window\_t, [16](#)
- README.md, [133](#)
- RIGHT
  - TinyWindow.h, [129](#)
- RemoveWindowByIndex
  - windowManager, [81](#)
- RemoveWindowByName
  - windowManager, [81](#)
- resizeEvent
  - windowManager::window\_t, [16](#)
- resolution
  - windowManager::window\_t, [17](#)
- RestoreWindowByIndex
  - windowManager, [82](#)
- RestoreWindowByName
  - windowManager, [83](#)
- STRINGIFY\_HELPER
  - CMakeCXXCompilerId.cpp, [118](#)
- STRINGIFY
  - CMakeCXXCompilerId.cpp, [118](#)
- screenMousePosition
  - windowManager, [114](#)
- screenResolution
  - windowManager, [115](#)
- setAttributes
  - windowManager::window\_t, [17](#)
- SetFullScreenByIndex
  - windowManager, [83](#)
- SetFullScreenByName
  - windowManager, [84](#)
- SetMousePositionInScreen
  - windowManager, [85](#)
- SetMousePositionInWindowByIndex
  - windowManager, [85](#)
- SetMousePositionInWindowByName
  - windowManager, [86](#)
- SetWindowIconByIndex
  - windowManager, [87](#)
- SetWindowIconByName
  - windowManager, [88](#)
- SetWindowOnDestroyedByIndex
  - windowManager, [88](#)
- SetWindowOnDestroyedByName
  - windowManager, [89](#)
- SetWindowOnFocusByIndex
  - windowManager, [90](#)
- SetWindowOnFocusByName
  - windowManager, [90](#)
- SetWindowOnKeyEventByIndex
  - windowManager, [91](#)
- SetWindowOnKeyEventByName
  - windowManager, [92](#)
- SetWindowOnMaximizedByIndex
  - windowManager, [92](#)
- SetWindowOnMaximizedByName
  - windowManager, [93](#)
- SetWindowOnMinimizedByIndex
  - windowManager, [94](#)
- SetWindowOnMinimizedByName
  - windowManager, [94](#)
- SetWindowOnMouseButtonEventByIndex
  - windowManager, [95](#)
- SetWindowOnMouseButtonEventByName
  - windowManager, [96](#)
- SetWindowOnMouseMoveByIndex
  - windowManager, [97](#)
- SetWindowOnMouseMoveByName
  - windowManager, [97](#)
- SetWindowOnMouseWheelEventByIndex
  - windowManager, [98](#)
- SetWindowOnMouseWheelEventByName
  - windowManager, [99](#)
- SetWindowOnMovedByIndex
  - windowManager, [99](#)
- SetWindowOnMovedByName
  - windowManager, [100](#)
- SetWindowOnResizeByIndex
  - windowManager, [101](#)
- SetWindowOnResizeByName
  - windowManager, [101](#)
- SetWindowPositionByIndex
  - windowManager, [102](#)
- SetWindowPositionByName
  - windowManager, [103](#)
- SetWindowResolutionByIndex
  - windowManager, [104](#)

- SetWindowResolutionByName
  - windowManager, [104](#)
- SetWindowStyleByIndex
  - windowManager, [105](#)
- SetWindowStyleByName
  - windowManager, [106](#)
- SetWindowTitleBarByIndex
  - windowManager, [107](#)
- SetWindowTitleBarByName
  - windowManager, [107](#)
- shouldClose
  - windowManager::window\_t, [17](#)
- ShutDown
  - windowManager, [108](#)
- ShutdownWindow
  - windowManager, [109](#)
- stencilBits
  - windowManager::window\_t, [17](#)
- TINYWINDOW\_ERROR
  - TinyWindow.h, [125](#)
- TinyWindow.h
  - ALREADY\_INITIALIZED, [125](#)
  - BARE, [130](#)
  - BAD, [128](#)
  - DECORATOR\_BORDER, [125](#)
  - DECORATOR\_CLOSEBUTTON, [125](#)
  - DECORATOR\_ICON, [125](#)
  - DECORATOR\_MAXIMIZEBUTTON, [125](#)
  - DECORATOR\_MINIMIZEBUTTON, [125](#)
  - DECORATOR\_SIZEABLEBORDER, [125](#)
  - DECORATOR\_TITLEBAR, [125](#)
  - DEFAULT\_WINDOW\_HEIGHT, [132](#)
  - DEFAULT\_WINDOW\_WIDTH, [132](#)
  - DEFAULT, [130](#)
  - DOWN, [124](#), [128](#), [129](#)
  - EXISTING\_CONTEXT, [125](#)
  - FULLSCREEN, [129](#)
  - FUNCTION\_NOT\_IMPLEMENTED, [125](#)
  - INVALID\_CALLBACK, [125](#)
  - INVALID\_CONTEXT, [125](#)
  - INVALID\_ICON\_PATH, [125](#)
  - INVALID\_RESOLUTION, [125](#)
  - INVALID\_TITLEBAR, [125](#)
  - INVALID\_WINDOW\_INDEX, [125](#)
  - INVALID\_WINDOW\_NAME, [125](#)
  - INVALID\_WINDOW\_STATE, [125](#)
  - INVALID\_WINDOWSTYLE, [125](#)
  - KEY\_ARROW\_DOWN, [127](#)
  - KEY\_ARROW\_LEFT, [127](#)
  - KEY\_ARROW\_RIGHT, [127](#)
  - KEY\_ARROW\_UP, [127](#)
  - KEY\_BACKSPACE, [127](#)
  - KEY\_CAPSLOCK, [126](#)
  - KEY\_DELETE, [127](#)
  - KEY\_ENTER, [127](#)
  - KEY\_END, [127](#)
  - KEY\_ERROR, [126](#)
  - KEY\_ESCAPE, [127](#)
  - KEY\_F1, [126](#)
  - KEY\_F10, [126](#)
  - KEY\_F11, [126](#)
  - KEY\_F12, [126](#)
  - KEY\_F2, [126](#)
  - KEY\_F3, [126](#)
  - KEY\_F4, [126](#)
  - KEY\_F5, [126](#)
  - KEY\_F6, [126](#)
  - KEY\_F7, [126](#)
  - KEY\_F8, [126](#)
  - KEY\_F9, [126](#)
  - KEY\_FIRST, [126](#)
  - KEY\_HOME, [127](#)
  - KEY\_INSERT, [127](#)
  - KEY\_KEYPAD\_0, [127](#)
  - KEY\_KEYPAD\_1, [127](#)
  - KEY\_KEYPAD\_2, [127](#)
  - KEY\_KEYPAD\_3, [127](#)
  - KEY\_KEYPAD\_4, [127](#)
  - KEY\_KEYPAD\_5, [127](#)
  - KEY\_KEYPAD\_6, [127](#)
  - KEY\_KEYPAD\_7, [127](#)
  - KEY\_KEYPAD\_8, [127](#)
  - KEY\_KEYPAD\_9, [127](#)
  - KEY\_KEYPAD\_ADD, [127](#)
  - KEY\_KEYPAD\_DIVIDE, [127](#)
  - KEY\_KEYPAD\_ENTER, [127](#)
  - KEY\_KEYPAD\_MULTIPLY, [127](#)
  - KEY\_KEYPAD\_PERIOD, [127](#)
  - KEY\_KEYPAD\_SUBTRACT, [127](#)
  - KEY\_LAST, [127](#)
  - KEY\_LEFTALT, [126](#)
  - KEY\_LEFTCONTROL, [126](#)
  - KEY\_LEFTSHIFT, [126](#)
  - KEY\_LEFTWINDOW, [126](#)
  - KEY\_NUMLOCK, [127](#)
  - KEY\_PAGEDOWN, [127](#)
  - KEY\_PAGEUP, [127](#)
  - KEY\_PAUSE, [127](#)
  - KEY\_PRINTSCREEN, [127](#)
  - KEY\_RIGHTALT, [127](#)
  - KEY\_RIGHTCONTROL, [126](#)
  - KEY\_RIGHTSHIFT, [126](#)
  - KEY\_RIGHTWINDOW, [126](#)
  - KEY\_SCROLLLOCK, [127](#)
  - KEY\_TAB, [127](#)
  - LAST, [129](#)
  - LEFT, [129](#)
  - LINUX\_CANNOT\_CONNECT\_X\_SERVER, [125](#)
  - LINUX\_CANNOT\_CREATE\_WINDOW, [125](#)
  - LINUX\_DECORATOR, [133](#)
  - LINUX\_FUNCTION\_NOT\_IMPLEMENTED, [125](#)
  - LINUX\_FUNCTION, [133](#)
  - LINUX\_INVALID\_VISUALINFO, [125](#)
  - MAXIMIZED, [129](#)
  - MIDDLE, [129](#)
  - MINIMIZED, [129](#)



- NORMAL, [129](#)
- NOT\_INITIALIZED, [125](#)
- POPUP, [130](#)
- RIGHT, [129](#)
- TINYWINDOW\_ERROR, [125](#)
- TinyWindow\_PrintErrorMessage, [130](#)
- tinyWindowButtonState\_t, [124](#)
- tinyWindowDecorator\_t, [124](#)
- tinyWindowError\_t, [125](#)
- tinyWindowKey\_t, [126](#)
- tinyWindowKeyState\_t, [128](#)
- tinyWindowMouseButton\_t, [128](#)
- tinyWindowMouseScroll\_t, [129](#)
- tinyWindowState\_t, [129](#)
- tinyWindowStyle\_t, [129](#)
- UP, [124](#), [128](#), [129](#)
- WINDOW\_NOT\_FOUND, [125](#)
- WINDOWS\_CANNOT\_CREATE\_WINDOW, [125](#)
- WINDOWS\_CANNOT\_INITIALIZE, [125](#)
- WINDOWS\_FUNCTION\_NOT\_IMPLEMENTED, [126](#)
- TinyWindow\_PrintErrorMessage
  - TinyWindow.h, [130](#)
- tinyWindowButtonState\_t
  - TinyWindow.h, [124](#)
- tinyWindowDecorator\_t
  - TinyWindow.h, [124](#)
- tinyWindowError\_t
  - TinyWindow.h, [125](#)
- tinyWindowKey\_t
  - TinyWindow.h, [126](#)
- tinyWindowKeyState\_t
  - TinyWindow.h, [128](#)
- tinyWindowMouseButton\_t
  - TinyWindow.h, [128](#)
- tinyWindowMouseScroll\_t
  - TinyWindow.h, [129](#)
- tinyWindowState\_t
  - TinyWindow.h, [129](#)
- tinyWindowStyle\_t
  - TinyWindow.h, [129](#)
- UP
  - TinyWindow.h, [124](#), [128](#), [129](#)
- visualInfo
  - windowManager::window\_t, [17](#)
- WINDOW\_NOT\_FOUND
  - TinyWindow.h, [125](#)
- WINDOWS\_CANNOT\_CREATE\_WINDOW
  - TinyWindow.h, [125](#)
- WINDOWS\_CANNOT\_INITIALIZE
  - TinyWindow.h, [125](#)
- WINDOWS\_FUNCTION\_NOT\_IMPLEMENTED
  - TinyWindow.h, [126](#)
- WaitForEvents
  - windowManager, [110](#)
- window\_t
  - windowManager::window\_t, [8](#)
- WindowExists
  - windowManager, [111](#)
- WindowGetKeyByIndex
  - windowManager, [111](#)
- WindowGetKeyByName
  - windowManager, [112](#)
- windowHandle
  - windowManager::window\_t, [17](#)
- windowList
  - windowManager, [115](#)
- windowManager, [18](#)
  - ~windowManager, [22](#)
  - AddWindow, [23](#)
  - DisableWindowDecoratorByIndex, [24](#)
  - DisableWindowDecoratorByName, [25](#)
  - DoesExistByIndex, [25](#)
  - DoesExistByName, [27](#)
  - EnableWindowDecoratorsByIndex, [29](#)
  - EnableWindowDecoratorsByName, [30](#)
  - FocusWindowByIndex, [31](#)
  - FocusWindowByName, [32](#)
  - GetInstance, [32](#)
  - GetMousePositionInScreen, [33](#), [34](#)
  - GetMousePositionInWindowByIndex, [34](#), [35](#)
  - GetMousePositionInWindowByName, [36](#)
  - GetNumWindows, [37](#)
  - GetScreenResolution, [38](#), [39](#)
  - GetWindowByIndex, [39](#)
  - GetWindowByName, [41](#)
  - GetWindowIndexByName, [43](#)
  - GetWindowsFullScreenByIndex, [44](#)
  - GetWindowsFullScreenByName, [45](#)
  - GetWindowsInFocusByIndex, [45](#)
  - GetWindowsInFocusByName, [46](#)
  - GetWindowsMaximizedByIndex, [47](#)
  - GetWindowsMaximizedByName, [47](#)
  - GetWindowsMinimizedByIndex, [48](#)
  - GetWindowsMinimizedByName, [49](#)
  - GetWindowNameByIndex, [49](#)
  - GetWindowPositionByIndex, [50](#), [51](#)
  - GetWindowPositionByName, [51](#), [52](#)
  - GetWindowResolutionByIndex, [53](#)
  - GetWindowResolutionByName, [54](#), [55](#)
  - GetWindowShouldCloseByIndex, [55](#)
  - GetWindowShouldCloseByName, [56](#)
  - Initialize, [57](#)
  - instance, [114](#)
  - IsInitialized, [58](#)
  - isInitialized, [114](#)
  - IsValid, [59](#)
  - MakeWindowCurrentContextByIndex, [60](#)
  - MakeWindowCurrentContextByName, [61](#)
  - MaximizeWindowByIndex, [62](#)
  - MaximizeWindowByName, [63](#)
  - MinimizeWindowByIndex, [63](#)
  - MinimizeWindowByName, [64](#)
  - Platform\_DisableWindowDecorators, [65](#)

- Platform\_EnableWindowDecorators, 67
- Platform\_FocusWindow, 68
- Platform\_InitializeGL, 69
- Platform\_InitializeWindow, 70
- Platform\_MakeCurrentContext, 71
- Platform\_MaximizeWindow, 71
- Platform\_MinimizeWindow, 72
- Platform\_RestoreWindow, 73
- Platform\_SetFullScreen, 74
- Platform\_SetMousePositionInWindow, 75
- Platform\_SetWindowPosition, 75
- Platform\_SetWindowResolution, 76
- Platform\_SetWindowStyle, 77
- Platform\_SetWindowTitleBar, 79
- Platform\_SwapBuffers, 79
- PollForEvents, 80
- RemoveWindowByIndex, 81
- RemoveWindowByName, 81
- RestoreWindowByIndex, 82
- RestoreWindowByName, 83
- screenMousePosition, 114
- screenResolution, 115
- SetFullScreenByIndex, 83
- SetFullScreenByName, 84
- SetMousePositionInScreen, 85
- SetMousePositionInWindowByIndex, 85
- SetMousePositionInWindowByName, 86
- SetWindowIconByIndex, 87
- SetWindowIconByName, 88
- SetWindowOnDestroyedByIndex, 88
- SetWindowOnDestroyedByName, 89
- SetWindowOnFocusByIndex, 90
- SetWindowOnFocusByName, 90
- SetWindowOnKeyEventByIndex, 91
- SetWindowOnKeyEventByName, 92
- SetWindowOnMaximizedByIndex, 92
- SetWindowOnMaximizedByName, 93
- SetWindowOnMinimizedByIndex, 94
- SetWindowOnMinimizedByName, 94
- SetWindowOnMouseButtonEventByIndex, 95
- SetWindowOnMouseButtonEventByName, 96
- SetWindowOnMouseMoveByIndex, 97
- SetWindowOnMouseMoveByName, 97
- SetWindowOnMouseWheelEventByIndex, 98
- SetWindowOnMouseWheelEventByName, 99
- SetWindowOnMovedByIndex, 99
- SetWindowOnMovedByName, 100
- SetWindowOnResizeByIndex, 101
- SetWindowOnResizeByName, 101
- SetWindowPositionByIndex, 102
- SetWindowPositionByName, 103
- SetWindowResolutionByIndex, 104
- SetWindowResolutionByName, 104
- SetWindowStyleByIndex, 105
- SetWindowStyleByName, 106
- SetWindowTitleBarByIndex, 107
- SetWindowTitleBarByName, 107
- ShutDown, 108
- ShutdownWindow, 109
- WaitForEvents, 110
- WindowExists, 111
- WindowGetKeyByIndex, 111
- WindowGetKeyByName, 112
- windowList, 115
- windowManager, 22
- WindowSwapBuffersByIndex, 112
- WindowSwapBuffersByName, 113
- windowManager::window\_t, 7
  - AtomActionClose, 9
  - AtomActionMaximizeHorz, 9
  - AtomActionMaximizeVert, 9
  - AtomActionMinimize, 9
  - AtomActionResize, 9
  - AtomActionShade, 10
  - AtomActive, 10
  - AtomAllowedActions, 10
  - AtomCardinal, 10
  - AtomClose, 10
  - AtomDemandsAttention, 10
  - AtomDesktopGeometry, 10
  - AtomFocused, 11
  - AtomFullScreen, 11
  - AtomHidden, 11
  - AtomHints, 11
  - AtomIcon, 11
  - AtomMaxHorz, 11
  - AtomMaxVert, 12
  - AtomState, 12
  - AtomWindowType, 12
  - AtomWindowTypeDesktop, 12
  - AtomWindowTypeNormal, 12
  - AtomWindowTypeSplash, 12
  - attributes, 12
  - colorBits, 12
  - context, 13
  - contextCreated, 13
  - currentState, 13
  - currentWindowStyle, 13
  - decorators, 13
  - depthBits, 13
  - destroyedEvent, 14
  - focusEvent, 14
  - iD, 14
  - inFocus, 14
  - initialized, 14
  - isCurrentContext, 14
  - keyEvent, 15
  - keys, 15
  - maximizedEvent, 15
  - minimizedEvent, 15
  - mouseButton, 15
  - mouseButtonEvent, 15
  - mouseMoveEvent, 15
  - mousePosition, 16
  - mouseWheelEvent, 16
  - movedEvent, 16

- name, [16](#)
- position, [16](#)
- resizeEvent, [16](#)
- resolution, [17](#)
- setAttributes, [17](#)
- shouldClose, [17](#)
- stencilBits, [17](#)
- visualInfo, [17](#)
- window\_t, [8](#)
- windowHandle, [17](#)
- WindowSwapBuffersByIndex
  - windowManager, [112](#)
- WindowSwapBuffersByName
  - windowManager, [113](#)