

# Foundation window management API

## 0.3

Generated by Doxygen 1.8.6

Wed Jan 21 2015 22:42:21



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	FWindow Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	8
3.1.2	Constructor & Destructor Documentation . . . . .	8
3.1.2.1	FWindow . . . . .	8
3.1.2.2	~FWindow . . . . .	9
3.1.3	Member Function Documentation . . . . .	9
3.1.3.1	DisableDecorator . . . . .	9
3.1.3.2	EnableDecorator . . . . .	9
3.1.3.3	Focus . . . . .	10
3.1.3.4	FullScreen . . . . .	10
3.1.3.5	GetContextHasBeenCreated . . . . .	11
3.1.3.6	GetCurrentState . . . . .	12
3.1.3.7	GetInFocus . . . . .	12
3.1.3.8	GetIsCurrentContext . . . . .	13
3.1.3.9	GetIsFullScreen . . . . .	13
3.1.3.10	GetIsMaximized . . . . .	14
3.1.3.11	GetIsMinimized . . . . .	14
3.1.3.12	GetKeyState . . . . .	14
3.1.3.13	GetMousePosition . . . . .	15
3.1.3.14	GetMousePosition . . . . .	15
3.1.3.15	GetOpenGLExtensions . . . . .	16
3.1.3.16	GetOpenGLVersion . . . . .	16
3.1.3.17	GetPosition . . . . .	17
3.1.3.18	GetPosition . . . . .	17
3.1.3.19	GetResolution . . . . .	18

3.1.3.20	<a href="#">GetResolution</a>	18
3.1.3.21	<a href="#">GetShouldClose</a>	19
3.1.3.22	<a href="#">GetWindowName</a>	19
3.1.3.23	<a href="#">InitGLExtensions</a>	20
3.1.3.24	<a href="#">Initialize</a>	20
3.1.3.25	<a href="#">InitializeEvents</a>	20
3.1.3.26	<a href="#">InitializeGL</a>	21
3.1.3.27	<a href="#">MakeCurrentContext</a>	21
3.1.3.28	<a href="#">Maximize</a>	22
3.1.3.29	<a href="#">Minimize</a>	23
3.1.3.30	<a href="#">PrintOpenGLExtensions</a>	24
3.1.3.31	<a href="#">PrintOpenGLVersion</a>	24
3.1.3.32	<a href="#">Restore</a>	24
3.1.3.33	<a href="#">SetCurrentState</a>	25
3.1.3.34	<a href="#">SetIcon</a>	26
3.1.3.35	<a href="#">SetMousePosition</a>	26
3.1.3.36	<a href="#">SetOnDestroyed</a>	27
3.1.3.37	<a href="#">SetOnFocus</a>	28
3.1.3.38	<a href="#">SetOnKeyEvent</a>	28
3.1.3.39	<a href="#">SetOnMaximized</a>	29
3.1.3.40	<a href="#">SetOnMinimized</a>	29
3.1.3.41	<a href="#">SetOnMouseButtonEvent</a>	30
3.1.3.42	<a href="#">SetOnMouseMove</a>	31
3.1.3.43	<a href="#">SetOnMouseWheelEvent</a>	31
3.1.3.44	<a href="#">SetOnMoved</a>	32
3.1.3.45	<a href="#">SetOnResize</a>	32
3.1.3.46	<a href="#">SetPosition</a>	33
3.1.3.47	<a href="#">SetResolution</a>	33
3.1.3.48	<a href="#">SetStyle</a>	34
3.1.3.49	<a href="#">SetSwapInterval</a>	34
3.1.3.50	<a href="#">SetTitleBar</a>	35
3.1.3.51	<a href="#">Shutdown</a>	36
3.1.3.52	<a href="#">SwapDrawBuffers</a>	37
3.1.4	<a href="#">Friends And Related Function Documentation</a>	37
3.1.4.1	<a href="#">WindowManager</a>	37
3.1.5	<a href="#">Member Data Documentation</a>	37
3.1.5.1	<a href="#">ColourBits</a>	37
3.1.5.2	<a href="#">ContextCreated</a>	37
3.1.5.3	<a href="#">CurrentState</a>	38
3.1.5.4	<a href="#">CurrentSwapInterval</a>	38

3.1.5.5	CurrentWindowStyle	38
3.1.5.6	DepthBits	38
3.1.5.7	DestroyedEvent	38
3.1.5.8	EXTSwapControlSupported	38
3.1.5.9	FocusEvent	38
3.1.5.10	ID	38
3.1.5.11	InFocus	39
3.1.5.12	Initialized	39
3.1.5.13	IsCurrentContext	39
3.1.5.14	KeyEvent	39
3.1.5.15	Keys	39
3.1.5.16	MaximizedEvent	39
3.1.5.17	MESASwapControlSupported	39
3.1.5.18	MinimizedEvent	39
3.1.5.19	MouseButton	40
3.1.5.20	MouseButtonEvent	40
3.1.5.21	MouseMoveEvent	40
3.1.5.22	MousePosition	40
3.1.5.23	MouseWheelEvent	40
3.1.5.24	MovedEvent	40
3.1.5.25	Name	40
3.1.5.26	Position	40
3.1.5.27	ResizeEvent	41
3.1.5.28	Resolution	41
3.1.5.29	SGISwapControlSupported	41
3.1.5.30	ShouldClose	41
3.1.5.31	StencilBits	41
3.2	WindowManager Class Reference	41
3.2.1	Detailed Description	45
3.2.2	Constructor & Destructor Documentation	46
3.2.2.1	WindowManager	46
3.2.2.2	~WindowManager	46
3.2.3	Member Function Documentation	46
3.2.3.1	AddWindow	46
3.2.3.2	DisableWindowDecorator	47
3.2.3.3	DisableWindowDecorator	47
3.2.3.4	DoesExist	48
3.2.3.5	DoesExist	48
3.2.3.6	EnableWindowDecorator	49
3.2.3.7	EnableWindowDecorator	49

3.2.3.8	FocusWindow	50
3.2.3.9	FocusWindow	50
3.2.3.10	GetInstance	51
3.2.3.11	GetMousePositionInScreen	51
3.2.3.12	GetMousePositionInScreen	52
3.2.3.13	GetMousePositionInWindow	52
3.2.3.14	GetMousePositionInWindow	53
3.2.3.15	GetMousePositionInWindow	53
3.2.3.16	GetMousePositionInWindow	54
3.2.3.17	GetNumWindows	54
3.2.3.18	GetScreenResolution	55
3.2.3.19	GetScreenResolution	56
3.2.3.20	GetWindowByIndex	56
3.2.3.21	GetWindowByName	57
3.2.3.22	GetWindowIndex	58
3.2.3.23	GetWindowsFullScreen	59
3.2.3.24	GetWindowsFullScreen	59
3.2.3.25	GetWindowsInFocus	60
3.2.3.26	GetWindowsInFocus	60
3.2.3.27	GetWindowsMaximized	61
3.2.3.28	GetWindowsMaximized	61
3.2.3.29	GetWindowsMinimized	62
3.2.3.30	GetWindowsMinimized	62
3.2.3.31	GetWindowName	63
3.2.3.32	GetWindowPosition	63
3.2.3.33	GetWindowPosition	64
3.2.3.34	GetWindowPosition	64
3.2.3.35	GetWindowPosition	65
3.2.3.36	GetWindowResolution	65
3.2.3.37	GetWindowResolution	66
3.2.3.38	GetWindowResolution	67
3.2.3.39	GetWindowResolution	67
3.2.3.40	GetWindowShouldClose	68
3.2.3.41	GetWindowShouldClose	68
3.2.3.42	Initialize	69
3.2.3.43	IsInitialized	69
3.2.3.44	MaximizeWindow	69
3.2.3.45	MaximizeWindow	70
3.2.3.46	MinimizeWindow	70
3.2.3.47	MinimizeWindow	71

3.2.3.48	PollForEvents	71
3.2.3.49	RemoveWindow	72
3.2.3.50	RestoreWindow	72
3.2.3.51	RestoreWindow	72
3.2.3.52	SetFullScreen	73
3.2.3.53	SetFullScreen	73
3.2.3.54	SetMousePositionInScreen	74
3.2.3.55	SetMousePositionInWindow	74
3.2.3.56	SetMousePositionInWindow	75
3.2.3.57	SetWindowIcon	75
3.2.3.58	SetWindowIcon	75
3.2.3.59	SetWindowOnDestroyed	75
3.2.3.60	SetWindowOnDestroyed	76
3.2.3.61	SetWindowOnFocus	76
3.2.3.62	SetWindowOnFocus	77
3.2.3.63	SetWindowOnKeyEvent	78
3.2.3.64	SetWindowOnKeyEvent	78
3.2.3.65	SetWindowOnMaximized	79
3.2.3.66	SetWindowOnMaximized	79
3.2.3.67	SetWindowOnMinimized	80
3.2.3.68	SetWindowOnMinimized	80
3.2.3.69	SetWindowOnMouseButtonEvent	81
3.2.3.70	SetWindowOnMouseButtonEvent	81
3.2.3.71	SetWindowOnMouseMove	82
3.2.3.72	SetWindowOnMouseMove	82
3.2.3.73	SetWindowOnMouseWheelEvent	83
3.2.3.74	SetWindowOnMouseWheelEvent	83
3.2.3.75	SetWindowOnMoved	84
3.2.3.76	SetWindowOnMoved	84
3.2.3.77	SetWindowOnResize	85
3.2.3.78	SetWindowOnResize	85
3.2.3.79	SetWindowPosition	86
3.2.3.80	SetWindowPosition	86
3.2.3.81	SetWindowResolution	87
3.2.3.82	SetWindowResolution	87
3.2.3.83	SetWindowStyle	88
3.2.3.84	SetWindowStyle	88
3.2.3.85	SetWindowSwapInterval	88
3.2.3.86	SetWindowSwapInterval	89
3.2.3.87	SetWindowTitleBar	89

3.2.3.88	SetWindowTitleBar	90
3.2.3.89	ShutDown	90
3.2.3.90	WindowGetKey	91
3.2.3.91	WindowGetKey	91
3.2.3.92	WindowSwapBuffers	92
3.2.3.93	WindowSwapBuffers	92
3.2.4	Member Data Documentation	93
3.2.4.1	FWindow	93
3.2.4.2	Initialized	93
3.2.4.3	Instance	93
3.2.4.4	ScreenMousePosition	93
3.2.4.5	ScreenResolution	93
3.2.4.6	Windows	93
<b>4</b>	<b>File Documentation</b>	<b>95</b>
4.1	Example.cpp File Reference	95
4.1.1	Function Documentation	95
4.1.1.1	main	95
4.1.1.2	OnWindowKeyPressed	96
4.2	Example.cpp	96
4.3	Window.cpp File Reference	97
4.4	Window.cpp	97
4.5	Window.h File Reference	107
4.6	Window.h	107
4.7	Window_Linux.cpp File Reference	112
4.8	Window_Linux.cpp	112
4.9	Window_Windows.cpp File Reference	119
4.10	Window_Windows.cpp	119
4.11	WindowAPI_Defs.h File Reference	124
4.11.1	Macro Definition Documentation	127
4.11.1.1	DECORATOR_BORDER	127
4.11.1.2	DECORATOR_CLOSEBUTTON	127
4.11.1.3	DECORATOR_HORIZONTALSCROLLBAR	127
4.11.1.4	DECORATOR_ICON	128
4.11.1.5	DECORATOR_MAXIMIZEBUTTON	128
4.11.1.6	DECORATOR_MINIMIZEBUTTON	128
4.11.1.7	DECORATOR_SIZEABLEBORDER	128
4.11.1.8	DECORATOR_TITLEBAR	128
4.11.1.9	DECORATOR_VERTICALSCROLLBAR	128
4.11.1.10	ERROR_ALREADYINITIALIZED	128



4.11.1.11 ERROR_EXISTINGCONTEXT . . . . .	128
4.11.1.12 ERROR_FUNCTIONNOTIMPLEMENTED . . . . .	128
4.11.1.13 ERROR_INVALIDCONTEXT . . . . .	128
4.11.1.14 ERROR_INVALIDEVENT . . . . .	129
4.11.1.15 ERROR_INVALIDRESOLUTION . . . . .	129
4.11.1.16 ERROR_INVALIDTITLEBAR . . . . .	129
4.11.1.17 ERROR_INVALIDWINDOW . . . . .	129
4.11.1.18 ERROR_INVALIDWINDOWINDEX . . . . .	129
4.11.1.19 ERROR_INVALIDWINDOWNAME . . . . .	129
4.11.1.20 ERROR_INVALIDWINDOWSTATE . . . . .	129
4.11.1.21 ERROR_INVALIDWINDOWSTYLE . . . . .	129
4.11.1.22 ERROR_LINUX_CANNOTCONNECTXSERVER . . . . .	129
4.11.1.23 ERROR_LINUX_CANNOTCREATEWINDOW . . . . .	130
4.11.1.24 ERROR_LINUX_FUNCTIONNOTIMPLEMENTED . . . . .	130
4.11.1.25 ERROR_LINUX_INVALIDVISUALINFO . . . . .	130
4.11.1.26 ERROR_NOCONTEXT . . . . .	130
4.11.1.27 ERROR_NOTINITIALIZED . . . . .	130
4.11.1.28 ERROR_WINDOWNOTFOUND . . . . .	130
4.11.1.29 ERROR_WINDOWS_CANNOTCREATEWINDOW . . . . .	130
4.11.1.30 ERROR_WINDOWS_CANNOTINITIALIZE . . . . .	130
4.11.1.31 ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED . . . . .	131
4.11.1.32 FOUNDATION_ERROR . . . . .	131
4.11.1.33 FOUNDATION_OKAY . . . . .	131
4.11.1.34 KEY_ARROW_DOWN . . . . .	131
4.11.1.35 KEY_ARROW_LEFT . . . . .	132
4.11.1.36 KEY_ARROW_RIGHT . . . . .	132
4.11.1.37 KEY_ARROW_UP . . . . .	132
4.11.1.38 KEY_BACKSPACE . . . . .	132
4.11.1.39 KEY_CAPSLOCK . . . . .	132
4.11.1.40 KEY_DELETE . . . . .	132
4.11.1.41 KEY_END . . . . .	132
4.11.1.42 KEY_ENTER . . . . .	132
4.11.1.43 KEY_ERROR . . . . .	132
4.11.1.44 KEY_ESCAPE . . . . .	133
4.11.1.45 KEY_F1 . . . . .	133
4.11.1.46 KEY_F10 . . . . .	133
4.11.1.47 KEY_F11 . . . . .	133
4.11.1.48 KEY_F12 . . . . .	133
4.11.1.49 KEY_F2 . . . . .	133
4.11.1.50 KEY_F3 . . . . .	133

4.11.1.51 KEY_F4 . . . . .	133
4.11.1.52 KEY_F5 . . . . .	133
4.11.1.53 KEY_F6 . . . . .	134
4.11.1.54 KEY_F7 . . . . .	134
4.11.1.55 KEY_F8 . . . . .	134
4.11.1.56 KEY_F9 . . . . .	134
4.11.1.57 KEY_FIRST . . . . .	134
4.11.1.58 KEY_HOME . . . . .	134
4.11.1.59 KEY_INSERT . . . . .	134
4.11.1.60 KEY_KEYPAD_0 . . . . .	134
4.11.1.61 KEY_KEYPAD_1 . . . . .	134
4.11.1.62 KEY_KEYPAD_2 . . . . .	135
4.11.1.63 KEY_KEYPAD_3 . . . . .	135
4.11.1.64 KEY_KEYPAD_4 . . . . .	135
4.11.1.65 KEY_KEYPAD_5 . . . . .	135
4.11.1.66 KEY_KEYPAD_6 . . . . .	135
4.11.1.67 KEY_KEYPAD_7 . . . . .	135
4.11.1.68 KEY_KEYPAD_8 . . . . .	135
4.11.1.69 KEY_KEYPAD_9 . . . . .	135
4.11.1.70 KEY_KEYPAD_ADD . . . . .	135
4.11.1.71 KEY_KEYPAD_DIVIDE . . . . .	136
4.11.1.72 KEY_KEYPAD_ENTER . . . . .	136
4.11.1.73 KEY_KEYPAD_MULTIPLY . . . . .	136
4.11.1.74 KEY_KEYPAD_PERIOD . . . . .	136
4.11.1.75 KEY_KEYPAD_SUBTRACT . . . . .	136
4.11.1.76 KEY_LAST . . . . .	136
4.11.1.77 KEY_LEFTALT . . . . .	136
4.11.1.78 KEY_LEFTCONTROL . . . . .	136
4.11.1.79 KEY_LEFTSHIFT . . . . .	136
4.11.1.80 KEY_LEFTWINDOW . . . . .	137
4.11.1.81 KEY_NUMLOCK . . . . .	137
4.11.1.82 KEY_PAGEDOWN . . . . .	137
4.11.1.83 KEY_PAGEUP . . . . .	137
4.11.1.84 KEY_PAUSE . . . . .	137
4.11.1.85 KEY_PRINTSCREEN . . . . .	137
4.11.1.86 KEY_RIGHTALT . . . . .	137
4.11.1.87 KEY_RIGHTCONTROL . . . . .	137
4.11.1.88 KEY_RIGHTSHIFT . . . . .	137
4.11.1.89 KEY_RIGHTWINDOW . . . . .	138
4.11.1.90 KEY_SCROLLLOCK . . . . .	138

4.11.1.91 KEY_TAB	138
4.11.1.92 KEYSTATE_DOWN	138
4.11.1.93 KEYSTATE_UP	138
4.11.1.94 LINUX_DECORATOR	138
4.11.1.95 LINUX_DECORATOR_BORDER	138
4.11.1.96 LINUX_DECORATOR_CLOSE	138
4.11.1.97 LINUX_DECORATOR_MAXIMIZE	138
4.11.1.98 LINUX_DECORATOR_MINIMIZE	138
4.11.1.99 LINUX_DECORATOR_MOVE	139
4.11.1.100 LINUX_FUNCTION	139
4.11.1.101 MOUSE_BUTTONDOWN	139
4.11.1.102 MOUSE_BUTTONUP	139
4.11.1.103 MOUSE_LAST	139
4.11.1.104 MOUSE_LEFTBUTTON	139
4.11.1.105 MOUSE_MIDDLEBUTTON	139
4.11.1.106 MOUSE_RIGHTBUTTON	139
4.11.1.107 MOUSE_SCROLL_DOWN	139
4.11.1.108 MOUSE_SCROLL_UP	139
4.11.1.109 WARNING_NOGLEXTEENSIONS	140
4.11.1.110 WARNING_NOTCURRENTCONTEXT	140
4.11.1.111 WINDOWSTATE_FULLSCREEN	140
4.11.1.112 WINDOWSTATE_MAXIMIZED	140
4.11.1.113 WINDOWSTATE_MINIMIZED	140
4.11.1.114 WINDOWSTATE_NORMAL	140
4.11.1.115 WINDOWSTYLE_BARE	140
4.11.1.116 WINDOWSTYLE_DEFAULT	140
4.11.1.117 WINDOWSTYLE_POPUP	141
4.11.2 Typedef Documentation	141
4.11.2.1 OnDestroyedEvent	141
4.11.2.2 OnFocusEvent	141
4.11.2.3 OnKeyEvent	141
4.11.2.4 OnMaximizedEvent	141
4.11.2.5 OnMinimizedEvent	141
4.11.2.6 OnMouseButtonEvent	141
4.11.2.7 OnMouseMoveEvent	141
4.11.2.8 OnMouseWheelEvent	141
4.11.2.9 OnMovedEvent	142
4.11.2.10 OnResizeEvent	142
4.11.3 Function Documentation	142
4.11.3.1 IsValidDestroyedEvent	142

4.11.3.2	<a href="#">IsValidFocusEvent</a>	142
4.11.3.3	<a href="#">IsValidKeyEvent</a>	142
4.11.3.4	<a href="#">IsValidMouseMoveEvent</a>	142
4.11.3.5	<a href="#">IsValidMouseWheelEvent</a>	143
4.11.3.6	<a href="#">IsValidMovedEvent</a>	143
4.11.3.7	<a href="#">IsValidString</a>	143
4.11.3.8	<a href="#">PrintErrorMessage</a>	143
4.11.3.9	<a href="#">PrintWarningMessage</a>	145
4.12	<a href="#">WindowAPI_Defs.h</a>	146
4.13	<a href="#">WindowManager.cpp</a> File Reference	150
4.14	<a href="#">WindowManager.cpp</a>	150
4.15	<a href="#">WindowManager.h</a> File Reference	164
4.16	<a href="#">WindowManager.h</a>	165
4.17	<a href="#">WindowManager_Linux.cpp</a> File Reference	168
4.18	<a href="#">WindowManager_Linux.cpp</a>	168
4.19	<a href="#">WindowManager_Windows.cpp</a> File Reference	180
4.20	<a href="#">WindowManager_Windows.cpp</a>	180
<b>Index</b>		<b>190</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">FWindow</a>	.....	5
<a href="#">WindowManager</a>	.....	41



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Example.cpp</a>	96
<a href="#">Window.cpp</a>	97
<a href="#">Window.h</a>	107
<a href="#">Window_Linux.cpp</a>	112
<a href="#">Window_Windows.cpp</a>	119
<a href="#">WindowAPI_Defs.h</a>	146
<a href="#">WindowManager.cpp</a>	150
<a href="#">WindowManager.h</a>	165
<a href="#">WindowManager_Linux.cpp</a>	168
<a href="#">WindowManager_Windows.cpp</a>	180





## Chapter 3

# Class Documentation

### 3.1 FWindow Class Reference

```
#include <Window.h>
```

#### Public Member Functions

- **FWindow** (const char \*WindowName, GLuint Width=1280, GLuint Height=720, GLuint ColourBits=8, GLuint DepthBits=24, GLuint StencilBits=8)  
*Constructor.*
- **~FWindow** ()  
*FWindow Destructor.*
- GLboolean **Initialize** ()  
*Initializes this object.*
- GLboolean **Shutdown** ()  
*Shuts down this object and frees any resources it is using.*
- GLboolean **GetResolution** (GLuint &Width, GLuint &Height)  
*Gets the resolution of the window by setting width and height.*
- GLuint \* **GetResolution** ()  
*Gets the resolution of the window.*
- GLboolean **SetResolution** (GLuint Width, GLuint Height)  
*Sets the resolution of the window.*
- GLboolean **GetMousePosition** (GLuint &X, GLuint &Y)  
*Gets mouse position relative to the window coordinates by setting X and Y.*
- GLuint \* **GetMousePosition** ()  
*Gets mouse position relative to the window as an array.*
- GLboolean **SetMousePosition** (GLuint X, GLuint Y)  
*Sets mouse position.*
- GLboolean **GetPosition** (GLuint &X, GLuint &Y)  
*Gets a position.*
- GLuint \* **GetPosition** ()  
*Gets the position.*
- GLboolean **SetPosition** (GLuint X, GLuint Y)  
*Sets a position.*
- GLuint **GetCurrentState** ()  
*Gets current state.*
- GLboolean **SetCurrentState** (GLuint NewState)

- Sets the current state of the window.*

  - GLboolean [GetKeyState](#) (GLuint Key)

*Gets key state.*
- GLboolean [GetShouldClose](#) ()

*Gets should close.*
- GLboolean [SwapDrawBuffers](#) ()

*Swap draw buffers.*
- GLboolean [FullScreen](#) (GLboolean NewState)

*toggle the full screen mode for the window.*
- GLboolean [GetIsFullScreen](#) ()

*returns whether the window is in full screen mode.*
- GLboolean [Minimize](#) (GLboolean NewState)

*set the window to be minimized depending on NewState.*
- GLboolean [GetIsMinimized](#) ()

*Gets whether the window is minimized.*
- GLboolean [Maximize](#) (GLboolean NewState)

*Maximizes the window depending on New state.*
- GLboolean [GetIsMaximized](#) ()

*Gets whether the window is maximized.*
- GLboolean [Restore](#) ()

*Restores the window to its default setting.*
- GLboolean [InitializeGL](#) ()

*Initializes OpenGL for this window.*
- const char \* [GetWindowName](#) ()

*Gets window name.*
- GLboolean [SetTitleBar](#) (const char \*NewText)

*Sets title bar.*
- GLboolean [SetStyle](#) (GLuint WindowType)
- GLboolean [SetIcon](#) (const char \*Icon, GLuint Width, GLuint Height)
- GLboolean [MakeCurrentContext](#) ()

*Makes the window be the current OpenGL context to be drawn to. NOTE: Does not change the IsCurrentContext variable for other windows.*
- GLboolean [GetIsCurrentContext](#) ()

*Gets is current context.*
- GLboolean [GetContextHasBeenCreated](#) ()

*Gets context has been created.*
- GLboolean [GetInFocus](#) ()

*Gets whether the window is in event focus.*
- GLboolean [Focus](#) (GLboolean NewState)

*put the window into event focus.*
- GLboolean [SetSwapInterval](#) (GLint SwapSetting)

*Sets swap interval(V-sync).*
- GLboolean [SetOnKeyEvent](#) (OnKeyEvent OnKey)

*Sets on key event.*
- GLboolean [SetOnMouseButtonEvent](#) (OnMouseButtonEvent OnMouseButton)

*Sets on mouse button event.*
- GLboolean [SetOnMouseWheelEvent](#) (OnMouseWheelEvent OnMouseWheel)

*Sets on mouse wheel event.*
- GLboolean [SetOnDestroyed](#) (OnDestroyedEvent OnDestroyed)

*Sets on destroyed.*
- GLboolean [SetOnMaximized](#) (OnMaximizedEvent OnMaximized)

- Sets on maximized.*
- GLboolean [SetOnMinimized](#) ([OnMinimizedEvent](#) OnMinimized)
- Sets on minimized.*
- GLboolean [SetOnFocus](#) ([OnFocusEvent](#) OnFocus)
- Sets on focus.*
- GLboolean [SetOnMoved](#) ([OnMovedEvent](#) OnMoved)
- Sets on moved.*
- GLboolean [SetOnResize](#) ([OnResizeEvent](#) OnResize)
- Sets on resize.*
- GLboolean [SetOnMouseMove](#) ([OnMouseMoveEvent](#) OnMouseMove)
- Sets on mouse move.*
- GLboolean [PrintOpenGLVersion](#) ()
- Print open gl version.*
- const char \* [GetOpenGLVersion](#) ()
- GLboolean [PrintOpenGLExtensions](#) ()
- const char \* [GetOpenGLExtensions](#) ()
- Gets open gl extensions.*
- GLboolean [EnableDecorator](#) (GLbitfield Decorator)
- GLboolean [DisableDecorator](#) (GLbitfield Decorator)

### Private Member Functions

- void [InitializeEvents](#) ()
- Initializes the events.*
- void [InitGLExtensions](#) ()
- Initializes the OpenGL extensions for this window.*

### Private Attributes

- const char \* [Name](#)
- GLuint [ID](#)
- GLint [ColourBits](#)
- GLint [DepthBits](#)
- GLint [StencilBits](#)
- GLboolean [Keys](#) [256+1+54]
- GLboolean [MouseButton](#) [2+1]
- GLuint [Resolution](#) [2]
- GLuint [Position](#) [2]
- GLuint [MousePosition](#) [2]
- GLboolean [ShouldClose](#)
- GLboolean [InFocus](#)
- GLboolean [Initialized](#)
- GLboolean [ContextCreated](#)
- GLboolean [IsCurrentContext](#)
- GLuint [CurrentState](#)
- GLuint [CurrentSwapInterval](#)
- GLbitfield [CurrentWindowStyle](#)
- [OnKeyEvent](#) KeyEvent
- [OnMouseButtonEvent](#) MouseButtonEvent
- [OnMouseWheelEvent](#) MouseWheelEvent
- [OnDestroyedEvent](#) DestroyedEvent
- [OnMaximizedEvent](#) MaximizedEvent

- [OnMinimizedEvent](#) [MinimizedEvent](#)
- [OnFocusEvent](#) [FocusEvent](#)
- [OnMovedEvent](#) [MovedEvent](#)
- [OnResizeEvent](#) [ResizeEvent](#)
- [OnMouseMoveEvent](#) [MouseMoveEvent](#)
- GLboolean [EXTSwapControlSupported](#)
- GLboolean [SGISwapControlSupported](#)
- GLboolean [MESASwapControlSupported](#)

## Friends

- class [WindowManager](#)

### 3.1.1 Detailed Description

Definition at line 22 of file [Window.h](#).

### 3.1.2 Constructor & Destructor Documentation

**3.1.2.1** [FWindow::FWindow](#) ( [const char \\*](#) *WindowName*, [GLuint](#) *Width* = 1280, [GLuint](#) *Height* = 720, [GLuint](#) *ColourBits* = 8, [GLuint](#) *DepthBits* = 24, [GLuint](#) *StencilBits* = 8 )

Constructor.

Author

Ziyad Barakat

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>Width</i>	The width.
<i>Height</i>	The height.
<i>ColourBits</i>	The colour bits.
<i>DepthBits</i>	The depth bits.
<i>StencilBits</i>	The stencil bits.

Definition at line 31 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [ERROR\\_INVALIDWINDOWNAME](#), [EXTSwapControlSupported](#), [InitializeEvents\(\)](#), [IsCurrentContext](#), [IsValidString\(\)](#), [MESASwapControlSupported](#), [Position](#), [PrintErrorMessage\(\)](#), [Resolution](#), [SGISwapControlSupported](#), [ShouldClose](#), and [WINDOWSTATE\\_NORMAL](#).

```

00036         :
00037         Name(WindowName),
00038         ColourBits(ColourBits),
00039         DepthBits(DepthBits),
00040         StencilBits(StencilBits)
00041 {
00042     Resolution[0] = Width;
00043     Resolution[1] = Height;
00044     Position[0] = 0;
00045     Position[1] = 0;
00046     ShouldClose = GL_FALSE;
00047     EXTSwapControlSupported = GL_FALSE;
00048     SGISwapControlSupported = GL_FALSE;
00049     MESASwapControlSupported = GL_FALSE;

```

```

00050
00051     if (!IsValidString(WindowName))
00052     {
00053         PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
00054         exit(0);
00055     }
00056
00057     InitializeEvents();
00058
00059     CurrentState = WINDOWSTATE_NORMAL;
00060     ContextCreated = GL_FALSE;
00061     IsCurrentContext = GL_FALSE;
00062 }

```

### 3.1.2.2 FWindow::~FWindow ( )

[FWindow](#) Destructor.

Author

Ziyad

Date

29/11/2014

Definition at line 73 of file [Window.cpp](#).

References [Shutdown\(\)](#).

```

00074 {
00075     Shutdown();
00076 }

```

## 3.1.3 Member Function Documentation

### 3.1.3.1 GLboolean FWindow::DisableDecorator ( GLbitfield *Decorator* )

Definition at line 1382 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::DisableWindowDecorator\(\)](#).

```

01383 {
01384     if (ContextCreated)
01385     {
01386         #if defined(CURRENT_OS_WINDOWS)
01387             Windows_DisableDecorator(Decorator);
01388         #endif
01389
01390         #if defined(CURRENT_OS_LINUX)
01391             Linux_DisableDecorator(Decorator);
01392         #endif
01393         return FOUNDATION_OKAY;
01394     }
01395
01396     PrintErrorMessage(ERROR_NOCONTEXT);
01397     return FOUNDATION_ERROR;
01398 }

```

### 3.1.3.2 GLboolean FWindow::EnableDecorator ( GLbitfield *Decorator* )

Definition at line 1364 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::EnableWindowDecorator\(\)](#).

```

01365 {
01366     if (ContextCreated)
01367     {
01368         #if defined(CURRENT_OS_WINDOWS)
01369             Windows_EnableDecorator (Decorator);
01370         #endif
01371
01372         #if defined(CURRENT_OS_LINUX)
01373             Linux_EnableDecorator (Decorator);
01374         #endif
01375
01376         return FOUNDATION_OKAY;
01377     }
01378     PrintErrorMessage (ERROR_NOCONTEXT);
01379     return FOUNDATION_ERROR;
01380 }

```

### 3.1.3.3 GLboolean FWindow::Focus ( GLboolean *NewState* )

put the window into event focus.

Author

Ziyad

Date

29/11/2014

Parameters

<i>NewState</i>	whether to put the window into event focus.
-----------------	---

Definition at line 1093 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [InFocus](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::FocusWindow\(\)](#).

```

01094 {
01095     if (ContextCreated)
01096     {
01097         InFocus = ShouldBeInFocus;
01098
01099         #if defined(CURRENT_OS_LINUX)
01100             Linux_Focus (ShouldBeInFocus);
01101         #endif
01102
01103         #if defined(CURRENT_OS_WINDOWS)
01104             Windows_Focus ();
01105         #endif
01106
01107         return FOUNDATION_OKAY;
01108     }
01109
01110     PrintErrorMessage (ERROR_NOCONTEXT);
01111     return FOUNDATION_ERROR;
01112 }

```

### 3.1.3.4 GLboolean FWindow::FullScreen ( GLboolean *ShouldBeFullscreen* )

toggle the full screen mode for the window.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>ShouldBeFullscreen</i>	whether the window should be in full screen mode.
---------------------------	---

**Returns**

A GLboolean.

Definition at line 378 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_OKAY](#), [PrintErrorMessage\(\)](#), [WINDOWSTATE\\_FULLSCREEN](#), and [WINDOWSTATE\\_NORMAL](#).

Referenced by [WindowManager::MaximizeWindow\(\)](#), [WindowManager::MinimizeWindow\(\)](#), [Restore\(\)](#), [SetCurrentState\(\)](#), and [WindowManager::SetFullScreen\(\)](#).

```

00379 {
00380     if (ContextCreated)
00381     {
00382         if (ShouldBeFullscreen)
00383         {
00384             CurrentState = WINDOWSTATE_FULLSCREEN;
00385         }
00386         else
00387         {
00388             CurrentState = WINDOWSTATE_NORMAL;
00389         }
00390     }
00391
00392     #if defined(CURRENT_OS_LINUX)
00393         Linux_FullScreen (ShouldBeFullscreen);
00394     #endif
00395
00396     #if defined(CURRENT_OS_WINDOWS)
00397         Windows_FullScreen();
00398     #endif
00399
00400     return FOUNDATION_OKAY;
00401 }
00402
00403 PrintErrorMessage(ERROR_NOCONTEXT);
00404 return FOUNDATION_OKAY;
00405 }
```

**3.1.3.5 GLboolean FWindow::GetContextHasBeenCreated ( )**

Gets context has been created.

**Author**

Ziyad

**Date**

3/01/2015

**Returns**

The context has been created.

Definition at line 971 of file [Window.cpp](#).

References [ContextCreated](#).

```
00972 {  
00973     return ContextCreated;  
00974 }
```

**3.1.3.6 GLuint FWindow::GetCurrentState ( )**

Gets current state.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

The current state.

Definition at line 283 of file [Window.cpp](#).

References [CurrentState](#).

```
00284 {  
00285     return CurrentState;  
00286 }
```

**3.1.3.7 GLboolean FWindow::GetInFocus ( )**

Gets whether the window is in event focus.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

The in focus.

Definition at line 1077 of file [Window.cpp](#).

References [InFocus](#).

Referenced by [WindowManager::GetWindowsInFocus\(\)](#).

```
01078 {  
01079     return InFocus;  
01080 }
```



### 3.1.3.8 GLboolean FWindow::GetIsCurrentContext ( )

Gets is current context.

#### Author

Ziyad

#### Date

3/01/2015

#### Returns

The is current context.

Definition at line 950 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [IsCurrentContext](#), and [PrintErrorMessage\(\)](#).

```
00951 {  
00952     if (ContextCreated)  
00953     {  
00954         return IsCurrentContext;  
00955     }  
00956     PrintErrorMessage (ERROR_NOCONTEXT);  
00957     return GL_FALSE;  
00958 }
```

### 3.1.3.9 GLboolean FWindow::GetIsFullScreen ( )

returns whether the window is in full screen mode.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

Whether the window is currently in full screen mode.

Definition at line 354 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [PrintErrorMessage\(\)](#), and [WINDOWSTATE\\_FULLSCREEN](#).

Referenced by [WindowManager::GetWindowIsFullScreen\(\)](#).

```
00355 {  
00356     if (ContextCreated)  
00357     {  
00358         return (CurrentState == WINDOWSTATE_FULLSCREEN);  
00359     }  
00360     PrintErrorMessage (ERROR_NOCONTEXT);  
00361     return FOUNDATION_ERROR;  
00362 }  
00363 }
```

### 3.1.3.10 GLboolean FWindow::GetIsMaximized ( )

Gets whether the window is maximized.

Author

Ziyad

Date

29/11/2014

Returns

whether the window is currently minimized.

Definition at line 475 of file [Window.cpp](#).

References [CurrentState](#), and [WINDOWSTATE\\_MAXIMIZED](#).

Referenced by [WindowManager::GetWindowsMaximized\(\)](#).

```
00476 {  
00477     return (CurrentState == WINDOWSTATE_MAXIMIZED) ;  
00478 }
```

### 3.1.3.11 GLboolean FWindow::GetIsMinimized ( )

Gets whether the window is minimized.

Author

Ziyad

Date

29/11/2014

Returns

whether the window is currently minimized.

Definition at line 418 of file [Window.cpp](#).

References [CurrentState](#), and [WINDOWSTATE\\_MINIMIZED](#).

Referenced by [WindowManager::GetWindowsMinimized\(\)](#).

```
00419 {  
00420     return (CurrentState == WINDOWSTATE_MINIMIZED) ;  
00421 }
```

### 3.1.3.12 GLboolean FWindow::GetKeyState ( GLuint Key )

Gets key state.

Author

Ziyad

Date

29/11/2014

## Parameters

<i>Key</i>	The key.
------------	----------

## Returns

The key state.

Definition at line 182 of file [Window.cpp](#).

References [Keys](#).

Referenced by [WindowManager::WindowGetKey\(\)](#).

```
00183 {
00184     return Keys[Key];
00185 }
```

## 3.1.3.13 GLboolean FWindow::GetMousePosition ( GLuint &amp; X, GLuint &amp; Y )

Gets mouse position relative to the window coordinates by setting X and Y.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>in, out</i>	<i>X</i>	The X position of the mouse.
<i>in, out</i>	<i>Y</i>	The Y position of the mouse.

< .

Definition at line 667 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [MousePosition](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::GetMousePositionInWindow\(\)](#).

```
00668 {
00669     if (ContextCreated)
00670     {
00671         X = MousePosition[0];
00672         Y = MousePosition[1];
00673         return FOUNDATION_OKAY;
00674     }
00675     PrintErrorMessage(ERROR_NOCONTEXT);
00676     return FOUNDATION_ERROR;
00677 }
00678 }
```

## 3.1.3.14 GLuint \* FWindow::GetMousePosition ( )

Gets mouse position relative to the window as an array.

## Author

Ziyad

**Date**

29/11/2014

**Returns**

null if it fails, else the mouse position. MousePosition[0] always returns the X and MousePosition[1] always returns the Y.

Definition at line 692 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [MousePosition](#), and [PrintErrorMessage\(\)](#).

```
00693 {
00694     if (ContextCreated)
00695     {
00696         return MousePosition;
00697     }
00698     PrintErrorMessage(ERROR_NOCONTEXT);
00699     return nullptr;
00700 }
00701 }
```

**3.1.3.15 const char \* FWindow::GetOpenGLExtensions ( )**

Gets open gl extensions.

**Author**

Ziyad

**Date**

3/01/2015

**Returns**

null if it fails, else the open gl extensions.

Definition at line 1052 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), and [PrintErrorMessage\(\)](#).

```
01053 {
01054     if (ContextCreated)
01055     {
01056         return (const char*)glGetString(GL_EXTENSIONS);
01057     }
01058     else
01059     {
01060         PrintErrorMessage(ERROR_NOCONTEXT);
01061         return nullptr;
01062     }
01063 }
01064 }
```

**3.1.3.16 const char \* FWindow::GetOpenGLVersion ( )**

Definition at line 1019 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), and [PrintErrorMessage\(\)](#).

```
01020 {
01021     if (ContextCreated)
01022     {
01023         return (const char*)glGetString(GL_VERSION);
01024     }
01025     PrintErrorMessage(ERROR_NOCONTEXT);
01026     return nullptr;
01027 }
```

## 3.1.3.17 GLboolean FWindow::GetPosition ( GLuint &amp; X, GLuint &amp; Y )

Gets a position.

## Author

Ziyad

## Date

29/11/2014

## Parameters

in, out	X	The X coordinate of the window position relative to screen coordinates.
in, out	Y	The Y coordinates of the window position relative to screen coordinates.

Definition at line 748 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [Position](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::GetWindowPosition\(\)](#).

```

00749 {
00750     if (ContextCreated)
00751     {
00752         X = Position[0];
00753         Y = Position[1];
00754         return FOUNDATION_OKAY;
00755     }
00756     PrintErrorMessage(ERROR_NOCONTEXT);
00757     return FOUNDATION_ERROR;
00758 }
00759
00760 }
```

## 3.1.3.18 GLuint \* FWindow::GetPosition ( )

Gets the position.

## Author

Ziyad

## Date

29/11/2014

## Returns

null if it fails, else the position. Position[0] always returns the X coordinate of the window relative to screen coordinates and Position[1] always returns the Y coordinates of the window.

Definition at line 775 of file [Window.cpp](#).

References [Position](#).

```

00776 {
00777     return Position;
00778 }
```

### 3.1.3.19 GLboolean FWindow::GetResolution ( GLuint & *Width*, GLuint & *Height* )

Gets the resolution of the window by setting width and height.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

in, out	<i>Width</i>	The width.
in, out	<i>Height</i>	The height.

#### Returns

The resolution.

Definition at line 580 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [PrintErrorMessage\(\)](#), and [Resolution](#).

Referenced by [WindowManager::GetWindowResolution\(\)](#).

```

00581 {
00582     if (ContextCreated)
00583     {
00584         Width = Resolution[0];
00585         Height = Resolution[1];
00586         return FOUNDATION_OKAY;
00587     }
00588
00589     PrintErrorMessage(ERROR_NOCONTEXT);
00590     return FOUNDATION_ERROR;
00591 }
```

### 3.1.3.20 GLuint \* FWindow::GetResolution ( )

Gets the resolution of the window.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

null if it fails, else the resolution as an array. Resolution[0] always returns the width and Resolution[1] always returns the Height.

Definition at line 605 of file [Window.cpp](#).

References [Resolution](#).

```

00606 {
00607     return Resolution;
00608 }
```

### 3.1.3.21 GLboolean FWindow::GetShouldClose ( )

Gets should close.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

The should close.

Definition at line 142 of file [Window.cpp](#).

References [ShouldClose](#).

Referenced by [WindowManager::GetWindowShouldClose\(\)](#).

```
00143 {  
00144     return ShouldClose;  
00145 }
```

### 3.1.3.22 const char \* FWindow::GetWindowName ( )

Gets window name.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

null if it fails, else the window name.

Definition at line 822 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [Name](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::GetWindowName\(\)](#).

```
00823 {  
00824     if (ContextCreated)  
00825     {  
00826         return Name;  
00827     }  
00828     PrintErrorMessage(ERROR_NOCONTEXT);  
00830     return nullptr;  
00831 }
```

### 3.1.3.23 void FWindow::InitGLExtensions ( ) [private]

Initializes the OpenGL extensions for this window.

#### Author

Ziyad

#### Date

29/11/2014

Definition at line 985 of file [Window.cpp](#).

```
00986 {  
00987     #if defined(CURRENT_OS_WINDOWS)  
00988         Windows_InitGLExtensions();  
00989     #endif  
00990  
00991     #if defined(CURRENT_OS_LINUX)  
00992         Linux_InitGLExtensions();  
00993     #endif  
00994 }
```

### 3.1.3.24 GLboolean FWindow::Initialize ( )

Initializes this object.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

A GLboolean.

Definition at line 120 of file [Window.cpp](#).

Referenced by [WindowManager::AddWindow\(\)](#).

```
00121 {  
00122     #if defined(CURRENT_OS_WINDOWS)  
00123         return Windows_Initialize();  
00124     #endif  
00125  
00126     #if defined(CURRENT_OS_LINUX)  
00127         return Linux_Initialize();  
00128     #endif  
00129 }
```

### 3.1.3.25 void FWindow::InitializeEvents ( ) [private]

Initializes the events.

#### Author

Ziyad



**Date**

29/11/2014

Definition at line 156 of file [Window.cpp](#).

References [DestroyedEvent](#), [KeyEvent](#), [MaximizedEvent](#), [MinimizedEvent](#), [MouseButtonEvent](#), [MouseMoveEvent](#), [MouseWheelEvent](#), and [MovedEvent](#).

Referenced by [FWindow\(\)](#).

```
00157 {
00158     KeyEvent = nullptr;
00159     MouseButtonEvent = nullptr;
00160     MouseWheelEvent = nullptr;
00161     DestroyedEvent = nullptr;
00162     MaximizedEvent = nullptr;
00163     MinimizedEvent = nullptr;
00164     // RestoredEvent = nullptr;
00165     MovedEvent = nullptr;
00166     MouseMoveEvent = nullptr;
00167 }
```

**3.1.3.26 GLboolean FWindow::InitializeGL ( )**

Initializes OpenGL for this window.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

A GLboolean.

Definition at line 198 of file [Window.cpp](#).

```
00199 {
00200     #if defined(CURRENT_OS_WINDOWS)
00201         return Windows_InitializeGL();
00202     #endif
00203
00204     #if defined(CURRENT_OS_LINUX)
00205         return Linux_InitializeGL();
00206     #endif
00207 }
```

**3.1.3.27 GLboolean FWindow::MakeCurrentContext ( )**

Makes the window be the current OpenGL context to be drawn to. NOTE: Does not change the IsCurrentContext variable for other windows.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

A GLboolean.

Definition at line 920 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsCurrentContext](#), and [PrintErrorMessage\(\)](#).

Referenced by [main\(\)](#).

```

00921 {
00922     if (ContextCreated)
00923     {
00924         IsCurrentContext = true;
00925 #if defined(CURRENT_OS_WINDOWS)
00926         wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);
00927 #endif
00928 #if defined(CURRENT_OS_LINUX)
00929         glXMakeCurrent(WindowManager::GetDisplay(), WindowHandle, Context);
00930 #endif
00931         return FOUNDATION_OKAY;
00932     }
00933     PrintErrorMessage(ERROR_NOCONTEXT);
00934     return FOUNDATION_ERROR;
00935 }

```

**3.1.3.28 GLboolean FWindow::Maximize ( GLboolean *NewState* )**

Maximizes the window depending on New state.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>NewState</i>	Whether to minimize the window.
-----------------	---------------------------------

**Returns**

A GLboolean.

Definition at line 493 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [PrintErrorMessage\(\)](#), [WINDOWSTATE\\_MAXIMIZED](#), and [WINDOWSTATE\\_NORMAL](#).

Referenced by [Restore\(\)](#), and [SetCurrentState\(\)](#).

```

00494 {
00495     if (ContextCreated)
00496     {
00497         if (NewState)
00498         {
00499             CurrentState = WINDOWSTATE_MAXIMIZED;
00500         }
00501         else
00502         {
00503             CurrentState = WINDOWSTATE_NORMAL;
00504         }
00505     }
00506 }

```

```

00507 #if defined(CURRENT_OS_WINDOWS)
00508     Windows_Maximize();
00509 #endif
00510
00511 #if defined(CURRENT_OS_LINUX)
00512     Linux_Maximize(NewState);
00513 #endif
00514     return FOUNDATION_OKAY;
00515 }
00516 PrintErrorMessage(ERROR_NOCONTEXT);
00517 return FOUNDATION_ERROR;
00518 }

```

### 3.1.3.29 GLboolean FWindow::Minimize ( GLboolean *NewState* )

set the window to be minimized depending on *NewState*.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>NewState</i>	whether the window should be minimized.
-----------------	---

#### Returns

A GLboolean.

Definition at line 436 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [WINDOWSTATE\\_MINIMIZED](#), and [WINDOWSTATE\\_NORMAL](#).

Referenced by [SetCurrentState\(\)](#).

```

00437 {
00438     if(ContextCreated)
00439     {
00440         if(NewState)
00441         {
00442             CurrentState = WINDOWSTATE_MINIMIZED;
00443         }
00444     }
00445     else
00446     {
00447         CurrentState = WINDOWSTATE_NORMAL;
00448     }
00449 }
00450 #if defined(CURRENT_OS_WINDOWS)
00451     Windows_Minimize();
00452 #endif
00453
00454 #if defined(CURRENT_OS_LINUX)
00455     Linux_Minimize(NewState);
00456 #endif
00457     return FOUNDATION_OKAY;
00458 }
00460
00461 return FOUNDATION_ERROR;
00462 }

```

### 3.1.3.30 GLboolean FWindow::PrintOpenGLExtensions ( )

Definition at line 1029 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

```

01030 {
01031     if(ContextCreated)
01032     {
01033         printf("%s \n", (const char*)glGetString(GL_EXTENSIONS));
01034         return FOUNDATION_OKAY;
01035     }
01036
01037     PrintErrorMessage(ERROR_NOCONTEXT);
01038     return FOUNDATION_ERROR;
01039 }
```

### 3.1.3.31 GLboolean FWindow::PrintOpenGLVersion ( )

Print open gl version.

**Author**

Ziyad

**Date**

3/01/2015

**Returns**

A GLboolean.

Definition at line 1007 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

```

01008 {
01009     if(ContextCreated)
01010     {
01011         printf("%s\n", glGetString(GL_VERSION));
01012         return FOUNDATION_OKAY;
01013     }
01014
01015     PrintErrorMessage(ERROR_NOCONTEXT);
01016     return FOUNDATION_ERROR;
01017 }
```

### 3.1.3.32 GLboolean FWindow::Restore ( )

Restores the window to its default setting.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

A GLboolean.

Definition at line 531 of file [Window.cpp](#).

References [ContextCreated](#), [CurrentState](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [FullScreen\(\)](#), [Maximize\(\)](#), [PrintErrorMessage\(\)](#), [WINDOWSTATE\\_FULLSCREEN](#), [WINDOWSTATE\\_MAXIMIZED](#), and [WINDOWSTATE\\_NORMAL](#).

Referenced by [WindowManager::RestoreWindow\(\)](#), and [SetCurrentState\(\)](#).

```

00532 {
00533     if (ContextCreated)
00534     {
00535         switch (CurrentState)
00536         {
00537             case WINDOWSTATE_MAXIMIZED:
00538             {
00539                 Maximize(GL_FALSE);
00540                 break;
00541             }
00542             case WINDOWSTATE_FULLSCREEN:
00543             {
00544                 FullScreen(GL_FALSE);
00545                 break;
00546             }
00547             }
00548         }
00549         CurrentState = WINDOWSTATE_NORMAL;
00550     #if defined(CURRENT_OS_WINDOWS)
00551         Windows_Restore();
00552     #endif
00553     #if defined(CURRENT_OS_LINUX)
00554         Linux_Restore();
00555     #endif
00556     return FOUNDATION_OKAY;
00557 }
00558 PrintErrorMessage(ERROR_NOCONTEXT);
00559 return FOUNDATION_ERROR;
00560 }
00561 }
00562 }
00563 }
00564 }
```

**3.1.3.33 GLboolean FWindow::SetCurrentState ( GLuint *NewState* )**

Sets the current state of the window.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>NewState</i>	new state of the window.
-----------------	--------------------------

**Returns**

A GLboolean.

first we restore the window to make moving from state to state as easy as possible

Definition at line 301 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FullScreen\(\)](#), [Maximize\(\)](#), [Minimize\(\)](#), [PrintErrorMessage\(\)](#), [Restore\(\)](#), [WINDOWSTATE\\_FULLSCREEN](#), [WINDOWSTATE\\_MAXIMIZED](#), and [WINDOWSTATE\\_MINIMIZED](#).

```

00302 {
00307     if (ContextCreated)
00308     {
00309         Restore();
00311
00312         switch (NewState)
00313         {
00314             case WINDOWSTATE_MAXIMIZED:
00315             {
00316                 Maximize(GL_TRUE);
00317                 break;
00318             }
00319
00320             case WINDOWSTATE_MINIMIZED:
00321             {
00322                 Minimize(GL_TRUE);
00323                 break;
00324             }
00325
00326             case WINDOWSTATE_FULLSCREEN:
00327             {
00328                 FullScreen(GL_FALSE);
00329                 break;
00330             }
00331
00332             default:
00333             {
00334                 break;
00335             }
00336         }
00337     }
00338
00339     PrintErrorMessage(ERROR_NOCONTEXT);
00340     return FOUNDATION_ERROR;
00341 }

```

### 3.1.334 GLboolean FWindow::SetIcon ( const char \* Icon, GLuint Width, GLuint Height )

Definition at line 871 of file [Window.cpp](#).

References [ContextCreated](#), [FOUNDATION\\_ERROR](#), and [FOUNDATION\\_OKAY](#).

```

00872 {
00873     if (ContextCreated)
00874     {
00875         #if defined(CURRENT_OS_WINDOWS)
00876             Windows_SetIcon(Icon, Width, Height);
00877         #endif
00878
00879         #if defined(CURRENT_OS_LINUX)
00880             Linux_SetIcon(Icon, Width, Height);
00881         #endif
00882
00883         return FOUNDATION_OKAY;
00884     }
00885
00886     return FOUNDATION_ERROR;
00887 }

```

### 3.1.335 GLboolean FWindow::SetMousePosition ( GLuint X, GLuint Y )

Sets mouse position.

**Author**

Ziyad

**Date**

29/11/2014

## Parameters

X	The new X position of the mouse relative to the window coordinates.
Y	The new Y position of the mouse relative to the window coordinates.

Definition at line 715 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [MousePosition](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetMousePositionInWindow\(\)](#).

```

00716 {
00717     if (ContextCreated)
00718     {
00719         MousePosition[0] = X;
00720         MousePosition[1] = Y;
00721 #if defined(CURRENT_OS_WINDOWS)
00722     Windows_SetMousePosition(X, Y);
00723 #endif
00724
00725 #if defined(CURRENT_OS_LINUX)
00726     Linux_SetMousePosition(X, Y);
00727 #endif
00728
00729     return FOUNDATION_OKAY;
00730 }
00731
00732 PrintErrorMessage(ERROR_NOCONTEXT);
00733 return FOUNDATION_ERROR;
00734 }
```

### 3.1.3.36 GLboolean FWindow::SetOnDestroyed ( OnDestroyedEvent OnDestroyed )

Sets on destroyed.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>OnDestroyed</i>	The on destroyed event.
--------------------	-------------------------

## Returns

A GLboolean.

Definition at line 1202 of file [Window.cpp](#).

References [DestroyedEvent](#), [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidDestroyedEvent\(\)](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnDestroyed\(\)](#).

```

01203 {
01204     if (IsValidDestroyedEvent(OnDestroyed))
01205     {
01206         DestroyedEvent = OnDestroyed;
01207         return FOUNDATION_OKAY;
01208     }
01209
01210     PrintErrorMessage(ERROR_INVALIDEVENT);
01211     return FOUNDATION_ERROR;
01212 }
```

### 3.1.3.37 GLboolean FWindow::SetOnFocus ( OnFocusEvent OnFocus )

Sets on focus.

Author

Ziyad

Date

29/11/2014

Parameters

<i>OnFocus</i>	The on focus event.
----------------	---------------------

Returns

A GLboolean.

Definition at line 1284 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FocusEvent](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidFocusEvent\(\)](#), and [PrintErrorMessage\(\)](#).

```
01285 {  
01286     if (IsValidFocusEvent (OnFocus) )  
01287     {  
01288         FocusEvent = OnFocus;  
01289         return FOUNDATION_OKAY;  
01290     }  
01291  
01292     PrintErrorMessage (ERROR_INVALIDEVENT) ;  
01293     return FOUNDATION_ERROR;  
01294 }
```

### 3.1.3.38 GLboolean FWindow::SetOnKeyEvent ( OnKeyEvent OnKey )

Sets on key event.

Author

Ziyad

Date

29/11/2014

Parameters

<i>OnKey</i>	The on key event.
--------------	-------------------

Returns

A GLboolean.

Definition at line 1127 of file [Window.cpp](#).

References [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidKeyEvent\(\)](#), and [KeyEvent](#).

Referenced by [WindowManager::SetWindowOnKeyEvent\(\)](#).



```

01128 {
01129     if (IsValidKeyEvent (OnKey))
01130     {
01131         KeyEvent = OnKey;
01132         return FOUNDATION_OKAY;
01133     }
01134
01135     return FOUNDATION_ERROR;
01136 }

```

### 3.1.3.39 GLboolean FWindow::SetOnMaximized ( OnMaximizedEvent OnMaximized )

Sets on maximized.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>OnMaximized</i>	The on maximized event.
--------------------	-------------------------

#### Returns

A GLboolean.

Definition at line 1227 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidDestroyedEvent\(\)](#), [MaximizedEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMaximized\(\)](#).

```

01228 {
01229     if (IsValidDestroyedEvent (OnMaximized))
01230     {
01231         MaximizedEvent = OnMaximized;
01232         return FOUNDATION_OKAY;
01233     }
01234     PrintErrorMessage (ERROR_INVALIDEVENT);
01235     return FOUNDATION_ERROR;
01236 }

```

### 3.1.3.40 GLboolean FWindow::SetOnMinimized ( OnMinimizedEvent OnMinimized )

Sets on minimized.

#### Author

Ziyad

#### Date

29/11/2014

## Parameters

<i>OnMinimized</i>	The on minimized event.
--------------------	-------------------------

## Returns

A GLboolean.

Definition at line 1251 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidDestroyedEvent\(\)](#), [MinimizedEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMinimized\(\)](#).

```

01252 {
01253     if (IsValidDestroyedEvent (OnMinimized))
01254     {
01255         MinimizedEvent = OnMinimized;
01256         return FOUNDATION_OKAY;
01257     }
01258
01259     PrintErrorMessage (ERROR_INVALIDEVENT);
01260     return FOUNDATION_ERROR;
01261 }
```

### 3.1.3.41 GLboolean FWindow::SetOnMouseButtonEvent ( OnMouseButtonEvent OnMouseButtonEvent )

Sets on mouse button event.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>OnMouseButton-Event</i>	The on mouse button event.
----------------------------	----------------------------

## Returns

A GLboolean.

Definition at line 1151 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidKeyEvent\(\)](#), [MouseButtonEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMouseButtonEvent\(\)](#).

```

01152 {
01153     //we don't really need to check if the context has been created
01154     if (IsValidKeyEvent (OnMouseButtonEvent))
01155     {
01156         MouseButtonEvent = OnMouseButtonEvent;
01157         return FOUNDATION_OKAY;
01158     }
01159
01160     PrintErrorMessage (ERROR_INVALIDEVENT);
01161     return FOUNDATION_ERROR;
01162 }
```

#### 3.1.3.42 GLboolean FWindow::SetOnMouseMove ( OnMouseMoveEvent OnMouseMove )

Sets on mouse move.

##### Author

Ziyad

##### Date

29/11/2014

##### Parameters

<i>OnMouseMove</i>	The on mouse move event.
--------------------	--------------------------

Definition at line 1352 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidMouseMoveEvent\(\)](#), [MouseMoveEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMouseMove\(\)](#).

```
01353 {  
01354     if (IsValidMouseMoveEvent (OnMouseMove) )  
01355     {  
01356         MouseMoveEvent = OnMouseMove;  
01357         return FOUNDATION_OKAY;  
01358     }  
01359  
01360     PrintErrorMessage(ERROR_INVALIDEVENT);  
01361     return FOUNDATION_ERROR;  
01362 }
```

#### 3.1.3.43 GLboolean FWindow::SetOnMouseWheelEvent ( OnMouseWheelEvent OnMouseWheel )

Sets on mouse wheel event.

##### Author

Ziyad

##### Date

29/11/2014

##### Parameters

<i>OnMouseWheel</i>	The on mouse wheel event.
---------------------	---------------------------

##### Returns

A GLboolean.

Definition at line 1177 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidMouseWheelEvent\(\)](#), [MouseWheelEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMouseWheelEvent\(\)](#).

```

01178 {
01179     if (IsValidMouseEvent (OnMouseWheel))
01180     {
01181         MouseWheelEvent = OnMouseWheel;
01182         return FOUNDATION_OKAY;
01183     }
01184     PrintErrorMessage (ERROR_INVALIDEVENT);
01185     return FOUNDATION_ERROR;
01186 }
01187 }

```

### 3.1.3.44 GLboolean FWindow::SetOnMoved ( OnMovedEvent *OnMoved* )

Sets on moved.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>OnMoved</i>	The on moved event.
----------------	---------------------

Definition at line 1307 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidMovedEvent\(\)](#), [MovedEvent](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowOnMoved\(\)](#).

```

01308 {
01309     if (IsValidMovedEvent (OnMoved))
01310     {
01311         MovedEvent = OnMoved;
01312         return FOUNDATION_OKAY;
01313     }
01314     PrintErrorMessage (ERROR_INVALIDEVENT);
01315     return FOUNDATION_ERROR;
01316 }

```

### 3.1.3.45 GLboolean FWindow::SetOnResize ( OnResizeEvent *OnResize* )

Sets on resize.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>OnResize</i>	The on resize event.
-----------------	----------------------

Definition at line 1329 of file [Window.cpp](#).

References [ERROR\\_INVALIDEVENT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [IsValidMovedEvent\(\)](#), [PrintErrorMessage\(\)](#), and [ResizeEvent](#).

Referenced by [WindowManager::SetWindowOnResize\(\)](#).

```

01330 {
01331     if (IsValidMovedEvent (OnResize))
01332     {
01333         ResizeEvent = OnResize;
01334         return FOUNDATION_OKAY;
01335     }
01336
01337     PrintErrorMessage (ERROR_INVALIDEVENT);
01338     return FOUNDATION_ERROR;
01339 }

```

### 3.1.3.46 GLboolean FWindow::SetPosition ( GLuint X, GLuint Y )

Sets a position.

Author

Ziyad

Date

29/11/2014

Parameters

X	The new X coordinate of the window position relative to screen coordinates.
Y	The new Y coordinate of the window position relative to screen coordinates.

Definition at line 792 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [Position](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowPosition\(\)](#).

```

00793 {
00794     if (ContextCreated)
00795     {
00796         Position[0] = X;
00797         Position[1] = Y;
00798 #if defined(CURRENT_OS_WINDOWS)
00799     Windows_SetPosition (Position[0], Position[1]);
00800 #endif
00801
00802 #if defined(CURRENT_OS_LINUX)
00803     Linux_SetPosition (X, Y);
00804 #endif
00805     }
00806
00807     PrintErrorMessage (ERROR_NOCONTEXT);
00808     return FOUNDATION_ERROR;
00809 }

```

### 3.1.3.47 GLboolean FWindow::SetResolution ( GLuint Width, GLuint Height )

Sets the resolution of the window.

Author

Ziyad

Date

29/11/2014

**Parameters**

<i>Width</i>	The new width of the window.
<i>Height</i>	The new height of the window.

Definition at line 622 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_INVALIDRESOLUTION](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [PrintErrorMessage\(\)](#), and [Resolution](#).

Referenced by [WindowManager::SetWindowResolution\(\)](#).

```

00623 {
00624     if (ContextCreated)
00625     {
00626         if (Width > 0 && Height > 0)
00627         {
00628             Resolution[0] = Width;
00629             Resolution[1] = Height;
00630
00631 #if defined(CURRENT_OS_WINDOWS)
00632     Windows_SetResolution(Resolution[0], Resolution[1]);
00633 #endif
00634
00635 #if defined(CURRENT_OS_LINUX)
00636     Linux_SetResolution(Width, Height);
00637 #endif
00638
00639     glViewport(0, 0, Resolution[0], Resolution[1]);
00640
00641     return FOUNDATION_OKAY;
00642 }
00643
00644 else
00645 {
00646     PrintErrorMessage(ERROR_INVALIDRESOLUTION);
00647     return FOUNDATION_ERROR;
00648 }
00649 }
00651 PrintErrorMessage(ERROR_NOCONTEXT);
00652 return FOUNDATION_ERROR;
00653 }
```

**3.1.3.48 GLboolean FWindow::SetStyle ( GLint WindowType )**

Definition at line 889 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

Referenced by [WindowManager::SetWindowStyle\(\)](#).

```

00890 {
00891     if (ContextCreated)
00892     {
00893 #if defined(CURRENT_OS_WINDOWS)
00894     Windows_SetStyle(WindowType);
00895 #endif
00896
00897 #if defined(CURRENT_OS_LINUX)
00898     Linux_SetStyle(WindowType);
00899 #endif
00900
00901     PrintErrorMessage(ERROR_NOCONTEXT);
00902     return FOUNDATION_OKAY;
00903 }
00904
00905 return FOUNDATION_ERROR;
00906 }
```

**3.1.3.49 GLboolean FWindow::SetSwapInterval ( GLint SwapSetting )**

Sets swap interval(V-sync).

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>SwapSetting</i>	The swap setting.
--------------------	-------------------

**Returns**

A GLboolean.

Definition at line 252 of file [Window.cpp](#).References [ContextCreated](#), [CurrentSwapInterval](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).Referenced by [WindowManager::SetWindowSwapInterval\(\)](#).

```

00253 {
00254     if (ContextCreated)
00255     {
00256         CurrentSwapInterval = SwapSetting;
00257 #if defined(CURRENT_OS_WINDOWS)
00258         Windows_VerticalSync(SwapSetting);
00259 #endif
00260 #if defined(CURRENT_OS_LINUX)
00261         Linux_VerticalSync(SwapSetting);
00262 #endif
00263 #endif
00264     return FOUNDATION_OKAY;
00265 }
00266
00267     PrintErrorMessage(ERROR_NOCONTEXT);
00268     return FOUNDATION_ERROR;
00269
00270 }
```

**3.1.3.50 GLboolean FWindow::SetTitleBar ( const char \* *NewTitle* )**

Sets title bar.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>NewTitle</i>	The new title bar of the window.
-----------------	----------------------------------

Definition at line 844 of file [Window.cpp](#).References [ContextCreated](#), [ERROR\\_INVALIDTITLBAR](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).Referenced by [WindowManager::SetWindowTitleBar\(\)](#).

```

00845 {
00846     if (ContextCreated)
00847     {
00848         if(NewTitle != nullptr)
00849         {
00850             #if defined(CURRENT_OS_LINUX)
00851                 Linux_SetTitleBar(NewTitle);
00852             #endif
00853             #if defined(CURRENT_OS_WINDOWS)
00854                 Windows_SetTitleBar(NewTitle);
00855             #endif
00856             return FOUNDATION_OKAY;
00857         }
00858     }
00859     else
00860     {
00861         PrintErrorMessage(ERROR_INVALIDTITLEBAR);
00862         return FOUNDATION_ERROR;
00863     }
00864 }
00865 }
00866
00867 PrintErrorMessage(ERROR_NOCONTEXT);
00868 return FOUNDATION_ERROR;
00869 }

```

### 3.1.3.51 GLboolean FWindow::Shutdown ( )

Shuts down this object and frees any resources it is using.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

A GLboolean.

Definition at line 89 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

Referenced by [~FWindow\(\)](#).

```

00090 {
00091     if(ContextCreated)
00092     {
00093         #if defined (CURRENT_OS_WINDOWS)
00094             Windows_Shutdown();
00095         #endif
00096         #if defined(CURRENT_OS_LINUX)
00097             Linux_Shutdown();
00098         #endif
00099         ContextCreated = GL_FALSE;
00100         return FOUNDATION_OKAY;
00101     }
00102     PrintErrorMessage(ERROR_NOCONTEXT);
00103     return FOUNDATION_ERROR;
00104 }
00105 }
00106 }
00107 }

```



### 3.1.3.52 GLboolean FWindow::SwapDrawBuffers ( )

Swap draw buffers.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

A GLboolean.

Definition at line 220 of file [Window.cpp](#).

References [ContextCreated](#), [ERROR\\_NOCONTEXT](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [PrintErrorMessage\(\)](#).

Referenced by [main\(\)](#), and [WindowManager::WindowSwapBuffers\(\)](#).

```
00221 {
00222     if (ContextCreated)
00223     {
00224         #if defined(CURRENT_OS_WINDOWS)
00225             SwapBuffers (DeviceContextHandle);
00226         #endif
00227     }
00228     #if defined(CURRENT_OS_LINUX)
00229         glXSwapBuffers (WindowManager::GetDisplay(), WindowHandle);
00230     #endif
00231     return FOUNDATION_OKAY;
00232 }
00233
00234
00235     PrintErrorMessage (ERROR_NOCONTEXT);
00236     return FOUNDATION_ERROR;
00237 }
```

## 3.1.4 Friends And Related Function Documentation

### 3.1.4.1 friend class WindowManager [friend]

Definition at line 157 of file [Window.h](#).

## 3.1.5 Member Data Documentation

### 3.1.5.1 GLint FWindow::ColourBits [private]

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

Definition at line 163 of file [Window.h](#).

### 3.1.5.2 GLboolean FWindow::ContextCreated [private]

Whether the OpenGL context for this window has been created

Definition at line 174 of file [Window.h](#).

Referenced by [DisableDecorator\(\)](#), [EnableDecorator\(\)](#), [Focus\(\)](#), [FullScreen\(\)](#), [FWindow\(\)](#), [GetContextHasBeenCreated\(\)](#), [GetIsCurrentContext\(\)](#), [GetIsFullScreen\(\)](#), [GetMousePosition\(\)](#), [GetOpenGLExtensions\(\)](#), [GetOpenGLVersion\(\)](#), [GetPosition\(\)](#), [GetResolution\(\)](#), [GetWindowName\(\)](#), [MakeCurrentContext\(\)](#), [Maximize\(\)](#), [Minimize\(\)](#), [PrintOpenGLExtensions\(\)](#), [PrintOpenGLVersion\(\)](#), [Restore\(\)](#), [SetCurrentState\(\)](#), [SetIcon\(\)](#), [SetMousePosition\(\)](#), [SetPosition\(\)](#), [SetResolution\(\)](#), [SetStyle\(\)](#), [SetSwapInterval\(\)](#), [SetTitleBar\(\)](#), [Shutdown\(\)](#), and [SwapDrawBuffers\(\)](#).

### 3.1.5.3 GLuint FWindow::CurrentState [private]

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

Definition at line 176 of file [Window.h](#).

Referenced by [FullScreen\(\)](#), [FWindow\(\)](#), [GetCurrentState\(\)](#), [GetIsFullScreen\(\)](#), [GetIsMaximized\(\)](#), [GetIsMinimized\(\)](#), [Maximize\(\)](#), [Minimize\(\)](#), and [Restore\(\)](#).

### 3.1.5.4 GLuint FWindow::CurrentSwapInterval [private]

The current swap interval of the window(V-Sync). a value of -1 enables adaptive V-Sync on supported systems

Definition at line 177 of file [Window.h](#).

Referenced by [SetSwapInterval\(\)](#).

### 3.1.5.5 GLbitfield FWindow::CurrentWindowStyle [private]

the current window style

Definition at line 178 of file [Window.h](#).

### 3.1.5.6 GLint FWindow::DepthBits [private]

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

Definition at line 164 of file [Window.h](#).

### 3.1.5.7 OnDestroyedEvent FWindow::DestroyedEvent [private]

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

Definition at line 188 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnDestroyed\(\)](#).

### 3.1.5.8 GLboolean FWindow::EXTSwapControlSupported [private]

Whether the EXT\_Swap\_Control(Generic) GL extension is supported on this machine

Definition at line 197 of file [Window.h](#).

Referenced by [FWindow\(\)](#).

### 3.1.5.9 OnFocusEvent FWindow::FocusEvent [private]

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

Definition at line 192 of file [Window.h](#).

Referenced by [SetOnFocus\(\)](#), and [WindowManager::SetWindowOnFocus\(\)](#).

### 3.1.5.10 GLuint FWindow::ID [private]

ID of the [FWindow](#). (where it belongs in the window manager)

Definition at line 162 of file [Window.h](#).

Referenced by [WindowManager::AddWindow\(\)](#), and [WindowManager::GetWindowIndex\(\)](#).

**3.1.5.11 GLboolean FWindow::InFocus** [private]

Whether the [FWindow](#) is currently in focus(if it is the current window be used)

Definition at line 172 of file [Window.h](#).

Referenced by [Focus\(\)](#), and [GetInFocus\(\)](#).

**3.1.5.12 GLboolean FWindow::Initialized** [private]

Whether the FWindow has been fully Initialized

Definition at line 173 of file [Window.h](#).

**3.1.5.13 GLboolean FWindow::IsCurrentContext** [private]

Whether the window is the current window that is being drawn to

Definition at line 175 of file [Window.h](#).

Referenced by [FWindow\(\)](#), [GetIsCurrentContext\(\)](#), and [MakeCurrentContext\(\)](#).

**3.1.5.14 OnKeyEvent FWindow::KeyEvent** [private]

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

Definition at line 185 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnKeyEvent\(\)](#).

**3.1.5.15 GLboolean FWindow::Keys[256+1+54]** [private]

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

Definition at line 166 of file [Window.h](#).

Referenced by [GetKeyState\(\)](#).

**3.1.5.16 OnMaximizedEvent FWindow::MaximizedEvent** [private]

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

Definition at line 189 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMaximized\(\)](#).

**3.1.5.17 GLboolean FWindow::MESASwapControlSupported** [private]

Whether the MESA\_Swap\_Control(Mesa) GL extension is supported on this machine

Definition at line 199 of file [Window.h](#).

Referenced by [FWindow\(\)](#).

**3.1.5.18 OnMinimizedEvent FWindow::MinimizedEvent** [private]

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

Definition at line 190 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMinimized\(\)](#).

### 3.1.5.19 GLboolean FWindow::MouseButton[2+1] [private]

Record of mouse buttons that are either presses or released

Definition at line 167 of file [Window.h](#).

### 3.1.5.20 OnMouseButtonEvent FWindow::MouseButtonEvent [private]

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

Definition at line 186 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMouseButtonEvent\(\)](#).

### 3.1.5.21 OnMouseMoveEvent FWindow::MouseMoveEvent [private]

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

Definition at line 195 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMouseMove\(\)](#).

### 3.1.5.22 GLuint FWindow::MousePosition[2] [private]

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

Definition at line 170 of file [Window.h](#).

Referenced by [GetMousePosition\(\)](#), and [SetMousePosition\(\)](#).

### 3.1.5.23 OnMouseWheelEvent FWindow::MouseWheelEvent [private]

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

Definition at line 187 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMouseWheelEvent\(\)](#).

### 3.1.5.24 OnMovedEvent FWindow::MovedEvent [private]

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

Definition at line 193 of file [Window.h](#).

Referenced by [InitializeEvents\(\)](#), and [SetOnMoved\(\)](#).

### 3.1.5.25 const char\* FWindow::Name [private]

Name of the window. also initially the title bar text

Definition at line 161 of file [Window.h](#).

Referenced by [GetWindowName\(\)](#).

### 3.1.5.26 GLuint FWindow::Position[2] [private]

Position of the [FWindow](#) relative to the screen co-ordinates

Definition at line 169 of file [Window.h](#).

Referenced by [FWindow\(\)](#), [GetPosition\(\)](#), and [SetPosition\(\)](#).

**3.1.5.27 OnResizeEvent FWindow::ResizeEvent** [private]

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

Definition at line 194 of file [Window.h](#).

Referenced by [SetOnResize\(\)](#).

**3.1.5.28 GLuint FWindow::Resolution[2]** [private]

Resolution/Size of the window stored in an array

Definition at line 168 of file [Window.h](#).

Referenced by [FWindow\(\)](#), [GetResolution\(\)](#), and [SetResolution\(\)](#).

**3.1.5.29 GLboolean FWindow::SGISwapControlSupported** [private]

Whether the SGI\_Swap\_Control(Silicon graphics) GL extension is supported on this machine

Definition at line 198 of file [Window.h](#).

Referenced by [FWindow\(\)](#).

**3.1.5.30 GLboolean FWindow::ShouldClose** [private]

Whether the [FWindow](#) should be closing

Definition at line 171 of file [Window.h](#).

Referenced by [FWindow\(\)](#), and [GetShouldClose\(\)](#).

**3.1.5.31 GLint FWindow::StencilBits** [private]

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

Definition at line 165 of file [Window.h](#).

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

- [Window.h](#)
- [Window.cpp](#)

## 3.2 WindowManager Class Reference

```
#include <WindowManager.h>
```

### Public Member Functions

- [WindowManager](#) ()  
*Default constructor.*
- [~WindowManager](#) ()  
*Destructor.*

## Static Public Member Functions

- static void [ShutDown](#) ()  
*Shuts down this object and frees any resources it is using.*
- static [FWindow](#) \* [GetWindowByName](#) (const char \*WindowName)  
*Gets window by name.*
- static [FWindow](#) \* [GetWindowByIndex](#) (GLuint WindowIndex)  
*Gets window by index.*
- static [WindowManager](#) \* [AddWindow](#) ([FWindow](#) \*NewWindow)  
*Adds a window.*
- static GLuint [GetNumWindows](#) ()  
*Gets the number of windows.*
- static GLboolean [GetMousePositionInScreen](#) (GLuint &X, GLuint &Y)  
*Gets mouse position in screen.*
- static GLuint \* [GetMousePositionInScreen](#) ()  
*Gets mouse position in screen.*
- static GLboolean [SetMousePositionInScreen](#) (GLuint X, GLuint Y)  
*Sets mouse position in screen.*
- static GLuint \* [GetScreenResolution](#) ()  
*Gets screen resolution.*
- static GLboolean [GetScreenResolution](#) (GLuint &Width, GLuint &Height)  
*Gets screen resolution.*
- static GLboolean [GetWindowResolution](#) (const char \*WindowName, GLuint &Width, GLuint &Height)  
*Gets window resolution.*
- static GLboolean [GetWindowResolution](#) (GLuint WindowIndex, GLuint &Width, GLuint &Height)  
*Gets window resolution.*
- static GLuint \* [GetWindowResolution](#) (const char \*WindowName)  
*Gets window resolution as an array.*
- static GLuint \* [GetWindowResolution](#) (GLuint WindowIndex)  
*Gets window resolution.*
- static GLboolean [SetWindowResolution](#) (const char \*WindowName, GLuint Width, GLuint Height)  
*Sets window resolution.*
- static GLboolean [SetWindowResolution](#) (GLuint WindowIndex, GLuint Width, GLuint Height)  
*Sets window resolution.*
- static GLboolean [GetWindowPosition](#) (const char \*WindowName, GLuint &X, GLuint &Y)  
*Gets window position relative to screen coordinates.*
- static GLboolean [GetWindowPosition](#) (GLuint WindowIndex, GLuint &X, GLuint &Y)  
*Gets window position relative to screen coordinates.*
- static GLuint \* [GetWindowPosition](#) (const char \*WindowName)  
*Gets window position relative to screen coordinates.*
- static GLuint \* [GetWindowPosition](#) (GLuint WindowIndex)  
*Gets window position relative to screen coordinates.*
- static GLboolean [SetWindowPosition](#) (const char \*WindowName, GLuint X, GLuint Y)  
*Sets window position relative to screen coordinates.*
- static GLboolean [SetWindowPosition](#) (GLuint WindowIndex, GLuint X, GLuint Y)  
*Sets window position relative to screen coordinates.*
- static GLboolean [GetMousePositionInWindow](#) (const char \*WindowName, GLuint &X, GLuint &Y)  
*Gets mouse position in window.*
- static GLboolean [GetMousePositionInWindow](#) (GLuint WindowIndex, GLuint &X, GLuint &Y)  
*Gets mouse position in window.*
- static GLuint \* [GetMousePositionInWindow](#) (const char \*WindowName)

- Gets mouse position in window.*
- static GLuint \* [GetMousePositionInWindow](#) (GLuint WindowIndex)
- Gets mouse position in window.*
- static GLboolean [SetMousePositionInWindow](#) (const char \*WindowName, GLuint X, GLuint Y)
- Sets mouse position in window.*
- static GLboolean [SetMousePositionInWindow](#) (GLuint WindowIndex, GLuint X, GLuint Y)
- Sets mouse position in window.*
- static GLboolean [WindowGetKey](#) (const char \*WindowName, GLuint Key)
- get the state of the key relative to the window.*
- static GLboolean [WindowGetKey](#) (GLuint WindowIndex, GLuint Key)
- get the state of the key relative to the window.*
- static GLboolean [GetWindowShouldClose](#) (const char \*WindowName)
- Gets whether the window should close.*
- static GLboolean [GetWindowShouldClose](#) (GLuint WindowIndex)
- Gets whether the window should close.*
- static GLboolean [WindowSwapBuffers](#) (const char \*WindowName)
- Swap DrawBuffers for that window.*
- static GLboolean [WindowSwapBuffers](#) (GLuint WindowIndex)
- Swap DrawBuffers for that window.*
- static GLboolean [SetFullScreen](#) (const char \*WindowName, GLboolean NewState)
- toggle the fullscreen mode for the window.*
- static GLboolean [SetFullScreen](#) (GLuint WindowIndex, GLboolean NewState)
- toggle the fullscreen mode for the window.*
- static GLboolean [GetWindowIsFullScreen](#) (const char \*WindowName)
- Gets whether the window is full screen.*
- static GLboolean [GetWindowIsFullScreen](#) (GLuint WindowIndex)
- Gets whether the window is full screen.*
- static GLboolean [GetWindowIsMinimized](#) (const char \*WindowName)
- Gets window is minimized.*
- static GLboolean [GetWindowIsMinimized](#) (GLuint WindowIndex)
- Gets window is minimized.*
- static GLboolean [MinimizeWindow](#) (const char \*WindowName, GLboolean NewState)
- set the window to be minimized depending on NewState.*
- static GLboolean [MinimizeWindow](#) (GLuint WindowIndex, GLboolean NewState)
- Minimize window.*
- static GLboolean [GetWindowIsMaximized](#) (const char \*WindowName)
- Gets window is maximized.*
- static GLboolean [GetWindowIsMaximized](#) (GLuint WindowIndex)
- Gets window is maximized.*
- static GLboolean [MaximizeWindow](#) (const char \*WindowName, GLboolean NewState)
- Maximize window.*
- static GLboolean [MaximizeWindow](#) (GLuint WindowIndex, GLboolean NewState)
- Maximize window.*
- static const char \* [GetWindowName](#) (GLuint WindowIndex)
- Gets window name.*
- static GLuint [GetWindowIndex](#) (const char \*WindowName)
- Gets window index.*
- static GLboolean [SetWindowTitleBar](#) (const char \*WindowName, const char \*NewName)
- Sets window title bar.*
- static GLboolean [SetWindowTitleBar](#) (GLuint WindowIndex, const char \*NewName)
- Sets window title bar.*

- static GLboolean [SetWindowIcon](#) (const char \*WindowName, const char \*Icon, GLuint Width, GLuint Height)
- static GLboolean [SetWindowIcon](#) (GLuint WindowIndex, const char \*Icon, GLuint Width, GLuint Height)
- static GLboolean [GetWindowIsInFocus](#) (const char \*WindowName)  
*Gets window is in focus.*
- static GLboolean [GetWindowIsInFocus](#) (GLuint WindowIndex)  
*Gets window is in focus.*
- static GLboolean [FocusWindow](#) (const char \*WindowName, GLboolean NewState)  
*Focus window.*
- static GLboolean [FocusWindow](#) (GLuint WindowIndex, GLboolean NewState)  
*Focus window.*
- static GLboolean [RestoreWindow](#) (const char \*WindowName)  
*Restore window.*
- static GLboolean [RestoreWindow](#) (GLuint WindowIndex)  
*Restore window.*
- static GLboolean [SetWindowSwapInterval](#) (const char \*WindowName, GLint EnableSync)  
*Sets window swap interval.*
- static GLboolean [SetWindowSwapInterval](#) (GLuint WindowIndex, GLint EnableSync)  
*Sets window swap interval.*
- static GLboolean [Initialize](#) ()  
*Initializes this object.*
- static GLboolean [IsInitialized](#) ()
- static GLboolean [PollForEvents](#) ()  
*Poll for events for all windows in the manager.*
- static GLboolean [RemoveWindow](#) (FWindow \*WindowToBeRemoved)
- static GLboolean [SetWindowStyle](#) (const char \*WindowName, GLuint WindowStyle)
- static GLboolean [SetWindowStyle](#) (GLuint WindowIndex, GLuint WindowStyle)
- static GLboolean [EnableWindowDecorator](#) (const char \*WindowName, GLbitfield Decorators)
- static GLboolean [EnableWindowDecorator](#) (GLuint WindowIndex, GLbitfield Decorators)
- static GLboolean [DisableWindowDecorator](#) (const char \*WindowName, GLbitfield Decorators)
- static GLboolean [DisableWindowDecorator](#) (GLuint WindowIndex, GLbitfield Decorators)
- static GLboolean [SetWindowOnKeyEvent](#) (const char \*WindowName, [OnKeyEvent](#) OnKey)  
*Sets window on key event.*
- static GLboolean [SetWindowOnKeyEvent](#) (GLuint WindowIndex, [OnKeyEvent](#) OnKey)  
*Sets window on key event.*
- static GLboolean [SetWindowOnMouseButtonEvent](#) (const char \*WindowName, [OnMouseButtonEvent](#) a\_  
OnMouseButtonEvent)  
*Sets window on mouse button event.*
- static GLboolean [SetWindowOnMouseButtonEvent](#) (GLuint WindowIndex, [OnMouseButtonEvent](#) a\_  
MouseButtonEvent)  
*Sets window on mouse button event.*
- static GLboolean [SetWindowOnMouseWheelEvent](#) (const char \*WindowName, [OnMouseWheelEvent](#) On-  
[MouseWheelEvent](#))  
*Sets window on mouse wheel event.*
- static GLboolean [SetWindowOnMouseWheelEvent](#) (GLuint WindowIndex, [OnMouseWheelEvent](#) OnMouse-  
[WheelEvent](#))  
*Sets window on mouse wheel event.*
- static GLboolean [SetWindowOnDestroyed](#) (const char \*WindowName, [OnDestroyedEvent](#) OnDestroyed)  
*Sets window on destroyed.*
- static GLboolean [SetWindowOnDestroyed](#) (GLuint WindowIndex, [OnDestroyedEvent](#) OnDestroyed)  
*Sets window on destroyed.*
- static GLboolean [SetWindowOnMaximized](#) (const char \*WindowName, [OnMaximizedEvent](#) OnMaximized)  
*Sets window on maximized.*



- static GLboolean [SetWindowOnMaximized](#) (GLuint WindowIndex, [OnMaximizedEvent](#) OnMaximized)  
*Sets window on maximized.*
- static GLboolean [SetWindowOnMinimized](#) (const char \*WindowName, [OnMinimizedEvent](#) a\_OnMiniimzed)  
*Sets window on minimized.*
- static GLboolean [SetWindowOnMinimized](#) (GLuint WindowIndex, [OnMinimizedEvent](#) a\_OnMiniimzed)  
*Sets window on minimized.*
- static GLboolean [SetWindowOnFocus](#) (const char \*WindowName, [OnFocusEvent](#) OnFocus)  
*Sets window on focus.*
- static GLboolean [SetWindowOnFocus](#) (GLuint WindowIndex, [OnFocusEvent](#) OnFocus)  
*Sets window on focus.*
- static GLboolean [SetWindowOnMoved](#) (const char \*WindowName, [OnMovedEvent](#) OnMoved)  
*Sets window on moved.*
- static GLboolean [SetWindowOnMoved](#) (GLuint WindowIndex, [OnMovedEvent](#) OnMoved)  
*Sets window on moved.*
- static GLboolean [SetWindowOnResize](#) (const char \*WindowName, [OnResizeEvent](#) OnResize)  
*Sets window on resize.*
- static GLboolean [SetWindowOnResize](#) (GLuint WindowIndex, [OnResizeEvent](#) OnResize)  
*Sets window on resize.*
- static GLboolean [SetWindowOnMouseMove](#) (const char \*WindowName, [OnMouseMoveEvent](#) OnMouseMove)  
*Sets window on mouse move.*
- static GLboolean [SetWindowOnMouseMove](#) (GLuint WindowIndex, [OnMouseMoveEvent](#) OnMouseMove)  
*Sets window on mouse move.*

### Static Private Member Functions

- static GLboolean [DoesExist](#) (const char \*WindowName)  
*Does the window exist.*
- static GLboolean [DoesExist](#) (GLuint WindowIndex)  
*Does the window exist.*
- static [WindowManager](#) \* [GetInstance](#) ()  
*Gets the instance to the [WindowManager](#).*

### Private Attributes

- friend [FWindow](#)
- std::list< [FWindow](#) \* > [Windows](#)
- GLuint [ScreenResolution](#) [2]
- GLuint [ScreenMousePosition](#) [2]
- GLboolean [Initialized](#)

### Static Private Attributes

- static [WindowManager](#) \* [Instance](#) = 0

#### 3.2.1 Detailed Description

Definition at line 10 of file [WindowManager.h](#).

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 WindowManager::WindowManager ( )

Default constructor.

Author

Ziyad

Date

29/11/2014

Definition at line 18 of file [WindowManager.cpp](#).

Referenced by [GetInstance\(\)](#).

```
00019 {
00020     //GetInstance()->Initialized = GL_FALSE;
00021 }
```

#### 3.2.2.2 WindowManager::~~WindowManager ( )

Destructor.

Author

Ziyad

Date

29/11/2014

Definition at line 58 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), and [Windows](#).

```
00059 {
00060     if (!GetInstance()->Windows.empty())
00061     {
00062         #if defined(CURRENT_OS_WINDOWS)
00063             for each(auto CurrentWindow in GetInstance()->Windows)
00064             {
00065                 delete CurrentWindow;
00066             }
00067         #endif
00068     }
00069     #if defined(CURRENT_OS_LINUX)
00070     for (auto CurrentWindow : GetInstance()->Windows)
00071     {
00072         delete CurrentWindow;
00073     }
00074     #endif
00075     GetInstance()->Windows.clear();
00076 }
00077 }
```

### 3.2.3 Member Function Documentation

#### 3.2.3.1 WindowManager \* WindowManager::AddWindow ( FWindow \* NewWindow ) [static]

Adds a window.

add a window to the manager. i ripped off a tree feature that allows the user to create multiple windows easily

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>in, out</i>	<i>NewWindow</i>	If non-null, the new window.
----------------	------------------	------------------------------

**Returns**

null if it fails, else a reference to the [WindowManager](#).

Definition at line 177 of file [WindowManager.cpp](#).

References [ERROR\\_INVALIDWINDOW](#), [ERROR\\_NOTINITIALIZED](#), [GetInstance\(\)](#), [FWindow::ID](#), [FWindow::Initialize\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).

Referenced by [main\(\)](#).

```

00178 {
00179     if (GetInstance()->IsInitialized())
00180     {
00181         if (NewWindow != nullptr)
00182         {
00183             GetInstance()->Windows.push_back(NewWindow);
00184             NewWindow->ID = GetInstance()->Windows.size() - 1;
00185             NewWindow->Initialize();
00186             return GetInstance();
00187         }
00188         PrintErrorMessage(ERROR_INVALIDWINDOW);
00189         return nullptr;
00190     }
00191     PrintErrorMessage(ERROR_NOTINITIALIZED);
00192     return nullptr;
00193 }
```

### 3.2.3.2 GLboolean WindowManager::DisableWindowDecorator ( const char \* WindowName, GLbitfield Decorators ) [static]

Definition at line 1678 of file [WindowManager.cpp](#).

References [FWindow::DisableDecorator\(\)](#), [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByName\(\)](#).

```

01679 {
01680     if (DoesExist(WindowName))
01681     {
01682         return GetWindowByName(WindowName)->DisableDecorator(Decorators);
01683     }
01684     return FOUNDATION_ERROR;
01685 }
01686 }
```

### 3.2.3.3 GLboolean WindowManager::DisableWindowDecorator ( GLuint WindowIndex, GLbitfield Decorators ) [static]

Definition at line 1688 of file [WindowManager.cpp](#).

References [FWindow::DisableDecorator\(\)](#), [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByIndex\(\)](#).

```

01689 {
01690     if (DoesExist(WindowIndex))
01691     {
01692         return GetWindowByIndex(WindowIndex)->DisableDecorator(Decorators);
01693     }
01694     return FOUNDATION_ERROR;
01695 }
01696 }
```

### 3.2.3.4 GLboolean WindowManager::DoesExist ( const char \* *WindowName* ) [static],[private]

Does the window exist.

Author

Ziyad

Date

30/11/2014

Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

Returns

whether the window is in the window manager.

Definition at line 233 of file [WindowManager.cpp](#).

References [ERROR\\_INVALIDWINDOWNAME](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [IsValidString\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).

Referenced by [DisableWindowDecorator\(\)](#), [EnableWindowDecorator\(\)](#), [FocusWindow\(\)](#), [GetMousePositionInWindow\(\)](#), [GetWindowByIndex\(\)](#), [GetWindowByName\(\)](#), [GetWindowIndex\(\)](#), [GetWindowIsFullScreen\(\)](#), [GetWindowIsInFocus\(\)](#), [GetWindowIsMaximized\(\)](#), [GetWindowIsMinimized\(\)](#), [GetWindowName\(\)](#), [GetWindowPosition\(\)](#), [GetWindowResolution\(\)](#), [GetWindowShouldClose\(\)](#), [MaximizeWindow\(\)](#), [MinimizeWindow\(\)](#), [RestoreWindow\(\)](#), [SetFullScreen\(\)](#), [SetMousePositionInWindow\(\)](#), [SetWindowOnDestroyed\(\)](#), [SetWindowOnFocus\(\)](#), [SetWindowOnKeyEvent\(\)](#), [SetWindowOnMaximized\(\)](#), [SetWindowOnMinimized\(\)](#), [SetWindowOnMouseButtonEvent\(\)](#), [SetWindowOnMouseMove\(\)](#), [SetWindowOnMouseWheelEvent\(\)](#), [SetWindowOnMoved\(\)](#), [SetWindowOnResize\(\)](#), [SetWindowPosition\(\)](#), [SetWindowResolution\(\)](#), [SetWindowStyle\(\)](#), [SetWindowSwapInterval\(\)](#), [SetWindowTitleBar\(\)](#), [WindowGetKey\(\)](#), and [WindowSwapBuffers\(\)](#).

```

00234 {
00235     if (GetInstance()->IsInitialized())
00236     {
00237         if (IsValidString(WindowName))
00238         {
00239 #if defined(CURRENT_OS_WINDOWS)
00240             for each(auto iter in GetInstance()->Windows)
00241             {
00242                 if (iter->Name == WindowName)
00243                 {
00244                     return GL_TRUE;
00245                 }
00246             }
00247 #endif
00248 #if defined(CURRENT_OS_LINUX)
00249             for (auto iter : GetInstance()->Windows)
00250             {
00251                 if (iter->Name == WindowName)
00252                 {
00253                     return GL_TRUE;
00254                 }
00255             }
00256 #endif
00257         }
00258         PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
00259         return GL_FALSE;
00260     }
00261     return GL_FALSE;
00262 }
00263 }
```

### 3.2.3.5 GLboolean WindowManager::DoesExist ( GLuint *WindowIndex* ) [static],[private]

Does the window exist.

**Author**

Ziyad

**Date**

30/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

**Returns**

whether the window index given is lower then the current size of the windows container.

Definition at line 278 of file [WindowManager.cpp](#).

References [ERROR\\_INVALIDWINDOWINDEX](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).

```

00279 {
00280     if (GetInstance()->IsInitialized())
00281     {
00282         if (WindowIndex <= (GetInstance()->Windows.size() - 1))
00283         {
00284             return FOUNDATION_OKAY;
00285         }
00286         PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
00287         return FOUNDATION_ERROR;
00288     }
00289     return FOUNDATION_ERROR;
00290 }
00291 }
```

### 3.2.3.6 GLboolean WindowManager::EnableWindowDecorator ( const char \* *WindowName*, GLbitfield *Decorators* ) [static]

Definition at line 1657 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::EnableDecorator\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByName\(\)](#).

Referenced by [main\(\)](#).

```

01658 {
01659     if (DoesExist(WindowName))
01660     {
01661         return GetWindowByName(WindowName)->EnableDecorator(Decorators);
01662     }
01663     return FOUNDATION_ERROR;
01664 }
01665 }
01666 }
```

### 3.2.3.7 GLboolean WindowManager::EnableWindowDecorator ( GLuint *WindowIndex*, GLbitfield *Decorators* ) [static]

Definition at line 1668 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::EnableDecorator\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByIndex\(\)](#).

```

01669 {
01670     if (DoesExist(WindowIndex))
01671     {
01672         return GetWindowByIndex(WindowIndex)->EnableDecorator(Decorators);
01673     }
01674     return FOUNDATION_ERROR;
01675 }
01676 }
```

### 3.2.3.8 GLboolean WindowManager::FocusWindow ( const char \* *WindowName*, GLboolean *ShouldBeFocused* ) [static]

Focus window.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>ShouldBeFocused</i>	Whether the window should be in event focus.

Definition at line 1518 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::Focus\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByName\(\)](#).

```

01519 {
01520     if (DoesExist (WindowName) )
01521     {
01522         return GetWindowByName (WindowName) ->Focus (ShouldBeFocused) ;
01523     }
01524     return FOUNDATION_ERROR;
01525 }
01526 }
```

### 3.2.3.9 GLboolean WindowManager::FocusWindow ( GLuint *WindowIndex*, GLboolean *ShouldBeFocused* ) [static]

Focus window.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>ShouldBeFocused</i>	Whether the window should be in event focus.

Definition at line 1540 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::Focus\(\)](#), [FOUNDATION\\_ERROR](#), and [GetWindowByIndex\(\)](#).

```

01541 {
01542     if (DoesExist (WindowIndex) )
01543     {
01544         return GetWindowByIndex (WindowIndex) ->Focus (ShouldBeFocused) ;
01545     }
01546     return FOUNDATION_ERROR;
01547 }
01548 }
```

**3.2.3.10** `WindowManager * WindowManager::GetInstance ( )` `[static], [private]`

Gets the instance to the [WindowManager](#).

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

null if it fails, else the instance to the [WindowManager](#).

Definition at line 206 of file [WindowManager.cpp](#).

References [Instance](#), and [WindowManager\(\)](#).

Referenced by [AddWindow\(\)](#), [DoesExist\(\)](#), [GetMousePositionInScreen\(\)](#), [GetNumWindows\(\)](#), [GetScreenResolution\(\)](#), [GetWindowByIndex\(\)](#), [GetWindowByName\(\)](#), [GetWindowIsFullScreen\(\)](#), [GetWindowPosition\(\)](#), [GetWindowResolution\(\)](#), [Initialize\(\)](#), [IsInitialized\(\)](#), [PollForEvents\(\)](#), [SetMousePositionInScreen\(\)](#), [SetWindowPosition\(\)](#), [ShutDown\(\)](#), and [~WindowManager\(\)](#).

```
00207 {
00208     if (!WindowManager::Instance)
00209     {
00210         WindowManager::Instance = new WindowManager();
00211         return WindowManager::Instance;
00212     }
00213     else
00214     {
00215         return WindowManager::Instance;
00216     }
00217 }
00218 }
```

**3.2.3.11** `GLboolean WindowManager::GetMousePositionInScreen ( GLuint & X, GLuint & Y )` `[static]`

Gets mouse position in screen.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<code>in, out</code>	<code>X</code>	The X coordinate of the mouse relative to screen position.
<code>in, out</code>	<code>Y</code>	The Y coordinate of the mouse relative to screen position.

Definition at line 362 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [ScreenMousePosition](#).

```
00363 {
00364     if (GetInstance()->IsInitialized())
00365     {
00366         X = GetInstance()->ScreenMousePosition[0];
```

```

00367         Y = GetInstance()->ScreenMousePosition[1];
00368         return FOUNDATION_OKAY;
00369     }
00370
00371     PrintErrorMessage(ERROR_NOTINITIALIZED);
00372     return FOUNDATION_ERROR;
00373
00374 }

```

### 3.2.3.12 GLuint \* WindowManager::GetMousePositionInScreen ( ) [static]

Gets mouse position in screen.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

null if it fails, else the mouse position in screen.

Definition at line 387 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [ScreenMousePosition](#).

```

00388 {
00389     if (GetInstance()->IsInitialized())
00390     {
00391         return GetInstance()->ScreenMousePosition;
00392     }
00393
00394     PrintErrorMessage(ERROR_NOTINITIALIZED);
00395     return nullptr;
00396 }

```

### 3.2.3.13 GLboolean WindowManager::GetMousePositionInWindow ( const char \* *WindowName*, GLuint & X, GLuint & Y ) [static]

Gets mouse position in window.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

	<i>WindowName</i>	Name of the window.
--	-------------------	---------------------



in, out	X	The X coordinate of the mouse position relative to window coordinates.
in, out	Y	The Y coordinate of the mouse position relative to window coordinates.

Definition at line 835 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetMousePosition\(\)](#), and [GetWindowByName\(\)](#).

```

00836 {
00837     if (DoesExist (WindowName) )
00838     {
00839         return GetWindowByName (WindowName) ->GetMousePosition (X, Y);
00840     }
00841
00842     return FOUNDATION_ERROR;
00843 }
```

### 3.2.3.14 GLboolean WindowManager::GetMousePositionInWindow ( GLuint *WindowIndex*, GLint & *X*, GLint & *Y* ) [static]

Gets mouse position in window.

Author

Ziyad

Date

29/11/2014

Parameters

	<i>WindowIndex</i>	Zero-based index of the window.
in, out	X	The X coordinate of the mouse position relative to window coordinates.
in, out	Y	The Y coordinate of the mouse position relative to window coordinates.

Definition at line 858 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetMousePosition\(\)](#), and [GetWindowByIndex\(\)](#).

```

00859 {
00860     if (DoesExist (WindowIndex) )
00861     {
00862         return GetWindowByIndex (WindowIndex) ->GetMousePosition (X, Y);
00863     }
00864
00865     return FOUNDATION_ERROR;
00866 }
```

### 3.2.3.15 GLint \* WindowManager::GetMousePositionInWindow ( const char \* *WindowName* ) [static]

Gets mouse position in window.

Author

Ziyad

Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

## Returns

null if it fails, else the mouse position in window. MousePosition[0] will always return the X coordinate of the mouse relative to screen coordinates and WindowPosition[1] will always return the Y coordinate of the mouse relative to screen coordinates.

Definition at line 883 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetMousePosition\(\)](#), and [GetWindowByName\(\)](#).

```

00884 {
00885     if (DoesExist (WindowName))
00886     {
00887         return GetWindowByName (WindowName) ->GetMousePosition ();
00888     }
00889     return nullptr;
00890 }
00891 }
```

### 3.2.3.16 GLuint \* WindowManager::GetMousePositionInWindow ( GLuint WindowIndex ) [static]

Gets mouse position in window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

## Returns

null if it fails, else the mouse position in window. MousePosition[0] will always return the X coordinate of the mouse relative to screen coordinates and WindowPosition[1] will always return the Y coordinate of the mouse relative to screen coordinates.

Definition at line 908 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [ERROR\\_INVALIDWINDOWINDEX](#), [FWindow::GetMousePosition\(\)](#), [GetWindowByIndex\(\)](#), and [PrintErrorMessage\(\)](#).

```

00909 {
00910     if (DoesExist (WindowIndex))
00911     {
00912         return GetWindowByIndex (WindowIndex) ->GetMousePosition ();
00913     }
00914     PrintErrorMessage (ERROR_INVALIDWINDOWINDEX);
00915     return nullptr;
00916 }
```

**3.2.3.17 GLuint WindowManager::GetNumWindows ( ) [static]**

Gets the number of windows.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

The number of windows in the manager.

Definition at line 304 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).

Referenced by [main\(\)](#).

```
00305 {
00306     if (GetInstance()->IsInitialized())
00307     {
00308         return GetInstance()->Windows.size();
00309     }
00310
00311     PrintErrorMessage(ERROR_NOTINITIALIZED);
00312     return FOUNDATION_ERROR;
00313 }
```

**3.2.3.18 GLuint \* WindowManager::GetScreenResolution ( ) [static]**

Gets screen resolution.

**Author**

Ziyad

**Date**

29/11/2014

**Returns**

null if it fails, else the screen resolution. ScreenResolution[0] will always the width of the screen and ScreenResolution[1] will always return the height of the screen.

Definition at line 435 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), [PrintErrorMessage\(\)](#), and [ScreenResolution](#).

```
00436 {
00437     if (GetInstance()->IsInitialized())
00438     {
00439         #if defined(CURRENT_OS_WINDOWS)
00440             RECT l_Screen;
00441             HWND m_Desktop = GetDesktopWindow();
00442             GetWindowRect(m_Desktop, &l_Screen);
00443
00444             GetInstance()->ScreenResolution[0] = l_Screen.right;
00445             GetInstance()->ScreenResolution[1] = l_Screen.bottom;
00446         #endif
00447     }
00448 }
```

```

00446         return GetInstance()->ScreenResolution;
00447
00448 #endif
00449
00450 #if defined(CURRENT_OS_LINUX)
00451     GetInstance()->ScreenResolution[0] = WidthOfScreen(
XDefaultScreenOfDisplay(GetInstance()->m_Display));
00452     GetInstance()->ScreenResolution[1] = HeightOfScreen(
XDefaultScreenOfDisplay(GetInstance()->m_Display));
00453
00454     return GetInstance()->ScreenResolution;
00455 #endif
00456 }
00457 PrintErrorMessage(ERROR_NOTINITIALIZED);
00458 return nullptr;
00459
00460 }

```

### 3.2.3.19 GLboolean WindowManager::GetScreenResolution ( GLuint & *Width*, GLuint & *Height* ) [static]

Gets screen resolution.

Author

Ziyad

Date

29/11/2014

Parameters

in, out	<i>Width</i>	The width.
in, out	<i>Height</i>	The height.

Definition at line 500 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [GetInstance\(\)](#), [Is-Initialized\(\)](#), [PrintErrorMessage\(\)](#), and [ScreenResolution](#).

```

00501 {
00502     if (GetInstance()->IsInitialized())
00503     {
00504 #if defined(CURRENT_OS_WINDOWS)
00505
00506         RECT l_Screen;
00507         HWND m_Desktop = GetDesktopWindow();
00508         GetWindowRect(m_Desktop, &l_Screen);
00509         Width = l_Screen.right;
00510         Height = l_Screen.bottom;
00511 #endif
00512
00513 #if defined(CURRENT_OS_LINUX)
00514
00515         Width = WidthOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
00516         Height = HeightOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
00517
00518
00519         GetInstance()->ScreenResolution[0] = Width;
00520         GetInstance()->ScreenResolution[1] = Height;
00521 #endif
00522
00523         return FOUNDATION_OKAY;
00524     }
00525     PrintErrorMessage(ERROR_NOTINITIALIZED);
00526     return FOUNDATION_ERROR;
00527 }

```

### 3.2.3.20 FWindow \* WindowManager::GetWindowByIndex ( GLuint *WindowIndex* ) [static]

Gets window by index.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

**Returns**

null if it fails, else the window by index.

Definition at line 134 of file [WindowManager.cpp](#).References [DoesExist\(\)](#), [ERROR\\_WINDOWNOTFOUND](#), [FOUNDATION\\_ERROR](#), [GetInstance\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).Referenced by [DisableWindowDecorator\(\)](#), [EnableWindowDecorator\(\)](#), [FocusWindow\(\)](#), [GetMousePositionInWindow\(\)](#), [GetWindowIsFullScreen\(\)](#), [GetWindowIsInFocus\(\)](#), [GetWindowIsMaximized\(\)](#), [GetWindowIsMinimized\(\)](#), [GetWindowName\(\)](#), [GetWindowPosition\(\)](#), [GetWindowResolution\(\)](#), [GetWindowShouldClose\(\)](#), [main\(\)](#), [MaximizeWindow\(\)](#), [MinimizeWindow\(\)](#), [RestoreWindow\(\)](#), [SetFullScreen\(\)](#), [SetMousePositionInWindow\(\)](#), [SetWindowOnDestroyed\(\)](#), [SetWindowOnFocus\(\)](#), [SetWindowOnKeyEvent\(\)](#), [SetWindowOnMaximized\(\)](#), [SetWindowOnMinimized\(\)](#), [SetWindowOnMouseButtonEvent\(\)](#), [SetWindowOnMouseMove\(\)](#), [SetWindowOnMouseWheelEvent\(\)](#), [SetWindowOnMoved\(\)](#), [SetWindowOnResize\(\)](#), [SetWindowPosition\(\)](#), [SetWindowResolution\(\)](#), [SetWindowStyle\(\)](#), [SetWindowSwapInterval\(\)](#), [SetWindowTitleBar\(\)](#), [WindowGetKey\(\)](#), and [WindowSwapBuffers\(\)](#).

```

00135 {
00136     if (DoesExist(WindowIndex))
00137     {
00138         #if defined(CURRENT_OS_WINDOWS)
00139             for each (auto CurrentWindow in GetInstance()->Windows)
00140             {
00141                 if (CurrentWindow->ID == WindowIndex)
00142                 {
00143                     return CurrentWindow;
00144                 }
00145             }
00146         #endif
00147         #if defined(CURRENT_OS_LINUX)
00148             for (auto CurrentWindow : GetInstance()->Windows)
00149             {
00150                 if (CurrentWindow->ID == WindowIndex)
00151                 {
00152                     return CurrentWindow;
00153                 }
00154             }
00155         #endif
00156         PrintErrorMessage(ERROR_WINDOWNOTFOUND);
00157         return nullptr;
00158     }
00159     return FOUNDATION_ERROR;
00160 }
00161
00162 }
```

**3.2.3.21 FWindow \* WindowManager::GetWindowByName ( const char \* WindowName ) [static]**

Gets window by name.

get a pointer to a window via name or index

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

**Returns**

null if it fails, else the window by name.

Definition at line 92 of file [WindowManager.cpp](#).References [DoesExist\(\)](#), [ERROR\\_WINDOWNOTFOUND](#), [GetInstance\(\)](#), [PrintErrorMessage\(\)](#), and [Windows](#).

Referenced by [DisableWindowDecorator\(\)](#), [EnableWindowDecorator\(\)](#), [FocusWindow\(\)](#), [GetMousePositionInWindow\(\)](#), [GetWindowIndex\(\)](#), [GetWindowsFullScreen\(\)](#), [GetWindowsInFocus\(\)](#), [GetWindowsMaximized\(\)](#), [GetWindowsMinimized\(\)](#), [GetWindowPosition\(\)](#), [GetWindowResolution\(\)](#), [GetWindowShouldClose\(\)](#), [MaximizeWindow\(\)](#), [MinimizeWindow\(\)](#), [RestoreWindow\(\)](#), [SetFullScreen\(\)](#), [SetMousePositionInWindow\(\)](#), [SetWindowOnDestroyed\(\)](#), [SetWindowOnFocus\(\)](#), [SetWindowOnKeyEvent\(\)](#), [SetWindowOnMaximized\(\)](#), [SetWindowOnMinimized\(\)](#), [SetWindowOnMouseButtonEvent\(\)](#), [SetWindowOnMouseMove\(\)](#), [SetWindowOnMouseWheelEvent\(\)](#), [SetWindowOnMoved\(\)](#), [SetWindowOnResize\(\)](#), [SetWindowPosition\(\)](#), [SetWindowResolution\(\)](#), [SetWindowStyle\(\)](#), [SetWindowSwapInterval\(\)](#), [SetWindowTitleBar\(\)](#), [WindowGetKey\(\)](#), and [WindowSwapBuffers\(\)](#).

```

00093 {
00094     if (DoesExist(WindowName))
00095     {
00096         #if defined(CURRENT_OS_WINDOWS)
00097             for each(auto CurrentWindow in GetInstance()->Windows)
00098             {
00099                 if (CurrentWindow->Name == WindowName)
00100                 {
00101                     return CurrentWindow;
00102                 }
00103             }
00104         #endif
00105         #if defined(CURRENT_OS_LINUX)
00106             for (auto CurrentWindow : GetInstance()->Windows)
00107             {
00108                 if (CurrentWindow->Name == WindowName)
00109                 {
00110                     return CurrentWindow;
00111                 }
00112             }
00113         #endif
00114         PrintErrorMessage(ERROR_WINDOWNOTFOUND);
00115         return nullptr;
00116     }
00117     return nullptr;
00118 }
00119 }
```

**3.2.3.22 GLuint WindowManager::GetWindowIndex ( const char \* *WindowName* ) [static]**

Gets window index.

**Author**

Ziyad

**Date**

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

## Returns

The window index.

Definition at line 1406 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [GetWindowByName\(\)](#), and [FWindow::ID](#).

```

01407 {
01408     if (DoesExist (WindowName))
01409     {
01410         return GetWindowByName (WindowName) ->ID;
01411     }
01412
01413     return 0;
01414 }
```

### 3.2.3.23 GLboolean WindowManager::GetWindowIsFullScreen ( const char \* *WindowName* ) [static]

Gets whether the window is full screen.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

## Returns

Whether the window is full screen.

Definition at line 1113 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetIsFullScreen\(\)](#), and [GetWindowByName\(\)](#).

```

01114 {
01115     if (DoesExist (WindowName))
01116     {
01117         return GetWindowByName (WindowName) ->GetIsFullScreen ();
01118     }
01119
01120     return FOUNDATION_ERROR;
01121 }
```

### 3.2.3.24 GLboolean WindowManager::GetWindowIsFullScreen ( GLuint *WindowIndex* ) [static]

Gets whether the window is full screen.

## Author

Ziyad

## Date

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

**Returns**

Whether the window is full screen.

Definition at line 1136 of file [WindowManager.cpp](#).

References [FOUNDATION\\_ERROR](#), [GetInstance\(\)](#), [FWindow::GetIsFullScreen\(\)](#), [GetWindowByIndex\(\)](#), and [Windows](#).

```

01137 {
01138     if (WindowIndex <= GetInstance()->Windows.size() -1)
01139     {
01140         return GetWindowByIndex(WindowIndex)->GetIsFullScreen();
01141     }
01142     return FOUNDATION_ERROR;
01143 }
01144 }
```

**3.2.3.25 GLboolean WindowManager::GetWindowIsInFocus ( const char \* *WindowName* ) [static]**

Gets window is in focus.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

**Returns**

The window is in focus.

Definition at line 1473 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetInFocus\(\)](#), and [GetWindowByName\(\)](#).

```

01474 {
01475     if (DoesExist (WindowName) )
01476     {
01477         return GetWindowByName (WindowName)->GetInFocus();
01478     }
01479     return FOUNDATION_ERROR;
01480 }
01481 }
```

**3.2.3.26 GLboolean WindowManager::GetWindowIsInFocus ( GLuint *WindowIndex* ) [static]**

Gets window is in focus.

**Author**

Ziyad

**Date**

29/11/2014



## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

## Returns

The window is in focus.

Definition at line 1496 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetInFocus\(\)](#), and [GetWindowByIndex\(\)](#).

```
01497 {
01498     if (DoesExist (WindowIndex) )
01499     {
01500         return GetWindowByIndex (WindowIndex) ->GetInFocus () ;
01501     }
01502
01503     return FOUNDATION_ERROR;
01504 }
```

### 3.2.3.27 GLboolean WindowManager::GetWindowIsMaximized ( const char \* *WindowName* ) [static]

Gets window is maximized.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

## Returns

The window is maximized.

Definition at line 1293 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetIsMaximized\(\)](#), and [GetWindowByName\(\)](#).

```
01294 {
01295     if (DoesExist (WindowName) )
01296     {
01297         return GetWindowByName (WindowName) ->GetIsMaximized () ;
01298     }
01299
01300     return FOUNDATION_ERROR;
01301 }
```

### 3.2.3.28 GLboolean WindowManager::GetWindowIsMaximized ( GLuint *WindowIndex* ) [static]

Gets window is maximized.

## Author

Ziyad

## Date

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

**Returns**

The window is maximized.

Definition at line 1316 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetIsMaximized\(\)](#), and [GetWindowByIndex\(\)](#).

```

01317 {
01318     if (DoesExist (WindowIndex) )
01319     {
01320         return GetWindowByIndex (WindowIndex) ->GetIsMaximized () ;
01321     }
01322
01323     return FOUNDATION_ERROR;
01324 }
```

### 3.2.3.29 GLboolean WindowManager::GetWindowIsMinimized ( const char \* *WindowName* ) [static]

Gets window is minimized.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

**Returns**

The window is minimized.

Definition at line 1203 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetIsMinimized\(\)](#), and [GetWindowByName\(\)](#).

```

01204 {
01205     if (DoesExist (WindowName) )
01206     {
01207         return GetWindowByName (WindowName) ->GetIsMinimized () ;
01208     }
01209
01210     return FOUNDATION_ERROR;
01211 }
```

### 3.2.3.30 GLboolean WindowManager::GetWindowIsMinimized ( GLuint *WindowIndex* ) [static]

Gets window is minimized.

**Author**

Ziyad

**Date**

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

## Returns

The window is minimized.

Definition at line 1226 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetIsMinimized\(\)](#), and [GetWindowByIndex\(\)](#).

```

01227 {
01228     if (DoesExist (WindowIndex))
01229     {
01230         return GetWindowByIndex (WindowIndex) ->GetIsMinimized ();
01231     }
01232
01233     return FOUNDATION_ERROR;
01234 }
```

### 3.2.3.31 const char \* WindowManager::GetWindowName ( GLuint *WindowIndex* ) [static]

Gets window name.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

## Returns

null if it fails, else the window name.

Definition at line 1383 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [GetWindowByIndex\(\)](#), and [FWindow::GetWindowName\(\)](#).

```

01384 {
01385     if (DoesExist (WindowIndex))
01386     {
01387         return GetWindowByIndex (WindowIndex) ->GetWindowName ();
01388     }
01389
01390     return nullptr;
01391 }
```

### 3.2.3.32 GLboolean WindowManager::GetWindowPosition ( const char \* *WindowName*, GLuint & X, GLuint & Y ) [static]

Gets window position relative to screen coordinates.

## Author

Ziyad

## Date

29/11/2014

**Parameters**

	<i>WindowName</i>	Name of the window.
<i>in, out</i>	<i>X</i>	The X coordinate of the window relative to screen coordinates.
<i>in, out</i>	<i>Y</i>	The Y coordinate of the window relative to screen coordinates.

Definition at line 693 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetPosition\(\)](#), and [GetWindowByName\(\)](#).

```

00694 {
00695     if (DoesExist (WindowName) )
00696     {
00697         return GetWindowByName (WindowName) ->GetPosition (X, Y);
00698     }
00699
00700     return GL_FALSE;
00701 }
```

### 3.2.3.33 GLboolean WindowManager::GetWindowPosition ( GLuint *WindowIndex*, GLuint & *X*, GLuint & *Y* ) [static]

Gets window position relative to screen coordinates.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

	<i>WindowIndex</i>	Zero-based index of the window.
<i>in, out</i>	<i>X</i>	The X coordinate of the window relative to screen coordinates.
<i>in, out</i>	<i>Y</i>	The Y coordinate of the window relative to screen coordinates.

Definition at line 716 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetPosition\(\)](#), and [GetWindowByIndex\(\)](#).

```

00717 {
00718     if (DoesExist (WindowIndex) )
00719     {
00720         return GetWindowByIndex (WindowIndex) ->GetPosition (X, Y);
00721     }
00722
00723     return GL_FALSE;
00724 }
```

### 3.2.3.34 GLuint \* WindowManager::GetWindowPosition ( const char \* *WindowName* ) [static]

Gets window position relative to screen coordinates.

**Author**

Ziyad

**Date**

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

## Returns

null if it fails, else the window position relative to screen coordinates. Position[0] will always return the X coordinate of the window and Position[1] will always return the Y coordinate of the window.

Definition at line 741 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetPosition\(\)](#), and [GetWindowByName\(\)](#).

```

00742 {
00743     if (DoesExist (WindowName))
00744     {
00745         return GetWindowByName (WindowName) ->GetPosition();
00746     }
00747
00748     return nullptr;
00749 }
```

### 3.2.3.35 GLuint \* WindowManager::GetWindowPosition ( GLuint *WindowIndex* ) [static]

Gets window position relative to screen coordinates.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

## Returns

null if it fails, else the window position relative to screen coordinates. Position[0] will always return the X coordinate of the window and Position[1] will always return the Y coordinate of the window.

Definition at line 766 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), [FWindow::GetPosition\(\)](#), [GetWindowByIndex\(\)](#), and [Windows](#).

```

00767 {
00768     if (WindowIndex <= GetInstance() ->Windows.size() -1)
00769     {
00770         return GetWindowByIndex (WindowIndex) ->GetPosition();
00771     }
00772
00773     return nullptr;
00774 }
```

### 3.2.3.36 GLboolean WindowManager::GetWindowResolution ( const char \* *WindowName*, GLuint & *Width*, GLuint & *Height* ) [static]

Gets window resolution.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

	<i>WindowName</i>	Name of the window.
in, out	<i>Width</i>	The width.
in, out	<i>Height</i>	The height.

Definition at line 542 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [GetInstance\(\)](#), [GetWindowByName\(\)](#), [IsInitialized\(\)](#), and [PrintErrorMessage\(\)](#).

```

00543 {
00544     if (GetInstance()->IsInitialized())
00545     {
00546         if (DoesExist(WindowName))
00547         {
00548             if (GetWindowByName(WindowName)->GetResolution(Width, Height))
00549             {
00550                 return FOUNDATION_OKAY;
00551             }
00552             return FOUNDATION_ERROR;
00553         }
00554         return FOUNDATION_ERROR;
00555     }
00556
00557     PrintErrorMessage(ERROR_NOTINITIALIZED);
00558     return FOUNDATION_ERROR;
00559 }
```

### 3.2.3.37 GLboolean WindowManager::GetWindowResolution ( GLuint *WindowIndex*, GLuint & *Width*, GLuint & *Height* ) [static]

Gets window resolution.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

	<i>WindowIndex</i>	Zero-based index of the window.
in, out	<i>Width</i>	The width.
in, out	<i>Height</i>	The height.

Definition at line 574 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [FWindow::GetResolution\(\)](#), [GetWindowByIndex\(\)](#), and [PrintErrorMessage\(\)](#).

```

00575 {
00576     if (DoesExist(WindowIndex))
00577     {
00578         GetWindowByIndex(WindowIndex)->GetResolution(Width, Height);
00579         return FOUNDATION_OKAY;
00580     }
00581
00582     PrintErrorMessage(ERROR_NOTINITIALIZED);
00583     return FOUNDATION_ERROR;
00584 }
```

**3.2.3.38** `GLuint * WindowManager::GetWindowResolution ( const char * WindowName ) [static]`

Gets window resolution as an array.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

**Returns**

null if it fails, else the window resolution. Resolution[0] will always return the width of the window and Resolution[1] will always return the height of the window.

Definition at line 600 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetResolution\(\)](#), and [GetWindowByName\(\)](#).

```
00601 {  
00602     if (DoesExist (WindowName) )  
00603     {  
00604         return GetWindowByName (WindowName) ->GetResolution ();  
00605     }  
00606     return nullptr;  
00607 }  
00608 }
```

**3.2.3.39** `GLuint * WindowManager::GetWindowResolution ( GLuint WindowIndex ) [static]`

Gets window resolution.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

**Returns**

null if it fails, else the window resolution. Resolution[0] will always return the width of the window and Resolution[1] will always return the height of the window.

Definition at line 624 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::GetResolution\(\)](#), and [GetWindowByIndex\(\)](#).

```
00625 {  
00626     if (DoesExist (WindowIndex) )  
00627     {  
00628         return GetWindowByIndex (WindowIndex) ->GetResolution ();  
00629     }  
00630     return nullptr;  
00631 }  
00632 }
```

### 3.2.3.40 GLboolean WindowManager::GetWindowShouldClose ( const char \* *WindowName* ) [static]

Gets whether the window should close.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
-------------------	---------------------

Returns

Whether the window should close.

Definition at line 1025 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetShouldClose\(\)](#), and [GetWindowByName\(\)](#).

Referenced by [main\(\)](#).

```
01026 {  
01027     if (DoesExist (WindowName) )  
01028     {  
01029         return GetWindowByName (WindowName) ->GetShouldClose () ;  
01030     }  
01031  
01032     return FOUNDATION_ERROR;  
01033 }
```

### 3.2.3.41 GLboolean WindowManager::GetWindowShouldClose ( GLuint *WindowIndex* ) [static]

Gets whether the window should close.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

Returns

Whether the window should close.

Definition at line 1048 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetShouldClose\(\)](#), and [GetWindowByIndex\(\)](#).

```
01049 {  
01050     if (DoesExist (WindowIndex) )  
01051     {  
01052         return GetWindowByIndex (WindowIndex) ->GetShouldClose () ;  
01053     }  
01054  
01055     return FOUNDATION_ERROR;  
01056 }
```



### 3.2.3.42 GLboolean WindowManager::Initialize ( ) [static]

Initializes this object.

#### Author

Ziyad

#### Date

29/11/2014

Definition at line 32 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), and [Initialized](#).

Referenced by [main\(\)](#).

```
00033 {  
00034     GetInstance\(\)->Initialized = GL_FALSE;  
00035     if defined(CURRENT_OS_LINUX)  
00036         return Linux\_Initialize\(\);  
00037     endif  
00038  
00039     if defined(CURRENT_OS_WINDOWS)  
00040         return Windows\_Initialize\(\);  
00041     endif  
00042 }
```

### 3.2.3.43 GLboolean WindowManager::IsInitialized ( ) [static]

Definition at line 44 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), and [Initialized](#).

Referenced by [AddWindow\(\)](#), [DoesExist\(\)](#), [GetMousePositionInScreen\(\)](#), [GetNumWindows\(\)](#), [GetScreen-Resolution\(\)](#), [GetWindowResolution\(\)](#), and [PollForEvents\(\)](#).

```
00045 {  
00046     return GetInstance\(\)->Initialized;  
00047 }
```

### 3.2.3.44 GLboolean WindowManager::MaximizeWindow ( const char \* *WindowName*, GLboolean *ShouldBeMaximized* ) [static]

Maximize window.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

---

<i>WindowName</i>	Name of the window.
<i>ShouldBeMaximized</i>	Whether the window should be maximized.

Definition at line 1338 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByName\(\)](#).

```

01339 {
01340     if (DoesExist(WindowName))
01341     {
01342         return GetWindowByName(WindowName)->FullScreen(ShouldBeMaximized);
01343     }
01344     return FOUNDATION_ERROR;
01345 }
01346 }
```

### 3.2.3.45 GLboolean WindowManager::MaximizeWindow ( GLuint *WindowIndex*, GLboolean *ShouldBeMaximized* ) [static]

Maximize window.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>ShouldBeMaximized</i>	Whether the window should be maximized.

Definition at line 1360 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByIndex\(\)](#).

```

01361 {
01362     if (DoesExist(WindowIndex))
01363     {
01364         return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMaximized);
01365     }
01366     return FOUNDATION_ERROR;
01367 }
01368 }
```

### 3.2.3.46 GLboolean WindowManager::MinimizeWindow ( const char \* *WindowName*, GLboolean *ShouldBeMinimized* ) [static]

set the window to be minimized depending on NewState.

Author

Ziyad

Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
<i>ShouldBeMinimized</i>	whether the window should be minimized.

Definition at line 1248 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByName\(\)](#).

```

01249 {
01250     if (DoesExist (WindowName) )
01251     {
01252         return GetWindowByName (WindowName) ->FullScreen (ShouldBeMinimized);
01253     }
01254
01255     return FOUNDATION_ERROR;
01256 }
```

### 3.2.3.47 GLboolean WindowManager::MinimizeWindow ( GLuint WindowIndex, GLboolean ShouldBeMinimized ) [static]

Minimize window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>ShouldBeMinimized</i>	Whether the window should be minimized.

Definition at line 1270 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByIndex\(\)](#).

```

01271 {
01272     if (DoesExist (WindowIndex) )
01273     {
01274         return GetWindowByIndex (WindowIndex) ->FullScreen (ShouldBeMinimized);
01275     }
01276
01277     return FOUNDATION_ERROR;
01278 }
```

### 3.2.3.48 GLboolean WindowManager::PollForEvents ( ) [static]

Poll for events for all windows in the manager.

## Author

Ziyad

**Date**

29/11/2014

Definition at line 471 of file [WindowManager.cpp](#).

References [ERROR\\_NOTINITIALIZED](#), [FOUNDATION\\_ERROR](#), [GetInstance\(\)](#), [IsInitialized\(\)](#), and [PrintErrorMessage\(\)](#).

Referenced by [main\(\)](#).

```

00472 {
00473     if (GetInstance()->IsInitialized())
00474     {
00475         #if defined(CURRENT_OS_WINDOWS)
00476             return GetInstance()->Windows_PollForEvents();
00477         #endif
00478
00479         #if defined (CURRENT_OS_LINUX)
00480             return GetInstance()->Linux_PollForEvents();
00481         #endif
00482     }
00483
00484     PrintErrorMessage(ERROR_NOTINITIALIZED);
00485     return FOUNDATION_ERROR;
00486 }
```

**3.2.3.49** static GLboolean WindowManager::RemoveWindow ( FWindow \* *WindowToBeRemoved* ) [static]

**3.2.3.50** GLboolean WindowManager::RestoreWindow ( const char \* *WindowName* ) [static]

Restore window.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

Definition at line 1561 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::Restore\(\)](#).

```

01562 {
01563     if (DoesExist (WindowName) )
01564     {
01565         return GetWindowByName (WindowName)->Restore();
01566     }
01567     return FOUNDATION_ERROR;
01568     //implement window focusing
01569 }
```

**3.2.3.51** GLboolean WindowManager::RestoreWindow ( GLuint *WindowIndex* ) [static]

Restore window.

**Author**

Ziyad

**Date**

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

Definition at line 1582 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::Restore\(\)](#).

```

01583 {
01584     if (DoesExist (WindowIndex))
01585     {
01586         return GetWindowByIndex (WindowIndex) ->Restore ();
01587     }
01588     return FOUNDATION_ERROR;
01589     //implement window focusing
01590 }
01591 }
```

### 3.2.3.52 GLboolean WindowManager::SetFullScreen ( const char \* *WindowName*, GLboolean *ShouldBeFullscreen* ) [static]

toggle the fullscreen mode for the window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
<i>ShouldBeFullscreen</i>	whether the window should be in fullscreen mode.

Definition at line 1158 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByName\(\)](#).

```

01159 {
01160     if (DoesExist (WindowName))
01161     {
01162         return GetWindowByName (WindowName) ->FullScreen (ShouldBeFullscreen);
01163     }
01164     return FOUNDATION_ERROR;
01165 }
01166 }
```

### 3.2.3.53 GLboolean WindowManager::SetFullScreen ( GLuint *WindowIndex*, GLboolean *ShouldBeFullscreen* ) [static]

toggle the fullscreen mode for the window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>ShouldBeFullscreen</i>	whether the window should be in fullscreen mode.

Definition at line 1180 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::FullScreen\(\)](#), and [GetWindowByIndex\(\)](#).

```

01181 {
01182     if (DoesExist (WindowIndex))
01183     {
01184         return GetWindowByIndex (WindowIndex) ->FullScreen (ShouldBeFullscreen);
01185     }
01186     return FOUNDATION_ERROR;
01187 }
01188 }
```

### 3.2.3.54 GLboolean WindowManager::SetMousePositionInScreen ( GLuint X, GLuint Y ) [static]

Sets mouse position in screen.

## Author

Ziyad

## Date

29/11/2014

## Parameters

X	The new X position of the mouse cursor relative to screen coordinates.
Y	The new Y position of the mouse cursor relative to screen coordinates.

Definition at line 410 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), and [ScreenMousePosition](#).

```

00411 {
00412     GetInstance ()->ScreenMousePosition[0] = X;
00413     GetInstance ()->ScreenMousePosition[1] = Y;
00414     #if defined(CURRENT_OS_WINDOWS)
00415     return Windows_SetMousePositionInScreen(X, Y);
00416     #endif
00417     #if defined(CURRENT_OS_LINUX)
00418     return Linux_SetMousePositionInScreen(X, Y);
00419     #endif
00420     #endif
00421 }
```

### 3.2.3.55 GLboolean WindowManager::SetMousePositionInWindow ( const char \* WindowName, GLuint X, GLuint Y ) [static]

Sets mouse position in window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
<i>X</i>	The new X coordinate of the mouse position relative to window coordinates.
<i>Y</i>	The new Y coordinate of the mouse position relative to window coordinates.

Definition at line 931 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetMousePosition\(\)](#).

```

00932 {
00933     if (DoesExist (WindowName) )
00934     {
00935         return GetWindowByName (WindowName) ->SetMousePosition (X, Y);
00936     }
00937
00938     return FOUNDATION_ERROR;
00939 }
```

### 3.2.3.56 GLboolean WindowManager::SetMousePositionInWindow ( GLuint *WindowIndex*, GLuint *X*, GLuint *Y* ) [static]

Sets mouse position in window.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>X</i>	The new X coordinate of the mouse position relative to window coordinates.
<i>Y</i>	The new Y coordinate of the mouse position relative to window coordinates.

Definition at line 954 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetMousePosition\(\)](#).

```

00955 {
00956     if (DoesExist (WindowIndex) )
00957     {
00958         return GetWindowByIndex (WindowIndex) ->SetMousePosition (X, Y);
00959     }
00960
00961     return FOUNDATION_ERROR;
00962 }
```

### 3.2.3.57 static GLboolean WindowManager::SetWindowIcon ( const char \* *WindowName*, const char \* *Icon*, GLuint *Width*, GLuint *Height* ) [static]

### 3.2.3.58 static GLboolean WindowManager::SetwindowIcon ( GLuint *WindowIndex*, const char \* *Icon*, GLuint *Width*, GLuint *Height* ) [static]

### 3.2.3.59 GLboolean WindowManager::SetWindowOnDestroyed ( const char \* *WindowName*, OnDestroyedEvent *OnDestroyed* ) [static]

Sets window on destroyed.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
<i>OnDestroyed</i>	The on destroyed.

Definition at line 1842 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnDestroyed\(\)](#).

```

01843 {
01844     if (DoesExist (WindowName) )
01845     {
01846         return GetWindowByName (WindowName) ->SetOnDestroyed (OnDestroyed) ;
01847     }
01848
01849     return FOUNDATION_ERROR;
01850 }
```

### 3.2.3.60 GLboolean WindowManager::SetWindowOnDestroyed ( GLuint *WindowIndex*, OnDestroyedEvent *OnDestroyed* ) [static]

Sets window on destroyed.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnDestroyed</i>	The on destroyed.

Definition at line 1864 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnDestroyed\(\)](#).

```

01865 {
01866     if (DoesExist (WindowIndex) )
01867     {
01868         return GetWindowByIndex (WindowIndex) ->SetOnDestroyed (OnDestroyed) ;
01869     }
01870
01871     return FOUNDATION_ERROR;
01872 }
```

### 3.2.3.61 GLboolean WindowManager::SetWindowOnFocus ( const char \* *WindowName*, OnFocusEvent *OnFocus* ) [static]

Sets window on focus.



## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowName</i>	Name of the window.
<i>OnFocus</i>	The on focus.

Definition at line 1990 of file [WindowManager.cpp](#).

References [FWindow::FocusEvent](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), [GetWindowByName\(\)](#), and [IsValidString\(\)](#).

```
01991 {  
01992     if (IsValidString (WindowName) )  
01993     {  
01994         GetWindowByName (WindowName) ->FocusEvent = OnFocus;  
01995         return FOUNDATION_OKAY;  
01996     }  
01997     return FOUNDATION_ERROR;  
01998 }  
01999 }
```

### 3.2.3.62 GLboolean WindowManager::SetWindowOnFocus ( GLuint WindowIndex, OnFocusEvent OnFocus ) [static]

Sets window on focus.

## Author

Ziyad

## Date

29/11/2014

## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnFocus</i>	The on focus.

Definition at line 2013 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FWindow::FocusEvent](#), [FOUNDATION\\_ERROR](#), [FOUNDATION\\_OKAY](#), and [GetWindowByIndex\(\)](#).

```
02014 {  
02015     if (DoesExist (WindowIndex) )  
02016     {  
02017         GetWindowByIndex (WindowIndex) ->FocusEvent = OnFocus;  
02018         return FOUNDATION_OKAY;  
02019     }  
02020     return FOUNDATION_ERROR;  
02021 }  
02022 }
```

### 3.2.3.63 GLboolean WindowManager::SetWindowOnKeyEvent ( const char \* *WindowName*, OnKeyEvent *OnKey* ) [static]

Sets window on key event.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowName</i>	Name of the window.
<i>OnKey</i>	The on key event.

Definition at line 1710 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnKeyEvent\(\)](#).

Referenced by [main\(\)](#).

```

01711 {
01712     if (DoesExist (WindowName) )
01713     {
01714         return GetWindowByName (WindowName) ->SetOnKeyEvent (OnKey) ;
01715     }
01716
01717     return FOUNDATION_ERROR;
01718 }
```

### 3.2.3.64 GLboolean WindowManager::SetWindowOnKeyEvent ( GLuint *WindowIndex*, OnKeyEvent *OnKey* ) [static]

Sets window on key event.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnKey</i>	The on key event.

Definition at line 1732 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnKeyEvent\(\)](#).

```

01733 {
01734     if (DoesExist (WindowIndex) )
01735     {
01736         return GetWindowByIndex (WindowIndex) ->SetOnKeyEvent (OnKey) ;
01737     }
01738
01739     return FOUNDATION_ERROR;
01740 }
```

### 3.2.3.65 GLboolean WindowManager::SetWindowOnMaximized ( const char \* *WindowName*, OnMaximizedEvent *OnMaximized* ) [static]

Sets window on maximized.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnMaximized</i>	The on maximized.

Definition at line 1886 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMaximized\(\)](#).

```
01887 {  
01888     if (DoesExist (WindowName) )  
01889     {  
01890         return GetWindowByName (WindowName) ->SetOnMaximized (OnMaximized) ;  
01891     }  
01892  
01893     return FOUNDATION_ERROR;  
01894 }
```

### 3.2.3.66 GLboolean WindowManager::SetWindowOnMaximized ( GLuint *WindowIndex*, OnMaximizedEvent *OnMaximized* ) [static]

Sets window on maximized.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMaximized</i>	The on maximized.

Definition at line 1908 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMaximized\(\)](#).

```
01909 {  
01910     if (DoesExist (WindowIndex) )  
01911     {  
01912         return GetWindowByIndex (WindowIndex) ->SetOnMaximized (OnMaximized) ;  
01913     }  
01914  
01915     return FOUNDATION_ERROR;  
01916 }
```

### 3.2.3.67 GLboolean WindowManager::SetWindowOnMinimized ( const char \* *WindowName*, OnMinimizedEvent *OnMinimized* ) [static]

Sets window on minimized.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnMinimized</i>	The on minimized.

Definition at line 1930 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMinimized\(\)](#).

```

01931 {
01932     if (DoesExist (WindowName) )
01933     {
01934         return GetWindowByName (WindowName) ->SetOnMinimized (OnMinimized);
01935     }
01936     return FOUNDATION_ERROR;
01937 }
01938 }
```

### 3.2.3.68 GLboolean WindowManager::SetWindowOnMinimized ( GLuint *WindowIndex*, OnMinimizedEvent *OnMinimized* ) [static]

Sets window on minimized.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMinimized</i>	The on minimized.

Definition at line 1952 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMinimized\(\)](#).

```

01953 {
01954     if (DoesExist (WindowIndex) )
01955     {
01956         return GetWindowByIndex (WindowIndex) ->SetOnMinimized (OnMinimized);
01957     }
01958     return FOUNDATION_ERROR;
01959 }
01960 }
```

### 3.2.3.69 GLboolean WindowManager::SetWindowOnMouseButtonEvent ( const char \* *WindowName*, *OnMouseButtonEvent OnMouseButton* ) [static]

Sets window on mouse button event.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnMouseButton</i>	The on mouse button event.

Definition at line 1754 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMouseButtonEvent\(\)](#).

```

01755 {
01756     if (DoesExist(WindowName))
01757     {
01758         return GetWindowByName(WindowName)->SetOnMouseButtonEvent(
OnMouseButton);
01759     }
01760
01761     return FOUNDATION_ERROR;
01762 }
```

### 3.2.3.70 GLboolean WindowManager::SetWindowOnMouseButtonEvent ( GLuint *WindowIndex*, *OnMouseButtonEvent OnMouseButton* ) [static]

Sets window on mouse button event.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMouseButton</i>	The on mouse button event.

Definition at line 1776 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMouseButtonEvent\(\)](#).

```

01777 {
01778     if (DoesExist(WindowIndex))
01779     {
01780         return GetWindowByIndex(WindowIndex)->
SetOnMouseButtonEvent(OnMouseButton);
01781     }
01782
01783     return FOUNDATION_ERROR;
01784 }
```

### 3.2.3.71 GLboolean WindowManager::SetWindowOnMouseMove ( const char \* *WindowName*, OnMouseMoveEvent *OnMouseMove* ) [static]

Sets window on mouse move.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnMouseMove</i>	The on mouse move.

Definition at line 2124 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMouseMove\(\)](#).

```

02125 {
02126     if (DoesExist (WindowName) )
02127     {
02128         return GetWindowByName (WindowName) ->SetOnMouseMove (OnMouseMove) ;
02129     }
02130
02131     return FOUNDATION_ERROR;
02132 }
```

### 3.2.3.72 GLboolean WindowManager::SetWindowOnMouseMove ( GLuint *WindowIndex*, OnMouseMoveEvent *OnMouseMove* ) [static]

Sets window on mouse move.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMouseMove</i>	The on mouse move.

Definition at line 2146 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMouseMove\(\)](#).

```

02147 {
02148     if (DoesExist (WindowIndex) )
02149     {
02150         return GetWindowByIndex (WindowIndex) ->SetOnMouseMove (OnMouseMove) ;
02151     }
02152
02153     return FOUNDATION_ERROR;
02154 }
```

### 3.2.3.73 GLboolean WindowManager::SetWindowOnMouseWheelEvent ( const char \* *WindowName*, OnMouseWheelEvent *OnMouseWheel* ) [static]

Sets window on mouse wheel event.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowName</i>	Name of the window.
<i>OnMouseWheel</i>	The on mouse wheel event.

Definition at line 1798 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMouseWheelEvent\(\)](#).

```
01799 {
01800     if (DoesExist(WindowName))
01801     {
01802         return GetWindowByName(WindowName)->SetOnMouseWheelEvent (
OnMouseWheel);
01803     }
01804
01805     return FOUNDATION_ERROR;
01806 }
```

### 3.2.3.74 GLboolean WindowManager::SetWindowOnMouseWheelEvent ( GLuint *WindowIndex*, OnMouseWheelEvent *OnMouseWheel* ) [static]

Sets window on mouse wheel event.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMouseWheel</i>	The on mouse wheel event.

Definition at line 1820 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMouseWheelEvent\(\)](#).

```
01821 {
01822     if (DoesExist(WindowIndex))
01823     {
01824         return GetWindowByIndex(WindowIndex)->
SetOnMouseWheelEvent (OnMouseWheel);
01825     }
01826
01827     return FOUNDATION_ERROR;
01828 }
```

### 3.2.3.75 GLboolean WindowManager::SetWindowOnMoved ( const char \* *WindowName*, OnMovedEvent *OnMoved* ) [static]

Sets window on moved.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnMoved</i>	The on moved.

Definition at line 2036 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnMoved\(\)](#).

```
02037 {
02038     if (DoesExist (WindowName) )
02039     {
02040         return GetWindowByName (WindowName) ->SetOnMoved (OnMoved) ;
02041     }
02042
02043     return FOUNDATION_ERROR;
02044 }
```

### 3.2.3.76 GLboolean WindowManager::SetWindowOnMoved ( GLuint *WindowIndex*, OnMovedEvent *OnMoved* ) [static]

Sets window on moved.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnMoved</i>	The on moved.

Definition at line 2058 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnMoved\(\)](#).

```
02059 {
02060     if (DoesExist (WindowIndex) )
02061     {
02062         return GetWindowByIndex (WindowIndex) ->SetOnMoved (OnMoved) ;
02063     }
02064
02065     return FOUNDATION_ERROR;
02066 }
```



### 3.2.3.77 GLboolean WindowManager::SetWindowOnResize ( const char \* *WindowName*, OnResizeEvent *OnResize* ) [static]

Sets window on resize.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>OnResize</i>	The on resize.

Definition at line 2080 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetOnResize\(\)](#).

```
02081 {  
02082     if (DoesExist (WindowName) )  
02083     {  
02084         return GetWindowByName (WindowName) ->SetOnResize (OnResize) ;  
02085     }  
02086     return FOUNDATION_ERROR;  
02087 }  
02088 }
```

### 3.2.3.78 GLboolean WindowManager::SetWindowOnResize ( GLuint *WindowIndex*, OnResizeEvent *OnResize* ) [static]

Sets window on resize.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>OnResize</i>	The on resize.

Definition at line 2102 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetOnResize\(\)](#).

```
02103 {  
02104     if (DoesExist (WindowIndex) )  
02105     {  
02106         return GetWindowByIndex (WindowIndex) ->SetOnResize (OnResize) ;  
02107     }  
02108     return FOUNDATION_ERROR;  
02109 }  
02110 }
```

### 3.2.3.79 GLboolean WindowManager::SetWindowPosition ( const char \* *WindowName*, GLuint *X*, GLuint *Y* ) [static]

Sets window position relative to screen coordinates.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowName</i>	Name of the window.
<i>X</i>	The new X coordinate of the window relative to screen coordinates.
<i>Y</i>	The new Y coordinate of the window relative to screen coordinates.

Definition at line 789 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetPosition\(\)](#).

```

00790 {
00791     if (DoesExist (WindowName))
00792     {
00793         return GetWindowByName (WindowName) ->SetPosition (X, Y);
00794     }
00795
00796     return FOUNDATION_ERROR;
00797 }
```

### 3.2.3.80 GLboolean WindowManager::SetWindowPosition ( GLuint *WindowIndex*, GLuint *X*, GLuint *Y* ) [static]

Sets window position relative to screen coordinates.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>X</i>	The new X coordinate of the window relative to screen coordinates.
<i>Y</i>	The new y coordinate of the window relative to screen coordinates.

Definition at line 812 of file [WindowManager.cpp](#).

References [FOUNDATION\\_ERROR](#), [GetInstance\(\)](#), [GetWindowByIndex\(\)](#), [FWindow::SetPosition\(\)](#), and [Windows](#).

```

00813 {
00814     if (WindowIndex <= GetInstance () ->Windows.size () -1)
00815     {
00816         return GetWindowByIndex (WindowIndex) ->SetPosition (X, Y);
00817     }
00818
00819     return FOUNDATION_ERROR;
00820 }
```

### 3.2.3.81 GLboolean WindowManager::SetWindowResolution ( const char \* *WindowName*, GLuint *Width*, GLuint *Height* ) [static]

Sets window resolution.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowName</i>	Name of the window.
<i>Width</i>	The width.
<i>Height</i>	The height.

Definition at line 647 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [GetWindowByName\(\)](#), and [FWindow::SetResolution\(\)](#).

```
00648 {
00649     if (DoesExist (WindowName) )
00650     {
00651         return GetWindowByName (WindowName)->SetResolution (Width, Height);
00652     }
00653
00654     return GL_FALSE;
00655 }
```

### 3.2.3.82 GLboolean WindowManager::SetWindowResolution ( GLuint *WindowIndex*, GLuint *Width*, GLuint *Height* ) [static]

Sets window resolution.

Author

Ziyad

Date

29/11/2014

Parameters

<i>WindowIndex</i>	Zero-based index of the window.
<i>Width</i>	The width.
<i>Height</i>	The height.

Definition at line 670 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [GetWindowByIndex\(\)](#), and [FWindow::SetResolution\(\)](#).

```
00671 {
00672     if (DoesExist (WindowIndex) )
00673     {
00674         return GetWindowByIndex (WindowIndex)->SetResolution (Width, Height);
00675     }
00676
00677     return GL_FALSE;
00678 }
```

**3.2.3.83 GLboolean WindowManager::SetWindowStyle ( const char \* *WindowName*, GLuint *WindowStyle* ) [static]**

Definition at line 1637 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetStyle\(\)](#).

Referenced by [main\(\)](#).

```
01638 {
01639     if (DoesExist(WindowName))
01640     {
01641         return GetWindowByName(WindowName)->SetStyle(WindowStyle);
01642     }
01643
01644     return FOUNDATION_ERROR;
01645 }
```

**3.2.3.84 GLboolean WindowManager::SetWindowStyle ( GLuint *WindowIndex*, GLuint *WindowStyle* ) [static]**

Definition at line 1647 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetStyle\(\)](#).

```
01648 {
01649     if (DoesExist(WindowIndex))
01650     {
01651         return GetWindowByIndex(WindowIndex)->SetStyle(WindowStyle);
01652     }
01653
01654     return FOUNDATION_ERROR;
01655 }
```

**3.2.3.85 GLboolean WindowManager::SetWindowSwapInterval ( const char \* *WindowName*, GLint *a\_SyncSetting* ) [static]**

Sets window swap interval.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
<i>a_SyncSetting</i>	The synchronize setting.

Definition at line 1605 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SetSwapInterval\(\)](#).

```
01606 {
01607     if (DoesExist(WindowName))
01608     {
01609         return GetWindowByName(WindowName)->SetSwapInterval(a_SyncSetting);
01610     }
01611
01612     return FOUNDATION_ERROR;
01613 }
```

**3.2.3.86** GLboolean WindowManager::SetWindowSwapInterval ( GLuint *WindowIndex*, GLint *a\_SyncSetting* ) [static]

Sets window swap interval.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
<i>a_SyncSetting</i>	The synchronize setting.

Definition at line 1627 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SetSwapInterval\(\)](#).

```

01628 {
01629     if (DoesExist(WindowIndex))
01630     {
01631         return GetWindowByIndex(WindowIndex)->SetSwapInterval(a_SyncSetting)
01632     }
01633
01634     return FOUNDATION_ERROR;
01635 }
```

**3.2.3.87** GLboolean WindowManager::SetWindowTitleBar ( const char \* *WindowName*, const char \* *NewTitle* ) [static]

Sets window title bar.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
<i>NewTitle</i>	The new title bar text.

Definition at line 1428 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), [IsValidString\(\)](#), and [FWindow::SetTitleBar\(\)](#).

```

01429 {
01430     if (DoesExist(WindowName) && IsValidString(NewTitle))
01431     {
01432         return GetWindowByName(WindowName)->SetTitleBar(NewTitle);
01433     }
01434
01435     return FOUNDATION_ERROR;
01436 }
```

**3.2.3.88** `GLboolean WindowManager::SetWindowTitleBar ( GLuint WindowIndex, const char * NewTitle )` `[static]`

Sets window title bar.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
<i>NewTitle</i>	The new title bar text.

Definition at line 1450 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), [IsValidString\(\)](#), and [FWindow::SetTitleBar\(\)](#).

```
01451 {
01452     if (DoesExist (WindowIndex) && IsValidString (NewTitle))
01453     {
01454         return GetWindowByIndex (WindowIndex) ->SetTitleBar (NewTitle);
01455     }
01456
01457     return FOUNDATION_ERROR;
01458 }
```

**3.2.3.89** `void WindowManager::ShutDown ( )` `[static]`

Shuts down this object and frees any resources it is using.

shutdown and delete all windows in the manager

**Author**

Ziyad

**Date**

29/11/2014

Definition at line 324 of file [WindowManager.cpp](#).

References [GetInstance\(\)](#), [Instance](#), and [Windows](#).

Referenced by [main\(\)](#).

```
00325 {
00326     #if defined(CURRENT_OS_WINDOWS)
00327     for each(auto CurrentWindow in GetInstance()->Windows)
00328     {
00329         delete CurrentWindow;
00330     }
00331
00332     GetInstance()->Windows.clear();
00333
00334     #endif
00335
00336     #if defined(CURRENT_OS_LINUX)
00337     for (auto CurrentWindow : GetInstance()->Windows)
00338     {
00339         delete CurrentWindow;
00340     }
```

```

00341
00342    GetInstance()->Windows.clear();
00343
00344     XCloseDisplay(GetInstance()->m_Display);
00345 #endif
00346
00347     delete Instance;
00348 }

```

### 3.2.3.90 GLboolean WindowManager::WindowGetKey ( const char \* *WindowName*, GLuint *Key* ) [static]

get the state of the key relative to the window.

#### Author

Ziyad

#### Date

29/11/2014

#### Parameters

<i>WindowName</i>	Name of the window.
<i>Key</i>	The key.

#### Returns

The state of the key.

Definition at line 978 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetKeyState\(\)](#), and [GetWindowByName\(\)](#).

```

00979 {
00980     if (DoesExist (WindowName) )
00981     {
00982         return GetWindowByName (WindowName)->GetKeyState (Key) ;
00983     }
00984
00985     return FOUNDATION_ERROR;
00986 }

```

### 3.2.3.91 GLboolean WindowManager::WindowGetKey ( GLuint *WindowIndex*, GLuint *Key* ) [static]

get the state of the key relative to the window.

#### Author

Ziyad

#### Date

29/11/2014

**Parameters**

<i>WindowIndex</i>	Zero-based index of the window.
<i>Key</i>	The key.

**Returns**

The state of the key.

Definition at line 1002 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [FWindow::GetKeyState\(\)](#), and [GetWindowByIndex\(\)](#).

```

01003 {
01004     if (DoesExist (WindowIndex) )
01005     {
01006         return GetWindowByIndex (WindowIndex) ->GetKeyState (Key) ;
01007     }
01008
01009     return FOUNDATION_ERROR;
01010 }
```

### 3.2.3.92 GLboolean WindowManager::WindowSwapBuffers ( const char \* *WindowName* ) [static]

Swap DrawBuffers for that window.

**Author**

Ziyad

**Date**

29/11/2014

**Parameters**

<i>WindowName</i>	Name of the window.
-------------------	---------------------

Definition at line 1069 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByName\(\)](#), and [FWindow::SwapDrawBuffers\(\)](#).

```

01070 {
01071     if (DoesExist (WindowName) )
01072     {
01073         return GetWindowByName (WindowName) ->SwapDrawBuffers () ;
01074     }
01075
01076     return FOUNDATION_ERROR;
01077 }
```

### 3.2.3.93 GLboolean WindowManager::WindowSwapBuffers ( GLuint *WindowIndex* ) [static]

Swap DrawBuffers for that window.

**Author**

Ziyad

**Date**

29/11/2014



## Parameters

<i>WindowIndex</i>	Zero-based index of the window.
--------------------	---------------------------------

Definition at line 1090 of file [WindowManager.cpp](#).

References [DoesExist\(\)](#), [FOUNDATION\\_ERROR](#), [GetWindowByIndex\(\)](#), and [FWindow::SwapDrawBuffers\(\)](#).

```

01091 {
01092     if (DoesExist (WindowIndex) )
01093     {
01094         return GetWindowByIndex (WindowIndex) ->SwapDrawBuffers ();
01095     }
01096     return FOUNDATION_ERROR;
01097 }
01098 }
```

### 3.2.4 Member Data Documentation

#### 3.2.4.1 friend WindowManager::FWindow [private]

Definition at line 12 of file [WindowManager.h](#).

#### 3.2.4.2 GLboolean WindowManager::Initialized [private]

whether the window manager has been initialized

Definition at line 201 of file [WindowManager.h](#).

Referenced by [Initialize\(\)](#), and [IsInitialized\(\)](#).

#### 3.2.4.3 WindowManager \* WindowManager::Instance = 0 [static], [private]

The static reference to the [WindowManager](#)

Definition at line 196 of file [WindowManager.h](#).

Referenced by [GetInstance\(\)](#), and [ShutDown\(\)](#).

#### 3.2.4.4 GLuint WindowManager::ScreenMousePosition[2] [private]

the position of the mouse relative to screen coordinates

Definition at line 199 of file [WindowManager.h](#).

Referenced by [GetMousePositionInScreen\(\)](#), and [SetMousePositionInScreen\(\)](#).

#### 3.2.4.5 GLuint WindowManager::ScreenResolution[2] [private]

the resolution of the screen as an array

Definition at line 198 of file [WindowManager.h](#).

Referenced by [GetScreenResolution\(\)](#).

#### 3.2.4.6 std::list<FWindow\*> WindowManager::Windows [private]

The FWindows storage

Definition at line 195 of file [WindowManager.h](#).

Referenced by [AddWindow\(\)](#), [DoesExist\(\)](#), [GetNumWindows\(\)](#), [GetWindowByIndex\(\)](#), [GetWindowByName\(\)](#), [GetWindowsIsFullScreen\(\)](#), [GetWindowPosition\(\)](#), [SetWindowPosition\(\)](#), [ShutDown\(\)](#), and [~WindowManager\(\)](#).

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

- [WindowManager.h](#)
- [WindowManager.cpp](#)

# Chapter 4

## File Documentation

### 4.1 Example.cpp File Reference

```
#include <stdio.h>
#include "WindowManager.h"
```

#### Functions

- void [OnWindowKeyPressed](#) (GLuint KeySym, GLboolean KeyState)  
*Executes the window key pressed action.*
- int [main](#) ()  
*Main entry-point for this application.*

#### 4.1.1 Function Documentation

##### 4.1.1.1 int main ( )

Main entry-point for this application.

#### Author

Ziyad

#### Date

29/11/2014

#### Returns

Exit-code for the process - 0 for success, else an error code.

Definition at line 40 of file [Example.cpp](#).

References [WindowManager::AddWindow\(\)](#), [DECORATOR\\_MINIMIZEBUTTON](#), [WindowManager::EnableWindowDecorator\(\)](#), [WindowManager::GetNumWindows\(\)](#), [WindowManager::GetWindowByIndex\(\)](#), [WindowManager::GetWindowShouldClose\(\)](#), [WindowManager::Initialize\(\)](#), [FWindow::MakeCurrentContext\(\)](#), [OnWindowKeyPressed\(\)](#), [WindowManager::PollForEvents\(\)](#), [WindowManager::SetWindowOnKeyEvent\(\)](#), [WindowManager::SetWindowStyle\(\)](#), [WindowManager::ShutDown\(\)](#), [FWindow::SwapDrawBuffers\(\)](#), and [WINDOWSTYLE\\_BARE](#).

```

00041 {
00042     WindowManager::Initialize();
00043     WindowManager::AddWindow(new FWindow("Example")); //->AddWindow(new
FWindow("Example2"));
00044     WindowManager::SetWindowOnKeyEvent("Example", &
OnWindowKeyPressed);
00045     WindowManager::SetWindowStyle("Example",
WINDOWSTYLE_BARE);
00046     WindowManager::EnableWindowDecorator("Example",
DECORATOR_MINIMIZEBUTTON);
00047     while (!WindowManager::GetWindowShouldClose("Example"))
00048     {
00049         WindowManager::PollForEvents();
00050
00051         for (GLuint i = 0; i < WindowManager::GetNumWindows(); i++)
00052         {
00053             WindowManager::GetWindowByIndex(i)->
MakeCurrentContext();
00054             glClearColor(0.25f, 0.25f, 0.25f, 0.25f);
00055             glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00056             WindowManager::GetWindowByIndex(i)->
SwapDrawBuffers();
00057         }
00058     }
00059
00060     WindowManager::ShutDown();
00061     return 0;
00062 }

```

#### 4.1.1.2 void OnWindowKeyPressed ( GLuint KeySym, GLboolean KeyState )

Executes the window key pressed action.

Author

Ziyad

Date

29/11/2014

Parameters

<i>KeySym</i>	The key symbol.
<i>KeyState</i>	State of the key.

Definition at line 21 of file [Example.cpp](#).

References [KEYSTATE\\_DOWN](#).

Referenced by [main\(\)](#).

```

00022 {
00023     if(KeySym == ' ' && KeyState == KEYSTATE_DOWN)
00024     {
00025         printf("1234\n");
00026     }
00027 }

```

## 4.2 Example.cpp

```

00001 /*****
00006 #include <stdio.h>
00007 #include "WindowManager.h"
00008
00009 *****/
00021 void OnWindowKeyPressed(GLuint KeySym, GLboolean KeyState)
00022 {
00023     if(KeySym == ' ' && KeyState == KEYSTATE_DOWN)
00024     {
00025         printf("1234\n");

```

```

00026     }
00027 }
00028
00029 /*****
00040 int main()
00041 {
00042     WindowManager::Initialize();
00043     WindowManager::AddWindow(new FWindow("Example")); //->AddWindow(new
        FWindow("Example2"));
00044     WindowManager::SetWindowOnKeyEvent("Example", &
        OnWindowKeyPressed);
00045     WindowManager::SetWindowStyle("Example",
        WINDOWSTYLE_BARE);
00046     WindowManager::EnableWindowDecorator("Example",
        DECORATOR_MINIMIZEBUTTON);
00047     while (!WindowManager::GetWindowShouldClose("Example"))
00048     {
00049         WindowManager::PollForEvents();
00050
00051         for (GLuint i = 0; i < WindowManager::GetNumWindows(); i++)
00052         {
00053             WindowManager::GetWindowByIndex(i)->
        MakeCurrentContext();
00054             glClearColor(0.25f, 0.25f, 0.25f, 0.25f);
00055             glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00056             WindowManager::GetWindowByIndex(i)->
        SwapDrawBuffers();
00057         }
00058     }
00059
00060     WindowManager::ShutDown();
00061     return 0;
00062 }

```

## 4.3 Window.cpp File Reference

```

#include <limits.h>
#include "Window.h"
#include "WindowManager.h"

```

## 4.4 Window.cpp

```

00001 /*****
00007 #include <limits.h>
00008 #include "Window.h"
00009 #include "WindowManager.h"
00010
00011 #if defined(CURRENT_OS_LINUX)
00012 #include <cstring>
00013 #endif
00014
00015 /*****
00031 FWindow::FWindow(const char* WindowName,
00032     GLuint Width /* = 1280 */,
00033     GLuint Height /* = 720 */,
00034     GLuint ColourBits /* = 32 */,
00035     GLuint DepthBits /* = 8 */,
00036     GLuint StencilBits /* = 8 */) :
00037     Name(WindowName),
00038     ColourBits(ColourBits),
00039     DepthBits(DepthBits),
00040     StencilBits(StencilBits)
00041 {
00042     Resolution[0] = Width;
00043     Resolution[1] = Height;
00044     Position[0] = 0;
00045     Position[1] = 0;
00046     ShouldClose = GL_FALSE;
00047     EXTSwapControlSupported = GL_FALSE;
00048     SGISwapControlSupported = GL_FALSE;
00049     MESASwapControlSupported = GL_FALSE;
00050
00051     if (!IsValidString(WindowName))
00052     {
00053         PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
00054         exit(0);

```

```

00055     }
00056
00057     InitializeEvents();
00058
00059     CurrentState = WINDOWSTATE_NORMAL;
00060     ContextCreated = GL_FALSE;
00061     IsCurrentContext = GL_FALSE;
00062 }
00063
00064 /*****
00073 FWindow::~FWindow()
00074 {
00075     Shutdown();
00076 }
00077
00078 /*****
00089 GLboolean FWindow::Shutdown()
00090 {
00091     if(ContextCreated)
00092     {
00093
00094 #if defined (CURRENT_OS_WINDOWS)
00095     Windows_Shutdown();
00096 #endif
00097
00098 #if defined(CURRENT_OS_LINUX)
00099     Linux_Shutdown();
00100 #endif
00101     ContextCreated = GL_FALSE;
00102     return FOUNDATION_OKAY;
00103 }
00104
00105     PrintErrorMessage(ERROR_NOCONTEXT);
00106     return FOUNDATION_ERROR;
00107 }
00108
00109 /*****
00120 GLboolean FWindow::Initialize()
00121 {
00122 #if defined(CURRENT_OS_WINDOWS)
00123     return Windows_Initialize();
00124 #endif
00125
00126 #if defined(CURRENT_OS_LINUX)
00127     return Linux_Initialize();
00128 #endif
00129 }
00130
00131 /*****
00142 GLboolean FWindow::GetShouldClose()
00143 {
00144     return ShouldClose;
00145 }
00146
00147 /*****
00156 void FWindow::InitializeEvents()
00157 {
00158     KeyEvent = nullptr;
00159     MouseButtonEvent = nullptr;
00160     MouseWheelEvent = nullptr;
00161     DestroyedEvent = nullptr;
00162     MaximizedEvent = nullptr;
00163     MinimizedEvent = nullptr;
00164     // RestoredEvent = nullptr;
00165     MovedEvent = nullptr;
00166     MouseMoveEvent = nullptr;
00167 }
00168
00169 /*****
00182 GLboolean FWindow::GetKeyState(GLuint Key)
00183 {
00184     return Keys[Key];
00185 }
00186
00187 /*****
00198 GLboolean FWindow::InitializeGL()
00199 {
00200 #if defined(CURRENT_OS_WINDOWS)
00201     return Windows_InitializeGL();
00202 #endif
00203
00204 #if defined(CURRENT_OS_LINUX)
00205     return Linux_InitializeGL();
00206 #endif
00207 }
00208
00209 /*****

```

```

00220 GLboolean FWindow::SwapDrawBuffers()
00221 {
00222     if (ContextCreated)
00223     {
00224         #if defined(CURRENT_OS_WINDOWS)
00225             SwapBuffers(DeviceContextHandle);
00226         #endif
00227         #if defined(CURRENT_OS_LINUX)
00228             glXSwapBuffers(WindowManager::GetDisplay(), WindowHandle);
00229         #endif
00230         return FOUNDATION_OKAY;
00231     }
00232     PrintErrorMessage(ERROR_NOCONTEXT);
00233     return FOUNDATION_ERROR;
00234 }
00235
00236 /*****
00252 GLboolean FWindow::SetSwapInterval(GLint SwapSetting)
00253 {
00254     if (ContextCreated)
00255     {
00256         CurrentSwapInterval = SwapSetting;
00257         #if defined(CURRENT_OS_WINDOWS)
00258             Windows_VerticalSync(SwapSetting);
00259         #endif
00260         #if defined(CURRENT_OS_LINUX)
00261             Linux_VerticalSync(SwapSetting);
00262         #endif
00263         return FOUNDATION_OKAY;
00264     }
00265     PrintErrorMessage(ERROR_NOCONTEXT);
00266     return FOUNDATION_ERROR;
00267 }
00268
00269 /*****
00283 GLuint FWindow::GetCurrentState()
00284 {
00285     return CurrentState;
00286 }
00287
00288 /*****
00301 GLboolean FWindow::SetCurrentState(GLuint NewState)
00302 {
00303     if (ContextCreated)
00304     {
00305         Restore();
00306         switch (NewState)
00307         {
00308             case WINDOWSTATE_MAXIMIZED:
00309             {
00310                 Maximize(GL_TRUE);
00311                 break;
00312             }
00313             case WINDOWSTATE_MINIMIZED:
00314             {
00315                 Minimize(GL_TRUE);
00316                 break;
00317             }
00318             case WINDOWSTATE_FULLSCREEN:
00319             {
00320                 FullScreen(GL_FALSE);
00321                 break;
00322             }
00323             default:
00324             {
00325                 break;
00326             }
00327         }
00328     }
00329     PrintErrorMessage(ERROR_NOCONTEXT);
00330     return FOUNDATION_ERROR;
00331 }
00332
00333 /*****
00354 GLboolean FWindow::GetIsFullScreen()

```

```

00355 {
00356     if (ContextCreated)
00357     {
00358         return (CurrentState == WINDOWSTATE_FULLSCREEN);
00359     }
00360
00361     PrintErrorMessage(ERROR_NOCONTEXT);
00362     return FOUNDATION_ERROR;
00363 }
00364
00365 /*****
00378 GLboolean FWindow::FullScreen(GLboolean ShouldBeFullscreen)
00379 {
00380     if (ContextCreated)
00381     {
00382         if (ShouldBeFullscreen)
00383         {
00384             CurrentState = WINDOWSTATE_FULLSCREEN;
00385         }
00386         else
00387         {
00388             CurrentState = WINDOWSTATE_NORMAL;
00389         }
00390     }
00391
00392     #if defined(CURRENT_OS_LINUX)
00393         Linux_FullScreen(ShouldBeFullscreen);
00394     #endif
00395
00396     #if defined(CURRENT_OS_WINDOWS)
00397         Windows_FullScreen();
00398     #endif
00399
00400     return FOUNDATION_OKAY;
00401 }
00402
00403     PrintErrorMessage(ERROR_NOCONTEXT);
00404     return FOUNDATION_OKAY;
00405 }
00406
00407 /*****
00418 GLboolean FWindow::GetIsMinimized()
00419 {
00420     return (CurrentState == WINDOWSTATE_MINIMIZED);
00421 }
00422
00423 /*****
00436 GLboolean FWindow::Minimize(GLboolean NewState)
00437 {
00438     if (ContextCreated)
00439     {
00440         if (NewState)
00441         {
00442             CurrentState = WINDOWSTATE_MINIMIZED;
00443         }
00444         else
00445         {
00446             CurrentState = WINDOWSTATE_NORMAL;
00447         }
00448     }
00449
00450     #if defined(CURRENT_OS_WINDOWS)
00451         Windows_Minimize();
00452     #endif
00453
00454     #if defined(CURRENT_OS_LINUX)
00455         Linux_Minimize(NewState);
00456     #endif
00457
00458     return FOUNDATION_OKAY;
00459 }
00460
00461     return FOUNDATION_ERROR;
00462 }
00463
00464 /*****
00475 GLboolean FWindow::GetIsMaximized()
00476 {
00477     return (CurrentState == WINDOWSTATE_MAXIMIZED);
00478 }
00479
00480 /*****
00493 GLboolean FWindow::Maximize(GLboolean NewState)
00494 {
00495     if (ContextCreated)
00496     {
00497         if (NewState)

```



```

00498     {
00499         CurrentState = WINDOWSTATE_MAXIMIZED;
00500     }
00501
00502     else
00503     {
00504         CurrentState = WINDOWSTATE_NORMAL;
00505     }
00506
00507 #if defined(CURRENT_OS_WINDOWS)
00508     Windows_Maximize();
00509 #endif
00510
00511 #if defined(CURRENT_OS_LINUX)
00512     Linux_Maximize(NewState);
00513 #endif
00514     return FOUNDATION_OKAY;
00515 }
00516 PrintErrorMessage(ERROR_NOCONTEXT);
00517 return FOUNDATION_ERROR;
00518 }
00519
00520 /*****
00531 GLboolean FWindow::Restore()
00532 {
00533     if (ContextCreated)
00534     {
00535         switch (CurrentState)
00536         {
00537             case WINDOWSTATE_MAXIMIZED:
00538             {
00539                 Maximize(GL_FALSE);
00540                 break;
00541             }
00542
00543             case WINDOWSTATE_FULLSCREEN:
00544             {
00545                 FullScreen(GL_FALSE);
00546                 break;
00547             }
00548         }
00549
00550         CurrentState = WINDOWSTATE_NORMAL;
00551 #if defined(CURRENT_OS_WINDOWS)
00552         Windows_Restore();
00553 #endif
00554 #if defined(CURRENT_OS_LINUX)
00555         Linux_Restore();
00556 #endif
00557
00558         return FOUNDATION_OKAY;
00559     }
00560
00561     PrintErrorMessage(ERROR_NOCONTEXT);
00562     return FOUNDATION_ERROR;
00563 }
00564
00565 /*****
00580 GLboolean FWindow::GetResolution(GLuint& Width, GLuint& Height)
00581 {
00582     if (ContextCreated)
00583     {
00584         Width = Resolution[0];
00585         Height = Resolution[1];
00586         return FOUNDATION_OKAY;
00587     }
00588
00589     PrintErrorMessage(ERROR_NOCONTEXT);
00590     return FOUNDATION_ERROR;
00591 }
00592
00593 /*****
00605 GLuint* FWindow::GetResolution()
00606 {
00607     return Resolution;
00608 }
00609
00610 /*****
00622 GLboolean FWindow::SetResolution(GLuint Width, GLuint Height)
00623 {
00624     if (ContextCreated)
00625     {
00626         if (Width > 0 && Height > 0)
00627         {
00628             Resolution[0] = Width;
00629             Resolution[1] = Height;

```

```

00630
00631 #if defined(CURRENT_OS_WINDOWS)
00632     Windows_SetResolution(Resolution[0], Resolution[1]);
00633 #endif
00634
00635 #if defined(CURRENT_OS_LINUX)
00636     Linux_SetResolution(Width, Height);
00637 #endif
00638
00639     glViewport(0, 0, Resolution[0], Resolution[1]);
00640
00641     return FOUNDATION_OKAY;
00642 }
00643
00644 else
00645 {
00646     PrintErrorMessage(ERROR_INVALIDRESOLUTION);
00647     return FOUNDATION_ERROR;
00648 }
00649 }
00651 PrintErrorMessage(ERROR_NOCONTEXT);
00652 return FOUNDATION_ERROR;
00653 }
00654
00655 /*****
00667 GLboolean FWindow::GetMousePosition(GLuint& X, GLuint& Y)
00668 {
00669     if (ContextCreated)
00670     {
00671         X = MousePosition[0];
00672         Y = MousePosition[1];
00673         return FOUNDATION_OKAY;
00674     }
00675
00676     PrintErrorMessage(ERROR_NOCONTEXT);
00677     return FOUNDATION_ERROR;
00678 }
00679
00680 /*****
00692 GLuint* FWindow::GetMousePosition()
00693 {
00694     if (ContextCreated)
00695     {
00696         return MousePosition;
00697     }
00698
00699     PrintErrorMessage(ERROR_NOCONTEXT);
00700     return nullptr;
00701 }
00702
00703 /*****
00715 GLboolean FWindow::SetMousePosition(GLuint X, GLuint Y)
00716 {
00717     if (ContextCreated)
00718     {
00719         MousePosition[0] = X;
00720         MousePosition[1] = Y;
00721     #if defined(CURRENT_OS_WINDOWS)
00722         Windows_SetMousePosition(X, Y);
00723     #endif
00724
00725     #if defined(CURRENT_OS_LINUX)
00726         Linux_SetMousePosition(X, Y);
00727     #endif
00728
00729     return FOUNDATION_OKAY;
00730 }
00731
00732     PrintErrorMessage(ERROR_NOCONTEXT);
00733     return FOUNDATION_ERROR;
00734 }
00735
00736 /*****
00748 GLboolean FWindow::GetPosition(GLuint& X, GLuint& Y)
00749 {
00750     if (ContextCreated)
00751     {
00752         X = Position[0];
00753         Y = Position[1];
00754
00755         return FOUNDATION_OKAY;
00756     }
00757
00758     PrintErrorMessage(ERROR_NOCONTEXT);
00759     return FOUNDATION_ERROR;
00760 }
00761

```

```

00762 /*****
00775 GLuint* FWindow::GetPosition()
00776 {
00777     return Position;
00778 }
00779
00780 /*****
00792 GLboolean FWindow::SetPosition(GLuint X, GLuint Y)
00793 {
00794     if (ContextCreated)
00795     {
00796         Position[0] = X;
00797         Position[1] = Y;
00798 #if defined(CURRENT_OS_WINDOWS)
00799     Windows_SetPosition(Position[0], Position[1]);
00800 #endif
00801
00802 #if defined(CURRENT_OS_LINUX)
00803     Linux_SetPosition(X, Y);
00804 #endif
00805     }
00806
00807     PrintErrorMessage(ERROR_NOCONTEXT);
00808     return FOUNDATION_ERROR;
00809 }
00810
00811 /*****
00822 const char* FWindow::GetWindowName()
00823 {
00824     if (ContextCreated)
00825     {
00826         return Name;
00827     }
00828
00829     PrintErrorMessage(ERROR_NOCONTEXT);
00830     return nullptr;
00831 }
00832
00833 /*****
00844 GLboolean FWindow::SetTitleBar(const char* NewTitle)
00845 {
00846     if (ContextCreated)
00847     {
00848         if(NewTitle != nullptr)
00849         {
00850 #if defined(CURRENT_OS_LINUX)
00851             Linux_SetTitleBar(NewTitle);
00852 #endif
00853
00854 #if defined(CURRENT_OS_WINDOWS)
00855             Windows_SetTitleBar(NewTitle);
00856 #endif
00857             return FOUNDATION_OKAY;
00858         }
00859         else
00860         {
00861             PrintErrorMessage(ERROR_INVALIDTITLEBAR);
00862             return FOUNDATION_ERROR;
00863         }
00864     }
00865
00866     PrintErrorMessage(ERROR_NOCONTEXT);
00867     return FOUNDATION_ERROR;
00868 }
00869
00870
00871 GLboolean FWindow::SetIcon(const char* Icon, GLuint Width, GLuint Height)
00872 {
00873     if (ContextCreated)
00874     {
00875 #if defined(CURRENT_OS_WINDOWS)
00876         Windows_SetIcon(Icon, Width, Height);
00877 #endif
00878
00879 #if defined(CURRENT_OS_LINUX)
00880         Linux_SetIcon(Icon, Width, Height);
00881 #endif
00882
00883         return FOUNDATION_OKAY;
00884     }
00885
00886     return FOUNDATION_ERROR;
00887 }
00888
00889 GLboolean FWindow::SetStyle(GLuint WindowType)
00890 {
00891     if (ContextCreated)

```

```

00892     {
00893     #if defined(CURRENT_OS_WINDOWS)
00894         Windows_SetStyle(WindowType);
00895     #endif
00896
00897     #if defined(CURRENT_OS_LINUX)
00898         Linux_SetStyle(WindowType);
00899     #endif
00900
00901         PrintErrorMessage(ERROR_NOCONTEXT);
00902         return FOUNDATION_OKAY;
00903     }
00904
00905     return FOUNDATION_ERROR;
00906 }
00907
00908 /*****
00920 GLboolean FWindow::MakeCurrentContext ()
00921 {
00922     if(ContextCreated)
00923     {
00924         IsCurrentContext = true;
00925     #if defined(CURRENT_OS_WINDOWS)
00926         wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);
00927     #endif
00928
00929     #if defined(CURRENT_OS_LINUX)
00930         glXMakeCurrent(WindowManager::GetDisplay(), WindowHandle, Context);
00931     #endif
00932         return FOUNDATION_OKAY;
00933     }
00934
00935     PrintErrorMessage(ERROR_NOCONTEXT);
00936     return FOUNDATION_ERROR;
00937 }
00938
00939 /*****
00950 GLboolean FWindow::GetIsCurrentContext ()
00951 {
00952     if(ContextCreated)
00953     {
00954         return IsCurrentContext;
00955     }
00956     PrintErrorMessage(ERROR_NOCONTEXT);
00957     return GL_FALSE;
00958 }
00959
00960 /*****
00971 GLboolean FWindow::GetContextHasBeenCreated()
00972 {
00973     return ContextCreated;
00974 }
00975
00976 /*****
00985 void FWindow::InitGLExtensions()
00986 {
00987     #if defined(CURRENT_OS_WINDOWS)
00988         Windows_InitGLExtensions();
00989     #endif
00990
00991     #if defined(CURRENT_OS_LINUX)
00992         Linux_InitGLExtensions();
00993     #endif
00994 }
00995
00996 /*****
01007 GLboolean FWindow::PrintOpenGLVersion ()
01008 {
01009     if(ContextCreated)
01010     {
01011         printf("%s\n", glGetString(GL_VERSION));
01012         return FOUNDATION_OKAY;
01013     }
01014
01015     PrintErrorMessage(ERROR_NOCONTEXT);
01016     return FOUNDATION_ERROR;
01017 }
01018
01019 const char* FWindow::GetOpenGLVersion ()
01020 {
01021     if(ContextCreated)
01022     {
01023         return (const char*)glGetString(GL_VERSION);
01024     }
01025     PrintErrorMessage(ERROR_NOCONTEXT);
01026     return nullptr;
01027 }

```

```

01028
01029 GLboolean FWindow::PrintOpenGLExtensions()
01030 {
01031     if(ContextCreated)
01032     {
01033         printf("%s \n", (const char*)glGetString(GL_EXTENSIONS));
01034         return FOUNDATION_OKAY;
01035     }
01036
01037     PrintErrorMessage(ERROR_NOCONTEXT);
01038     return FOUNDATION_ERROR;
01039 }
01040
01041 /*****
01052 const char* FWindow::GetOpenGLExtensions()
01053 {
01054     if(ContextCreated)
01055     {
01056         return (const char*)glGetString(GL_EXTENSIONS);
01057     }
01058     else
01059     {
01060         PrintErrorMessage(ERROR_NOCONTEXT);
01061         return nullptr;
01062     }
01063 }
01064 */
01065
01066 /*****
01077 GLboolean FWindow::GetInFocus()
01078 {
01079     return InFocus;
01080 }
01081 */
01082 /*****
01093 GLboolean FWindow::Focus(GLboolean ShouldBeInFocus)
01094 {
01095     if (ContextCreated)
01096     {
01097         InFocus = ShouldBeInFocus;
01098
01099 #if defined(CURRENT_OS_LINUX)
01100         Linux_Focus(ShouldBeInFocus);
01101 #endif
01102
01103 #if defined(CURRENT_OS_WINDOWS)
01104         Windows_Focus();
01105 #endif
01106
01107         return FOUNDATION_OKAY;
01108     }
01109
01110     PrintErrorMessage(ERROR_NOCONTEXT);
01111     return FOUNDATION_ERROR;
01112 }
01113 */
01114 /*****
01127 GLboolean FWindow::SetOnKeyEvent(OnKeyEvent OnKey)
01128 {
01129     if (IsValidKeyEvent (OnKey))
01130     {
01131         KeyEvent = OnKey;
01132         return FOUNDATION_OKAY;
01133     }
01134
01135     return FOUNDATION_ERROR;
01136 }
01137 */
01138 /*****
01151 GLboolean FWindow::SetOnMouseButtonEvent (
01152     OnMouseButtonEvent OnMouseButtonEvent)
01153 {
01154     //we don't really need to check if the context has been created
01155     if(IsValidKeyEvent (OnMouseButtonEvent))
01156     {
01157         MouseButtonEvent = OnMouseButtonEvent;
01158         return FOUNDATION_OKAY;
01159     }
01160
01161     PrintErrorMessage(ERROR_INVALIDEVENT);
01162     return FOUNDATION_ERROR;
01163 }
01164 */
01165 /*****
01177 GLboolean FWindow::SetOnMouseWheelEvent (
01178     OnMouseWheelEvent OnMouseWheel)
01179 {

```

```

01179     if (IsValidMouseEvent (OnMouseWheel))
01180     {
01181         MouseWheelEvent = OnMouseWheel;
01182         return FOUNDATION_OKAY;
01183     }
01184
01185     PrintErrorMessage(ERROR_INVALIDEVENT);
01186     return FOUNDATION_ERROR;
01187 }
01188
01189 /*****
01202 GLboolean FWindow::SetOnDestroyed(OnDestroyedEvent OnDestroyed)
01203 {
01204     if (IsValidDestroyedEvent (OnDestroyed))
01205     {
01206         DestroyedEvent = OnDestroyed;
01207         return FOUNDATION_OKAY;
01208     }
01209
01210     PrintErrorMessage(ERROR_INVALIDEVENT);
01211     return FOUNDATION_ERROR;
01212 }
01213
01214 /*****
01227 GLboolean FWindow::SetOnMaximized(OnMaximizedEvent OnMaximized)
01228 {
01229     if (IsValidDestroyedEvent (OnMaximized))
01230     {
01231         MaximizedEvent = OnMaximized;
01232         return FOUNDATION_OKAY;
01233     }
01234     PrintErrorMessage(ERROR_INVALIDEVENT);
01235     return FOUNDATION_ERROR;
01236 }
01237
01238 /*****
01251 GLboolean FWindow::SetOnMinimized(OnMinimizedEvent OnMinimized)
01252 {
01253     if (IsValidDestroyedEvent (OnMinimized))
01254     {
01255         MinimizedEvent = OnMinimized;
01256         return FOUNDATION_OKAY;
01257     }
01258
01259     PrintErrorMessage(ERROR_INVALIDEVENT);
01260     return FOUNDATION_ERROR;
01261 }
01262
01263 /*void FWindow::SetOnRestored(OnRestoredEvent OnRestored)
01264 {
01265     if (IsValid(OnRestored))
01266     {
01267         RestoredEvent = OnRestored;
01268     }
01269 }*/
01270
01271 /*****
01284 GLboolean FWindow::SetOnFocus(OnFocusEvent OnFocus)
01285 {
01286     if (IsValidFocusEvent (OnFocus))
01287     {
01288         FocusEvent = OnFocus;
01289         return FOUNDATION_OKAY;
01290     }
01291
01292     PrintErrorMessage(ERROR_INVALIDEVENT);
01293     return FOUNDATION_ERROR;
01294 }
01295
01296 /*****
01307 GLboolean FWindow::SetOnMoved(OnMovedEvent OnMoved)
01308 {
01309     if (IsValidMovedEvent (OnMoved))
01310     {
01311         MovedEvent = OnMoved;
01312         return FOUNDATION_OKAY;
01313     }
01314     PrintErrorMessage(ERROR_INVALIDEVENT);
01315     return FOUNDATION_ERROR;
01316 }
01317
01318 /*****
01329 GLboolean FWindow::SetOnResize(OnResizeEvent OnResize)
01330 {
01331     if (IsValidMovedEvent (OnResize))
01332     {
01333         ResizeEvent = OnResize;

```

```

01334         return FOUNDATION_OKAY;
01335     }
01336
01337     PrintErrorMessage(ERROR_INVALIDEVENT);
01338     return FOUNDATION_ERROR;
01339 }
01340
01341 /*****
01352 GLboolean FWindow::SetOnMouseMove(OnMouseMoveEvent OnMouseMove)
01353 {
01354     if(IsValidMouseMoveEvent(OnMouseMove))
01355     {
01356         MouseMoveEvent = OnMouseMove;
01357         return FOUNDATION_OKAY;
01358     }
01359
01360     PrintErrorMessage(ERROR_INVALIDEVENT);
01361     return FOUNDATION_ERROR;
01362 }
01363
01364 GLboolean FWindow::EnableDecorator(GLbitfield Decorator)
01365 {
01366     if (ContextCreated)
01367     {
01368 #if defined(CURRENT_OS_WINDOWS)
01369         Windows_EnableDecorator(Decorator);
01370 #endif
01371
01372 #if defined(CURRENT_OS_LINUX)
01373         Linux_EnableDecorator(Decorator);
01374 #endif
01375
01376         return FOUNDATION_OKAY;
01377     }
01378
01379     PrintErrorMessage(ERROR_NOCONTEXT);
01380     return FOUNDATION_ERROR;
01381 }
01382 GLboolean FWindow::DisableDecorator(GLbitfield Decorator)
01383 {
01384     if (ContextCreated)
01385     {
01386 #if defined(CURRENT_OS_WINDOWS)
01387         Windows_DisableDecorator(Decorator);
01388 #endif
01389
01390 #if defined(CURRENT_OS_LINUX)
01391         Linux_DisableDecorator(Decorator);
01392 #endif
01393
01394         return FOUNDATION_OKAY;
01395     }
01396
01397     PrintErrorMessage(ERROR_NOCONTEXT);
01398     return FOUNDATION_ERROR;
01399 }

```

## 4.5 Window.h File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <fcntl.h>
#include "WindowAPI_Defs.h"

```

### Classes

- class [FWindow](#)

## 4.6 Window.h

```

00001 #ifndef WINDOW_H
00002 #define WINDOW_H

```

```

00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <string>
00007 #include <fcntl.h>
00008 #include "WindowAPI_Defs.h"
00009
00010 #if defined(CURRENT_OS_WINDOWS)
00011 #include <io.h>
00012
00013 LRESULT CALLBACK FWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
00014 //this automatically loads the OpenGL library if you are using Visual studio
00015 #pragma comment (lib, "opengl32.lib")
00016 //this makes sure that the entry point of your program is main(). not Winmain
00017 #pragma comment(linker, "/subsystem:windows /ENTRY:mainCRTStartup")
00018 #endif
00019
00020 class WindowManager; // just a forward declaration for the window manager
00021
00022 class FWindow
00023 {
00024 public:
00025     //window constructor
00026     FWindow(const char* WindowName, GLuint Width = 1280, GLuint Height = 720, GLuint
ColourBits = 8,
00027             GLuint DepthBits = 24, GLuint StencilBits = 8);
00028
00029     //window deconstruction
00030     ~FWindow();
00031
00032     //Initializes the window depending on OS
00033     GLboolean Initialize();
00034
00035     //shut down respective OpenGL context
00036     GLboolean Shutdown();
00037
00038     //return the size/resolution of the window
00039     GLboolean GetResolution(GLuint& Width, GLuint& Height);
00040     //return the size/resolution of the window
00041     GLuint* GetResolution();
00042     //set the size/Resolution of the window
00043     GLboolean SetResolution(GLuint Width, GLuint Height);
00044
00045     //return the position of the mouse cursor relative to the window co-ordinates
00046     GLboolean GetMousePosition(GLuint& X, GLuint& Y);
00047     //return the Position of the mouse cursor relative to the window co-ordinates
00048     GLuint* GetMousePosition();
00049     //set the position of the mouse cursor relative the the window co-ordinates
00050     GLboolean SetMousePosition(GLuint X, GLuint Y);
00051
00052     //return the Position of the window relative to the screen co-ordinates
00053     GLboolean GetPosition(GLuint& X, GLuint& Y);
00054     //return the Position of the window relative to the screen co-ordinates
00055     GLuint* GetPosition();
00056     //Set the Position of the window relative to the screen co-ordinates
00057     GLboolean SetPosition(GLuint X, GLuint Y);
00058
00059     //return the current state of the window
00060     GLuint GetCurrentState();
00061     //set the current state of the window
00062     GLboolean SetCurrentState(GLuint NewState);
00063
00064     //get the state of a key(Down/Up) by index
00065     GLboolean GetKeyState(GLuint Key);
00066
00067     //whether or not the window should be closing
00068     GLboolean GetShouldClose();
00069
00070     //make the window swap draw buffers
00071     GLboolean SwapDrawBuffers();
00072
00073     //toggle full screen mode depending on NewState. (true = Full screen, false = normal)
00074     GLboolean FullScreen(GLboolean NewState);
00075     //return if the window is in full screen mode
00076     GLboolean GetIsFullScreen();
00077
00078     //toggle minimization depending on NewState. (true = minimized, false = normal)
00079     GLboolean Minimize(GLboolean NewState);
00080     //return if the window is Minimized
00081     GLboolean GetIsMinimized();
00082
00083     // set and get for maximizing a window
00084     GLboolean Maximize(GLboolean NewState);
00085     GLboolean GetIsMaximized();
00086
00087     //restore the window to its natural state
00088     GLboolean Restore();

```



```

00089
00090 //creates on OpenGL Context
00091 GLboolean InitializeGL();
00092
00093 //get and set for window name
00094 const char* GetWindowName();
00095 GLboolean SetTitleBar(const char* NewText);
00096
00097 //set the style for the window
00098 GLboolean SetStyle(GLuint WindowType);
00099
00100 //set the window icon
00101 GLboolean SetIcon(const char* Icon, GLuint Width, GLuint Height);
00102
00103 //make the window the current OpenGL context to be drawn
00104 GLboolean MakeCurrentContext();
00105
00106 //returns Whether the current window is the current OpenGL context to be drawn
00107 GLboolean GetIsCurrentContext();
00108
00109 //returns whether the OpenGL context for this window has been created
00110 GLboolean GetContextHasBeenCreated();
00111
00112 //whether the window is in focus
00113 GLboolean GetInFocus();
00114 GLboolean Focus(GLboolean NewState);
00115
00116 //enable vertical sync if supported
00117 //a swap setting of -1 turns on adaptive V-sync on supported systems
00118 GLboolean SetSwapInterval(GLint SwapSetting);
00119
00120 //set the on key event callback for this window
00121 GLboolean SetOnKeyEvent(OnKeyEvent OnKey);
00122 //set the on mouse button event callback for this window
00123 GLboolean SetOnMouseButtonEvent(OnMouseButtonEvent OnMouseButton
);
00124 //set the on mouse wheel event callback for this window
00125 GLboolean SetOnMouseWheelEvent(OnMouseWheelEvent OnMouseWheel);
00126 //set the window on destroyed event callback for this window
00127 GLboolean SetOnDestroyed(OnDestroyedEvent OnDestroyed);
00128 //set the window on maximizes event callback for this window
00129 GLboolean SetOnMaximized(OnMaximizedEvent OnMaximized);
00130 //set the window on minimized event callback for this window
00131 GLboolean SetOnMinimized(OnMinimizedEvent OnMinimized);
00132 //set the window on restored event callback for this window
00133 //void SetOnRestored(OnRestoredEvent OnRestored);
00134 //set the window on focus event callback for this window
00135 GLboolean SetOnFocus(OnFocusEvent OnFocus);
00136 //set the window on moved event callback for this window
00137 GLboolean SetOnMoved(OnMovedEvent OnMoved);
00138 //set the window on resize event callback for this window
00139 GLboolean SetOnResize(OnResizeEvent OnResize);
00140 //set the window on Mouse move callback event for this window
00141 GLboolean SetOnMouseMove(OnMouseMoveEvent OnMouseMove);
00142
00143 //print the current OpenGL version
00144 GLboolean PrintOpenGLVersion();
00145 //return the current OpenGL version as a string
00146 const char* GetOpenGLVersion();
00147 //print all supported extensions
00148 GLboolean PrintOpenGLExtensions();
00149 //return all the supported extensions
00150 const char* GetOpenGLExtensions();
00151
00152 //enable window decorator
00153 GLboolean EnableDecorator(GLbitfield Decorator);
00154 //disable window decorator
00155 GLboolean DisableDecorator(GLbitfield Decorator);
00156
00157 friend class WindowManager; // lets window use private variables of WindowManager
00158
00159 private:
00160
00161 const char* Name;
00162 GLuint ID;
00163 GLint ColourBits;
00164 GLint DepthBits;
00165 GLint StencilBits;
00166 GLboolean Keys[KEY_LAST];
00167 GLboolean MouseButton[MOUSE_LAST];
00168 GLuint Resolution[2];
00169 GLuint Position[2];
00170 GLuint MousePosition[2];
00171 GLboolean ShouldClose;
00172 GLboolean InFocus;
00173 GLboolean Initialized;
00174 GLboolean ContextCreated;

```

```

00175     GLboolean IsCurrentContext;
00176     GLuint CurrentState;
00177     GLuint CurrentSwapInterval;
00178     GLbitfield CurrentWindowStyle;
00180 //set all the Events to null
00181     void InitializeEvents();
00182     //Initializes OpenGL extensions
00183     void InitGLExtensions();
00184
00185     OnKeyEvent KeyEvent;
00186     OnMouseButtonEvent MouseButtonEvent;
00187     OnMouseWheelEvent MouseWheelEvent;
00188     OnDestroyedEvent DestroyedEvent;
00189     OnMaximizedEvent MaximizedEvent;
00190     OnMinimizedEvent MinimizedEvent;
00191     //OnRestoredEvent RestoredEvent; /**< this is the callback to be used when the window has been restored
    in a non-programmatic fashion*/
00192     OnFocusEvent FocusEvent;
00193     OnMovedEvent MovedEvent;
00194     OnResizeEvent ResizeEvent;
00195     OnMouseMoveEvent MouseMoveEvent;
00197     GLboolean EXTSwapControlSupported;
00198     GLboolean SGISwapControlSupported;
00199     GLboolean MESASwapControlSupported;
00201     //this section is for the windows side of the Window API
00202 #if defined(CURRENT_OS_WINDOWS)
00203
00204 private:
00205
00206     //tells windows to create a generic window. need to implement window styles sometime later
00207     GLboolean Windows_Initialize(UINT a_Style = CS_OWNDC | CS_HREDRAW | CS_VREDRAW,
00208         GLint a_ClearScreenExtra = 0, GLint FWindowExtra = 0,
00209         HINSTANCE a_Instance = GetModuleHandle(0),
00210         HICON a_Icon = LoadIcon(0, IDI_APPLICATION),
00211         HCURSOR a_Cursor = LoadCursor(0, IDC_ARROW),
00212         HBRUSH a_Brush = (HBRUSH)BLACK_BRUSH);
00213     //uses the Win32 system to set the resolution/size of the window
00214     void Windows_SetResolution(GLuint Width, GLuint Height);
00215     //uses the win32 system to set the position of the window relative to the screen
00216     void Windows_SetPosition(GLuint X, GLuint Y);
00217     //uses the win32 system to set the position of the mouse cursor relative to the window
00218     void Windows_SetMousePosition(GLuint X, GLuint Y);
00219     //uses the win32 system to have the window completely fill the screen and be
00220     drawn above the toolbar. changing the screen resolution to match has been disabled
00221     due to event handling issues*/
00222     void Windows_FullScreen();
00223     //uses the win32 system to minimize/hide the window. minimized windows don't receive events
00224     void Windows_Minimize();
00225     //uses the win32 system to maximize the window.
00226     void Windows_Maximize();
00227     //uses the win32 system to restore the window
00228     void Windows_Restore();
00229     //uses the win32 system to set the Title Bar text
00230     void Windows_SetTitleBar(const char* NewTitle);
00231     //uses the Win32 system to set the window icon
00232     void Windows_SetIcon(const char* Icon, GLuint Width, GLuint Height);
00233     //uses the win32 system to put the window into event focus
00234     void Windows_Focus();
00235     //initialize OpenGL for this window
00236     GLboolean Windows_InitializeGL();
00237     //cleanly shutdown this window(window would still need to be deleted of course)
00238     void Windows_Shutdown();
00239     //turns of vertical sync using the EXT extension
00240     void Windows_VerticalSync(GLint EnableSync);
00241     //enables given window decoration via Win32
00242     void Windows_EnableDecorator(GLbitfield Decorator);
00243     //disables given window decoration via Win32
00244     void Windows_DisableDecorator(GLbitfield Decorator);
00245
00246     void Windows_SetStyle(GLuint WindowType);
00247     //get the handle of the window. to be used internally only
00248     HWND GetWindowHandle();
00249
00250     //initialize the pixel format
00251     void InitializePixelFormat();
00252
00253     //initialize NEEDED OpenGL extensions for the windows platform
00254     void Windows_InitGLExtensions();
00255
00256     HDC DeviceContextHandle;
00257     HGLRC GLRenderingContextHandle;
00258     HPALETTE PaletteHandle;
00259     PIXELFORMATDESCRIPTOR PixelFormatDescriptor;
00261     WNDCLASS WindowClass;
00262     HWND WindowHandle;
00263     HINSTANCE InstanceHandle;
00265     PFNWGLSWAPINTERVALEXTPROC SwapIntervalEXT;

```

```

00266     PFNWGLSWAPBUFFERSMSCOMLPROC SwapIntervalMSCOM; // what the holy fuck is MSCOM?
00267     PFNWGLGETEXTENSIONSSTRINGEXTPROC GetExtensionsStringEXT;
00268 #endif
00269
00270 #if defined(CURRENT_OS_LINUX)
00271     //uses the X11 system to initialize the window
00272     GLboolean Linux_Initialize();
00273     //uses the X11 system to set the size/resolution of the window
00274     void Linux_SetResolution(GLuint Width, GLuint Height);
00275     //uses the X11 system to set the window position relative to screen co-ordinates
00276     void Linux_SetPosition(GLuint X, GLuint Y);
00277     //uses the X11 system to set the mouse position relative to the window co-ordinates
00278     void Linux_SetMousePosition(GLuint X, GLuint Y);
00279     //uses the X11 system to toggle full screen mode
00280     void Linux_FullScreen(GLboolean NewState);
00281     //uses the X11 system to toggle minimization
00282     void Linux_Minimize(GLboolean NewState);
00283     //uses the X11 system to toggle maximization
00284     void Linux_Maximize(GLboolean NewState);
00285     //uses the X11 system to restore the window
00286     void Linux_Restore();
00287     //uses the X11 system to toggle the window's event focus state
00288     void Linux_Focus(GLboolean NewState);
00289     //uses the X11 system to set the title bar of the window
00290     void Linux_SetTitleBar(const char* NewName);
00291     //uses the X11 system to set the icon of the window
00292     void Linux_SetIcon(const char* Icon, GLuint Width, GLuint Height);
00293     //uses the X11 system to initialize create an OpenGL context for the window
00294     GLboolean Linux_InitializeGL();
00295     //uses OpenGL extensions for Linux to toggle Vertical syncing
00296     void Linux_VerticalSync(GLint EnableSync);
00297     //shut down the window. closes all connections to the X11 system
00298     void Linux_Shutdown();
00299     //enables given window decoration via Win32
00300     void Linux_EnableDecorator(GLbitfield Decorator);
00301     //disables given window decoration via Win32
00302     void Linux_DisableDecorator(GLbitfield Decorator);
00303     //set the style of the window by enabling/disabling certain decorators
00304     void Linux_SetStyle(GLuint WindowStyle);
00305
00306     //initialize the window manager Atomics needed for the X11 extended window manager
00307     void InitializeAtomics();
00308     //initialize the NEEDED OpenGL extensions that are supported on Linux
00309     void Linux_InitGLExtensions();
00310
00311     //get the Handle To the Window
00312     Window GetWindowHandle();
00313
00314     GLXFBConfig GetBestFramebufferConfig();
00315
00316     Window WindowHandle;
00317     GLXContext Context;
00318     XVisualInfo* VisualInfo;
00319     GLint* Attributes;
00320     XSetWindowAttributes SetAttributes;
00321     GLbitfield Decorators;
00322     //these are the callbacks for the GLX swap interval extension.
00323     PFNGLXSWAPINTERVALEMESAPROC SwapIntervalMESA;
00324     PFNGLXSWAPINTERVALEXTPROC SwapIntervalEXT;
00325     PFNGLXSWAPINTERVALSGIPROC SwapIntervalSGI;
00326     /*these atomics are needed to change window states via the extended window manager
00327     I might move them to window manager considering these are essentially constants
00328     */
00329     Atom AtomState; // _NET_WM_STATE
00330     Atom AtomHidden; // _NET_WM_STATE_HIDDEN
00331     Atom AtomFullScreen; // _NET_WM_STATE_FULLSCREEN
00332     Atom AtomMaxHorz; // _NET_WM_STATE_MAXIMIZED_HORZ
00333     Atom AtomMaxVert; // _NET_WM_STATE_MAXIMIZED_VERT
00334     Atom AtomClose; // _NET_WM_CLOSE_WINDOW
00335     Atom AtomActive; // _NET_ACTIVE_WINDOW
00336     Atom AtomDemandsAttention; // _NET_WM_STATE_DEMANDS_ATTENTION
00337     Atom AtomFocused; // _NET_WM_STATE_FOCUSED
00338     Atom AtomCardinal; // _NET_WM_CARDINAL
00339     Atom AtomIcon; // _NET_WM_ICON
00340     Atom AtomHints; // _NET_WM_HINTS
00341
00342     Atom AtomWindowType;
00343     Atom AtomWindowTypeDesktop; // _NET_WM_WINDOW_TYPE_SPLASH
00344     Atom AtomWindowTypeSplash;
00345     Atom AtomWindowTypeNormal;
00346     Atom AtomAllowedActions;
00347     Atom AtomActionResize;
00348     Atom AtomActionMinimize;
00349     Atom AtomActionShade;
00350     Atom AtomActionMaximizeHorz;
00351     Atom AtomActionMaximizeVert;
00352     Atom AtomActionClose;

```

```

00357     Atom AtomDesktopGeometry;
00359 #endif
00360 };
00361
00362 #endif

```

## 4.7 Window\_Linux.cpp File Reference

```

#include "Window.h"
#include "WindowManager.h"

```

## 4.8 Window\_Linux.cpp

```

00001 /*****
00007 #include "Window.h"
00008 #include "WindowManager.h"
00009 #if defined(CURRENT_OS_LINUX)
00010 #include <cstring>
00011 *****/
00022 GLboolean FWindow::Linux_Initialize()
00023 {
00024     Attributes = new GLint[12]{GLX_DOUBLEBUFFER, GLX_DEPTH_SIZE, DepthBits, GLX_STENCIL_SIZE,
00025                               StencilBits,
00026                               GLX_RED_SIZE, ColourBits, GLX_GREEN_SIZE, ColourBits, GLX_BLUE_SIZE,
00027                               ColourBits, None};
00028     Decorators = 1;
00029     CurrentWindowStyle |= LINUX_DECORATOR_CLOSE |
00030                        LINUX_DECORATOR_MAXIMIZE | LINUX_DECORATOR_MINIMIZE |
00031                        LINUX_DECORATOR_MOVE;
00032     if (!WindowManager::GetDisplay())
00033     {
00034         PrintErrorMessage(ERROR_LINUX_CANNOTCONNECTXSERVER);
00035         exit(0);
00036     }
00037     VisualInfo = glXGetVisualFromFBConfig(WindowManager::GetDisplay(), GetBestFrameBufferConfig());
00038     //VisualInfo = glXChooseVisual(WindowManager::GetDisplay(), 0, Attributes);
00039     if (!VisualInfo)
00040     {
00041         PrintErrorMessage(ERROR_LINUX_INVALIDVISUALINFO);
00042         exit(0);
00043     }
00044     SetAttributes.colormap = XCreateColormap(WindowManager::GetDisplay(),
00045                                              DefaultRootWindow(WindowManager::GetDisplay()),
00046                                              VisualInfo->visual, AllocNone);
00047     SetAttributes.event_mask = ExposureMask | KeyPressMask
00048                             | KeyReleaseMask | MotionNotify | ButtonPressMask | ButtonReleaseMask
00049                             | FocusIn | FocusOut | Button1MotionMask | Button2MotionMask | Button3MotionMask |
00050                             Button4MotionMask | Button5MotionMask | PointerMotionMask | FocusChangeMask |
00051                             VisibilityChangeMask | PropertyChangeMask | SubstructureNotifyMask;
00052     WindowHandle = XCreateWindow(WindowManager::GetInstance()->m_Display,
00053                                  XDefaultRootWindow(WindowManager::GetInstance()->m_Display), 0, 0,
00054                                  Resolution[0], Resolution[1],
00055                                  0, VisualInfo->depth, InputOutput,
00056                                  VisualInfo->visual, CWColormap | CWEventMask,
00057                                  &SetAttributes);
00058     if (!WindowHandle)
00059     {
00060         PrintErrorMessage(ERROR_LINUX_CANNOTCREATEWINDOW);
00061         exit(0);
00062     }
00063     XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00064     XStoreName(WindowManager::GetDisplay(), WindowHandle,
00065               Name);
00066     InitializeAtoms();
00067 }

```

```

00075     XSetWMProtocols(WindowManager::GetDisplay(), WindowHandle, &AtomClose, GL_TRUE);
00076
00077     return Linux_InitializeGL();
00078 }
00079
00080 /*****
00081 void FWindow::Linux_Shutdown()
00082 {
00083     if(CurrentState == WINDOWSTATE_FULLSCREEN)
00084     {
00085         Restore();
00086     }
00087
00088     glXDestroyContext(WindowManager::GetDisplay(), Context);
00089     XUnmapWindow(WindowManager::GetDisplay(), WindowHandle);
00090     XDestroyWindow(WindowManager::GetDisplay(), WindowHandle);
00091     WindowHandle = 0;
00092     Context = 0;
00093 }
00094
00095 /*****
00096 void FWindow::Linux_SetResolution(GLuint Width, GLuint Height)
00097 {
00098     Resolution[0] = Width;
00099     Resolution[1] = Height;
00100     XResizeWindow(WindowManager::GetDisplay(),
00101         WindowHandle, Resolution[0], Resolution[1]);
00102 }
00103
00104 /*****
00105 void FWindow::Linux_SetPosition(GLuint X, GLuint Y)
00106 {
00107     XWindowChanges l_WindowChanges;
00108
00109     l_WindowChanges.x = X;
00110     l_WindowChanges.y = Y;
00111
00112     XConfigureWindow(
00113         WindowManager::GetDisplay(),
00114         WindowHandle, CWX | CWY, &l_WindowChanges);
00115 }
00116
00117 /*****
00118 void FWindow::Linux_SetMousePosition(GLuint X, GLuint Y)
00119 {
00120     XWarpPointer(
00121         WindowManager::GetInstance()->m_Display,
00122         WindowHandle, WindowHandle,
00123         Position[0], Position[1],
00124         Resolution[0], Resolution[1],
00125         X, Y);
00126 }
00127
00128 /*****
00129 void FWindow::Linux_FullScreen(GLboolean ShouldBeFullscreen)
00130 {
00131     XEvent l_Event;
00132     memset(&l_Event, 0, sizeof(l_Event));
00133
00134     l_Event.xany.type = ClientMessage;
00135     l_Event.xclient.message_type = AtomState;
00136     l_Event.xclient.format = 32;
00137     l_Event.xclient.window = WindowHandle;
00138     l_Event.xclient.data.l[0] = ShouldBeFullscreen;
00139     l_Event.xclient.data.l[1] = AtomFullscreen;
00140
00141     XSendEvent(WindowManager::GetDisplay(),
00142         XDefaultRootWindow(WindowManager::GetDisplay()),
00143         0, SubstructureNotifyMask, &l_Event);
00144 }
00145
00146 /*****
00147 void FWindow::Linux_Minimize(GLboolean ShouldBeMinimized)
00148 {
00149     if(ShouldBeMinimized)
00150     {
00151         XIconifyWindow(WindowManager::GetDisplay(),
00152             WindowHandle, 0);
00153     }
00154     else
00155     {
00156         XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00157     }
00158 }
00159
00160 *****/

```

```

00233 void FWindow::Linux_Maximize(GLboolean ShouldBeMaximized)
00234 {
00235     XEvent l_Event;
00236     memset(&l_Event, 0, sizeof(l_Event));
00237
00238     l_Event.xany.type = ClientMessage;
00239     l_Event.xclient.message_type = AtomState;
00240     l_Event.xclient.format = 32;
00241     l_Event.xclient.window = WindowHandle;
00242     l_Event.xclient.data.l[0] = ShouldBeMaximized;
00243     l_Event.xclient.data.l[1] = AtomMaxVert;
00244     l_Event.xclient.data.l[2] = AtomMaxHorz;
00245
00246     XSendEvent(WindowManager::GetDisplay(),
00247               XDefaultRootWindow(WindowManager::GetDisplay()),
00248               0, SubstructureNotifyMask, &l_Event);
00249 }
00250
00251 /*****
00260 void FWindow::Linux_Restore()
00261 {
00262     XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00263 }
00264
00265 /*****
00276 void FWindow::Linux_SetTitleBar(const char* NewTitle)
00277 {
00278     XStoreName(WindowManager::GetDisplay(),
00279               WindowHandle, NewTitle);
00280 }
00281
00282 void FWindow::Linux_SetIcon(const char* Icon, GLuint Width, GLuint Height)
00283 {
00284
00285     unsigned char* Image32 = (unsigned char*)malloc(Width * Height * 4);
00286
00287     //XImage* Image = XCreateImage(WindowManager::GetDisplay(), CopyFromParent, DepthBits, ZPixmap, 0,
00288     Image32,
00289     //Width, Height, ColourBits, 0);
00290
00291     FILE* l_File = fopen(Icon, "r");
00292
00293     if(l_File == 0)
00294     {
00295         printf("file not found \n");
00296     }
00297
00298     fclose(l_File);
00299
00300     Pixmap pix = XCreatePixmap(WindowManager::GetDisplay(), XDefaultRootWindow(WindowManager::GetDisplay())
00301     ,
00302     Width, Height, ColourBits);
00303
00304     XGCValues Values;
00305     GC gc = XCreateGC(WindowManager::GetDisplay(), pix, 0, &Values);
00306
00307     //XPutImage(WindowManager::GetDisplay(), pix, gc, Image, Width, Height, 0, 0, Width, Height);
00308
00309     GLuint width, height, X_Hot, Y_Hot = 0;
00310
00311     //XReadBitmapFile(Icon, &width, &height, &Data, &X_Hot, &Y_Hot);
00312
00313     //printf("%i %i %i %i \n", width, height, X_Hot, Y_Hot);
00314     //printf("%s\n", Data);
00315
00316     //XChangeProperty(WindowManager::GetDisplay(), WindowHandle,
00317     //AtomIcon, AtomCardinal, 32, PropModeReplace, (unsigned char*)Image, sizeof(Image));
00318
00319     system("ls");
00320     //XEvent Event
00321     //memset(&Event, 0, sizeof(Event));
00322
00323     //l_Event.xany.type = ClientMessage;
00324     //l_Event.xclient.message_type = AtomState;
00325 }
00326
00327 /*****
00338 void FWindow::Linux_Focus(GLboolean ShouldBeInFocus)
00339 {
00340     if(InFocus)
00341     {
00342         XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00343     }
00344
00345     else

```

```

00346     {
00347         XUnmapWindow(WindowManager::GetDisplay(), WindowHandle);
00348     }
00349 }
00350
00351 /*****
00362 void FWindow::Linux_VerticalSync(GLint EnableSync)
00363 {
00364     if(EXTSwapControlSupported)
00365     {
00366         SwapIntervalEXT(WindowManager::GetDisplay(), WindowHandle, EnableSync);
00367     }
00368     if(MESASwapControlSupported)
00369     {
00370         SwapIntervalMESA(EnableSync);
00371     }
00372     if(SGISwapControlSupported)
00373     {
00374         if(EnableSync < 0)
00375         {
00376             EnableSync = 0;
00377         }
00378         SwapIntervalSGI(EnableSync);
00379     }
00380 }
00381 }
00382 }
00383
00384 /*****
00395 GLboolean FWindow::Linux_InitializeGL()
00396 {
00397     if(!Context)
00398     {
00399         Context = glXCreateContext(
00400             WindowManager::GetDisplay(),
00401             VisualInfo, 0, GL_TRUE);
00402     }
00403     if(Context)
00404     {
00405         glXMakeCurrent(WindowManager::GetDisplay(),
00406             WindowHandle, Context);
00407
00408         XWindowAttributes l_Attributes;
00409
00410         XGetWindowAttributes(WindowManager::GetDisplay(),
00411             WindowHandle, &l_Attributes);
00412         Position[0] = l_Attributes.x;
00413         Position[1] = l_Attributes.y;
00414
00415         const char* ExtensionsAvailable = 0;
00416
00417         ExtensionsAvailable = glXQueryExtensionsString(WindowManager::GetDisplay(), 0);
00418
00419         if(!ExtensionsAvailable)
00420         {
00421             PrintWarningMessage(WARNING_NOGLEXENSIONS);
00422         }
00423         else
00424         {
00425             InitGLExtensions();
00426         }
00427         ContextCreated = GL_TRUE;
00428         return FOUNDATION_OKAY;
00429     }
00430 }
00431 }
00432 }
00433
00434 else
00435 {
00436     PrintErrorMessage(ERROR_EXISTINGCONTEXT);
00437     return FOUNDATION_ERROR;
00438 }
00439
00440 return FOUNDATION_ERROR;
00441 }
00442
00443 /*****
00452 void FWindow::InitializeAtomics()
00453 {
00454     AtomState = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE", GL_FALSE);
00455     AtomFullScreen = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_FULLSCREEN", GL_FALSE);
00456     AtomMaxHorz = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_MAXIMIZED_HORZ", GL_FALSE);
00457     AtomMaxVert = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_MAXIMIZED_VERT", GL_FALSE);
00458     AtomClose = XInternAtom(WindowManager::GetDisplay(), "WM_DELETE_WINDOW", GL_FALSE);
00459     AtomHidden = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_HIDDEN", GL_FALSE);
00460     AtomActive = XInternAtom(WindowManager::GetDisplay(), "_NET_ACTIVE_WINDOW", GL_FALSE);

```

```

00461     AtomDemandsAttention = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_DEMANDS_ATTENTION",
GL_FALSE);
00462     AtomFocused = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_FOCUSED", GL_FALSE);
00463     AtomCardinal = XInternAtom(WindowManager::GetDisplay(), "CARDINAL", GL_FALSE);
00464     AtomIcon = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_ICON", GL_FALSE);
00465     AtomHints = XInternAtom(WindowManager::GetDisplay(), "_MOTIF_WM_HINTS", GL_TRUE);
00466
00467     AtomWindowType = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE", GL_FALSE);
00468     AtomWindowTypeDesktop = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_UTILITY",
GL_FALSE);
00469     AtomWindowTypeSplash = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_SPLASH", GL_FALSE);
00470 ;
00471     AtomWindowTypeNormal = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_NORMAL", GL_FALSE);
00472 ;
00473     AtomAllowedActions = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_ALLOWED_ACTIONS", GL_FALSE);
00474     AtomActionResize = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_RESIZE", GL_FALSE);
00475     AtomActionMinimize = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_MINIMIZE", GL_FALSE);
00476     AtomActionShade = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_SHADE", GL_FALSE);
00477     AtomActionMaximizeHorz = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_MAXIMIZE_HORZ", GL_FALSE);
00478 ;
00479     AtomActionMaximizeVert = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_MAXIMIZE_VERT", GL_FALSE);
00480 ;
00481     AtomActionClose = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_CLOSE", GL_FALSE);
00482
00483     AtomDesktopGeometry = XInternAtom(WindowManager::GetDisplay(), "_NET_DESKTOP_GEOMETRY", GL_FALSE);
00484 }
00485
00486 /*****
00492 void FWindow::Linux_InitGLExtensions()
00493 {
00494     SwapIntervalEXT = nullptr;
00495     SwapIntervalSGI = nullptr;
00496     SwapIntervalMESA = nullptr;
00497
00498     SwapIntervalMESA = (PFNGLXSWAPINTERVALMESAPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalMESA");
00499 ;
00500     SwapIntervalEXT = (PFNGLXSWAPINTERVALEXTPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalEXT");
00501     SwapIntervalSGI = (PFNGLXSWAPINTERVALSGIPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalSGI");
00502
00503     if(SwapIntervalMESA)
00504     {
00505         //printf("MESA swap interval supported\n");
00506         MESASwapControlSupported = GL_TRUE;
00507     }
00508
00509     if(SwapIntervalEXT)
00510     {
00511         //printf("EXT swap interval supported \n");
00512         EXTSwapControlSupported = GL_TRUE;
00513     }
00514
00515     if(SwapIntervalSGI)
00516     {
00517         //printf("SGI swap interval supported \n");
00518         SGISwapControlSupported = GL_TRUE;
00519     }
00520 }
00521 /*****
00532 Window FWindow::GetWindowHandle()
00533 {
00534     return WindowHandle;
00535 }
00536
00537 void FWindow::Linux_EnableDecorator(GLbitfield Decorator)
00538 {
00539     if(Decorator & DECORATOR_CLOSEBUTTON)
00540     {
00541         CurrentWindowStyle |= LINUX_DECORATOR_CLOSE;
00542         Decorators = 1;
00543     }
00544
00545     if(Decorator & DECORATOR_MINIMIZEBUTTON)
00546     {
00547         CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
00548         Decorators = 1;
00549     }
00550
00551     if(Decorator & DECORATOR_MAXIMIZEBUTTON)
00552     {
00553         CurrentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
00554         Decorators = 1;
00555     }
00556
00557     if(Decorator & DECORATOR_ICON)
00558     {

```



```

00559         //Linux (at least cinammon) doesnt have icons in the window. only in the taskbar icon
00560     }
00561
00562     //just need to set it to 1 to enable all decorators that include title bar
00563     if(Decorator & DECORATOR_TITLEBAR)
00564     {
00565         Decorators = 1;
00566     }
00567
00568     if(Decorator & DECORATOR_BORDER)
00569     {
00570         Decorators = 1;
00571     }
00572
00573     if(Decorator & DECORATOR_SIZEABLEBORDER)
00574     {
00575         Decorators = 1;
00576     }
00577
00578     long hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR,
CurrentWindowStyle, Decorators, 0, 0};
00579
00580     XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
00581         PropModeReplace, (unsigned char*) hints, 5);
00582
00583     XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00584 }
00585
00586 void FWindow::Linux_DisableDecorator(GLbitfield Decorator)
00587 {
00588     if(Decorator & DECORATOR_CLOSEBUTTON)
00589     {
00590         //I hate doing this but it is neccessary to keep functionality going.
00591         GLboolean MinimizeEnabled, MaximizeEnabled;
00592
00593         if(Decorator & DECORATOR_MAXIMIZEBUTTON)
00594         {
00595             MaximizeEnabled = GL_TRUE;
00596         }
00597
00598         if(Decorator & DECORATOR_MINIMIZEBUTTON)
00599         {
00600             MinimizeEnabled = GL_TRUE;
00601         }
00602
00603         CurrentWindowStyle &= ~LINUX_DECORATOR_CLOSE;
00604
00605         if(MaximizeEnabled)
00606         {
00607             CurrentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
00608         }
00609
00610         if(MinimizeEnabled)
00611         {
00612             CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
00613         }
00614
00615         Decorators = 1;
00616     }
00617
00618     if(Decorator & DECORATOR_MINIMIZEBUTTON)
00619     {
00620         CurrentWindowStyle &= ~LINUX_DECORATOR_MINIMIZE;
00621         Decorators = 1;
00622     }
00623
00624     if(Decorator & DECORATOR_MAXIMIZEBUTTON)
00625     {
00626         GLboolean MinimizeEnabled;
00627
00628         if(Decorator & DECORATOR_MINIMIZEBUTTON)
00629         {
00630             MinimizeEnabled = GL_TRUE;
00631         }
00632
00633         CurrentWindowStyle &= ~LINUX_DECORATOR_MAXIMIZE;
00634
00635         if(MinimizeEnabled)
00636         {
00637             CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
00638         }
00639
00640         Decorators = 1;
00641     }
00642
00643     if(Decorator & DECORATOR_ICON)
00644     {

```

```

00645         //Linux (at least cinammon) doesnt have icons in the window. only in the taskbar icon
00646     }
00647
00648     //just need to set it to 1 to enable all decorators that include title bar
00649     if(Decorator & DECORATOR_TITLEBAR)
00650     {
00651         Decorators = LINUX_DECORATOR_BORDER;
00652     }
00653
00654     if(Decorator & DECORATOR_BORDER)
00655     {
00656         Decorators = 0;
00657     }
00658
00659     if(Decorator & DECORATOR_SIZEABLEBORDER)
00660     {
00661         Decorators = 0;
00662     }
00663
00664     long hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR,
CurrentWindowStyle, Decorators, 0, 0};
00665
00666     XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
PropModeReplace, (unsigned char*) hints, 5);
00667
00668     XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00669 }
00670
00671 void FWindow::Linux_SetStyle(GLuint WindowStyle)
00672 {
00673     switch(WindowStyle)
00674     {
00675     case WINDOWSTYLE_DEFAULT:
00676     {
00677         Decorators = (1L << 2);
00678         CurrentWindowStyle = LINUX_DECORATOR_MOVE |
LINUX_DECORATOR_CLOSE |
00680             LINUX_DECORATOR_MAXIMIZE |
LINUX_DECORATOR_MINIMIZE;
00681         long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR,
CurrentWindowStyle, Decorators, 0, 0};
00682
00683         XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
PropModeReplace,
00684             (unsigned char*)Hints, 5);
00685
00686         XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00687
00688         break;
00689     }
00690
00691     case WINDOWSTYLE_BARE:
00692     {
00693         Decorators = (1L << 2);
00694         CurrentWindowStyle = (1L << 2);
00695         long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR,
CurrentWindowStyle, Decorators, 0, 0};
00696
00697         XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
PropModeReplace,
00698             (unsigned char*)Hints, 5);
00699
00700         XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00701
00702         break;
00703     }
00704
00705     case WINDOWSTYLE_POPUP:
00706     {
00707         Decorators = 0;
00708         CurrentWindowStyle = (1L << 2);
00709         long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR,
CurrentWindowStyle, Decorators, 0, 0};
00710
00711         XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
PropModeReplace,
00712             (unsigned char*)Hints, 5);
00713
00714         XMapWindow(WindowManager::GetDisplay(), WindowHandle);
00715
00716         break;
00717     }
00718
00719     default:
00720     {
00721
00722

```

```

00723             PrintErrorMessage(ERROR_INVALIDWINDOWSTYLE);
00724             break;
00725         }
00726     }
00727 }
00728
00729 GLXFBConfig FWindow::GetBestFrameBufferConfig()
00730 {
00731     const GLint VisualAttributes[] =
00732     {
00733         GLX_X_RENDERABLE, GL_TRUE,
00734         GLX_DRAWABLE_TYPE, GLX_WINDOW_BIT,
00735         GLX_X_VISUAL_TYPE, GLX_TRUE_COLOR,
00736         GLX_RED_SIZE, ColourBits,
00737         GLX_GREEN_SIZE, ColourBits,
00738         GLX_BLUE_SIZE, ColourBits,
00739         GLX_ALPHA_SIZE, ColourBits,
00740         GLX_DEPTH_SIZE, DepthBits,
00741         GLX_STENCIL_SIZE, StencilBits,
00742         GLX_DOUBLEBUFFER, GL_TRUE,
00743         None
00744     };
00745
00746     GLint FrameBufferCount;
00747     GLuint BestBufferConfig, BestNumSamples = 0;
00748     GLXFBConfig* Configs = glXChooseFBConfig(WindowManager::GetDisplay(), 0, VisualAttributes, &
FrameBufferCount);
00749
00750     for(GLuint CurrentConfig = 0; CurrentConfig < FrameBufferCount; CurrentConfig++)
00751     {
00752         XVisualInfo* VisInfo = glXGetVisualFromFBConfig(WindowManager::GetDisplay(), Configs[CurrentConfig]
);
00753
00754         if(VisInfo)
00755         {
00756             //printf("%i %i %i\n", VisInfo->depth, VisInfo->bits_per_rgb, VisInfo->colormap_size);
00757             GLint Samples, SampleBuffer;
00758             glXGetFBConfigAttrib(WindowManager::GetDisplay(), Configs[CurrentConfig], GLX_SAMPLE_BUFFERS, &
SampleBuffer);
00759             glXGetFBConfigAttrib(WindowManager::GetDisplay(), Configs[CurrentConfig], GLX_SAMPLES, &Samples
);
00760
00761             if(SampleBuffer && Samples > -1)
00762             {
00763                 BestBufferConfig = CurrentConfig;
00764                 BestNumSamples = Samples;
00765             }
00766         }
00767         XFree(VisInfo);
00768     }
00769
00770     GLXFBConfig BestConfig = Configs[BestBufferConfig];
00771
00772     XFree(Configs);
00773
00774     return BestConfig;
00775 }
00776 }
00777 #endif

```

## 4.9 Window\_Windows.cpp File Reference

```

#include "Window.h"
#include "WindowManager.h"

```

## 4.10 Window\_Windows.cpp

```

00001 /*****
00007 #include "Window.h"
00008 #include "WindowManager.h"
00009
00010 #if defined(CURRENT_OS_WINDOWS)
00011
00012 /*****
00023 HWND FWindow::GetWindowHandle()
00024 {
00025     return WindowHandle;

```

```

00026 }
00027
00028 /*****
00039 GLboolean FWindow::Windows_InitializeGL()
00040 {
00041
00042     DeviceContextHandle = GetDC(WindowHandle);
00043     InitializePixelFormat();
00044     GLRenderingContextHandle = wglCreateContext(DeviceContextHandle);
00045     wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);
00046
00047     ContextCreated = (GLRenderingContextHandle != nullptr);
00048
00049     if (ContextCreated)
00050     {
00051 #ifndef CONTEXT_CREATED
00052 #define CONTEXT_CREATED
00053 #endif
00054         Windows_InitGLExtensions();
00055         return FOUNDATION_OKAY;
00056     }
00057
00058     /*****
00069     PrintErrorMessage(ERROR_INVALIDCONTEXT);
00070     return FOUNDATION_ERROR;
00071     */
00072
00073 /*****
00082 void FWindow::InitializePixelFormat()
00083 {
00084     PixelFormatDescriptor = {
00085         sizeof(PIXELFORMATDESCRIPTOR), /* size */
00086         1, /* version */
00087         PFD_SUPPORT_OPENGL |
00088         PFD_DRAW_TO_WINDOW |
00089         PFD_DOUBLEBUFFER, /* support double-buffering */
00090         PFD_TYPE_RGBA, /* color type */
00091         ColourBits, 0, /* preferred color depth */
00092         0, 0,
00093         0, 0,
00094         0, 0,
00095         0, /* color bits (ignored) */ /* no alpha buffer */
00096         0, /* alpha bits (ignored) */
00097         0, 0, 0, 0, /* no accumulation buffer */
00098         DepthBits, /* accum bits (ignored) */
00099         StencilBits, /* depth buffer */
00100         0, /* no stencil buffer */
00101         0, /* no auxiliary buffers */
00102         PFD_MAIN_PLANE, /* main layer */
00103         0, /* reserved */
00104         0, 0, 0, /* no layer, visible, damage masks */
00105     };
00106
00107     int l_PixelFormat = ChoosePixelFormat(DeviceContextHandle, &PixelFormatDescriptor);
00108
00109     if (l_PixelFormat)
00110     {
00111         SetPixelFormat(DeviceContextHandle, l_PixelFormat, &PixelFormatDescriptor);
00112         return;
00113     }
00114 }
00115
00116 GLboolean FWindow::Windows_Initialize(
00117     UINT a_Style /* = CS_OWNDC | CS_HREDRAW | CS_DROPSHADOW */,
00118     int a_ClearScreenExtra /* = 0 */,
00119     int WindowExtra /* = 0 */,
00120     HINSTANCE a_Instance /* = GetModuleHandle(0) */,
00121     HICON a_Icon /* = LoadIcon(0, IDI_APPLICATION) */,
00122     HCURSOR a_Cursor /* = LoadCursor(0, IDC_ARROW) */,
00123     HBRUSH a_Brush /* = (HBRUSH)BLACK_BRUSH */)
00124 {
00125     InstanceHandle = a_Instance;
00126     WindowClass.style = a_Style;
00127     WindowClass.lpfnWndProc = WindowManager::StaticWindowProcedure;
00128     WindowClass.cbClsExtra = 0;
00129     WindowClass.cbWndExtra = 0;
00130     WindowClass.hInstance = InstanceHandle;
00131     WindowClass.hIcon = a_Icon;
00132     WindowClass.hCursor = a_Cursor;
00133     WindowClass.hbrBackground = a_Brush;
00134     WindowClass.lpszMenuName = Name;
00135     WindowClass.lpszClassName = Name;
00136     RegisterClass(&WindowClass);
00137
00138     CurrentWindowStyle = WS_OVERLAPPEDWINDOW;
00139

```

```

00140     WindowHandle =
00141         CreateWindow(Name, Name, CurrentWindowStyle, 0,
00142             0, Resolution[0],
00143             Resolution[1],
00144             0, 0, 0, 0);
00145
00146     if (WindowHandle)
00147     {
00148         ShowWindow(WindowHandle, GL_TRUE);
00149         UpdateWindow(WindowHandle);
00150         return FOUNDATION_OKAY;
00151     }
00152
00153     PrintErrorMessage(ERROR_WINDOWS_CANNOTCREATEWINDOW);
00154     return FOUNDATION_ERROR;
00155 }
00156
00157 void FWindow::Windows_Shutdown()
00158 {
00159     if (GLRenderingContextHandle)
00160     {
00161         wglMakeCurrent(nullptr, nullptr);
00162         wglDeleteContext(GLRenderingContextHandle);
00163     }
00164
00165     if (PaletteHandle)
00166     {
00167         DeleteObject(PaletteHandle);
00168     }
00169     ReleaseDC(WindowHandle, DeviceContextHandle);
00170     UnregisterClass(Name, InstanceHandle);
00171
00172     FreeModule(InstanceHandle);
00173
00174     DeviceContextHandle = nullptr;
00175     WindowHandle = nullptr;
00176     GLRenderingContextHandle = nullptr;
00177
00178
00179     //exit here or the loop will just keep running
00180     //exit(FOUNDATION_OKAY);
00181 }
00182
00183 void FWindow::Windows_FullScreen()
00184 {
00185
00186     SetWindowLongPtr(WindowHandle, GWL_STYLE,
00187         WS_SYSMENU | WS_POPUP | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_VISIBLE);
00188
00189     MoveWindow(WindowHandle, 0, 0, WindowManager::GetScreenResolution
00190         () [0],
00191             WindowManager::GetScreenResolution() [1], GL_TRUE);
00192
00193     //
00194     /*
00195     DEVMODE l_ScreenSettings;
00196     memset(&l_ScreenSettings, 0, sizeof(l_ScreenSettings));
00197     l_ScreenSettings.dmSize = sizeof(l_ScreenSettings);
00198     l_ScreenSettings.dmPelsWidth = F_WM::GetScreenResolution() [0];
00199     l_ScreenSettings.dmPelsHeight = F_WM::GetScreenResolution() [1];
00200     l_ScreenSettings.dmBitsPerPel = m_ColourBits;
00201     l_ScreenSettings.dmFields = DM_PELSWIDTH | DM_PELSHEIGHT | DM_BITSPERPEL;
00202
00203     if (ChangeDisplaySettings(&l_ScreenSettings, CDS_FULLSCREEN) != DISP_CHANGE_SUCCESSFUL)
00204     {
00205         printf("could not successfully change to full screen mode \n");
00206     }*/
00207
00208
00209     /*RECT l_Rect;
00210     l_Rect.left = 0;
00211     l_Rect.top = 0;
00212     l_Rect.right = m_Resolution[0];
00213     l_Rect.bottom = m_Resolution[1];
00214
00215     DEVMODE l_ScreenSettings;
00216
00217     l_ScreenSettings.dmSize = sizeof(l_ScreenSettings);
00218     l_ScreenSettings.dmPelsWidth = m_Resolution[0];
00219     l_ScreenSettings.dmPelsHeight = m_Resolution[1];
00220     l_ScreenSettings.dmBitsPerPel = 32;
00221     l_ScreenSettings.dmFields = DM_BITSPERPEL | DM_PELSWIDTH | DM_PELSHEIGHT;
00222
00223     if (ChangeDisplaySettings(&l_ScreenSettings, CDS_RESET) != DISP_CHANGE_SUCCESSFUL)
00224     {
00225         printf("could not successfully change back to regular mode. dear god what have i done? \n");

```

```

00226     }
00227
00228     SetWindowLongPtr(m_WindowHandle, GWL_STYLE, WS_OVERLAPPEDWINDOW | WS_VISIBLE);
00229     AdjustWindowRect(&l_Rect, WS_OVERLAPPEDWINDOW, GL_FALSE);
00230     MoveWindow(m_WindowHandle, m_Position[0], m_Position[1], l_Rect.right, l_Rect.bottom, GL_TRUE);*/
00231 }
00232
00233 void FWindow::Windows_Minimize()
00234 {
00235     if (CurrentState == WINDOWSTATE_MINIMIZED)
00236     {
00237         ShowWindow(WindowHandle, SW_MINIMIZE);
00238     }
00239
00240     else
00241     {
00242         ShowWindow(WindowHandle, SW_RESTORE);
00243     }
00244 }
00245
00246 void FWindow::Windows_Maximize()
00247 {
00248     if (CurrentState == WINDOWSTATE_MAXIMIZED)
00249     {
00250         ShowWindow(WindowHandle, SW_MAXIMIZE);
00251     }
00252
00253     else
00254     {
00255         ShowWindow(WindowHandle, SW_RESTORE);
00256     }
00257 }
00258
00259 void FWindow::Windows_Restore()
00260 {
00261     ShowWindow(WindowHandle, SW_RESTORE);
00262 }
00263
00264 void FWindow::Windows_Focus()
00265 {
00266     if (InFocus)
00267     {
00268         SetFocus(WindowHandle);
00269     }
00270
00271     else
00272     {
00273         SetFocus(nullptr);
00274     }
00275 }
00276
00277 void FWindow::Windows_SetMousePosition(GLuint X, GLuint& Y)
00278 {
00279     POINT l_MousePoint;
00280     l_MousePoint.x = X;
00281     l_MousePoint.y = Y;
00282     ScreenToClient(WindowHandle, &l_MousePoint);
00283     SetCursorPos(l_MousePoint.x, l_MousePoint.y);
00284 }
00285
00286 void FWindow::Windows_SetTitleBar(const char* NewTitle)
00287 {
00288     SetWindowText(WindowHandle, Name);
00289 }
00290
00291 void FWindow::Windows_SetIcon(const char* Icon, GLuint Width, GLuint Height)
00292 {
00293     HANDLE icon = LoadImage(InstanceHandle, Icon,
00294         IMAGE_ICON, Width, Height, LR_LOADFROMFILE);
00295     SendMessage(WindowHandle, (UINT)WM_SETICON, ICON_BIG, (LPARAM)icon);
00296 }
00297
00298 void FWindow::Windows_SetPosition(GLuint X, GLuint Y)
00299 {
00300     RECT rect = { X, Y, X, Y };
00301     AdjustWindowRect(&rect, GWL_STYLE | WS_OVERLAPPEDWINDOW | WS_VISIBLE,
00302         GL_FALSE);
00303
00304     SetWindowPos(WindowHandle, HWND_TOP, X, Y,
00305         Resolution[0], Resolution[1], SWP_SHOWWINDOW | SWP_NOSIZE);
00306 }
00307
00308 void FWindow::Windows_SetResolution(GLuint Width, GLuint Height)
00309 {
00310     SetWindowPos(WindowHandle, HWND_TOP, Position[0], Position[1],
00311         Resolution[0], Resolution[1], SWP_SHOWWINDOW | SWP_NOMOVE);
00312 }

```

```

00313
00314 void FWindow::Windows_VerticalSync(GLint EnableSync)
00315 {
00316     if (EXTSwapControlSupported)
00317     {
00318         SwapIntervalEXT(EnableSync);
00319     }
00320 }
00321
00322 void FWindow::Windows_InitGLExtensions()
00323 {
00324     SwapIntervalEXT = nullptr;
00325     GetExtensionsStringEXT = nullptr;
00326
00327     GetExtensionsStringEXT = (PFNWGLGETEXTENSIONSSTRINGEXTPROC)
00328         wglGetProcAddress("wglGetExtensionsStringEXT");
00329
00330     SwapIntervalEXT = (PFNWGLSWAPINTERVALEXTPROC)
00331         wglGetProcAddress("wglSwapIntervalEXT");
00332
00333     if (SwapIntervalEXT)
00334     {
00335         EXTSwapControlSupported = GL_TRUE;
00336     }
00337 }
00338
00339 void FWindow::Windows_EnableDecorator(GLbitfield Decorator)
00340 {
00341     CurrentWindowStyle = WS_VISIBLE | WS_CLIPSIBLINGS;
00342
00343     if (Decorator & DECORATOR_BORDER)
00344     {
00345         CurrentWindowStyle |= WS_BORDER;
00346     }
00347
00348     if (Decorator & DECORATOR_TITLEBAR)
00349     {
00350         CurrentWindowStyle |= WS_CAPTION;
00351     }
00352
00353     if (Decorator & DECORATOR_ICON)
00354     {
00355         CurrentWindowStyle |= WS_ICONIC;
00356     }
00357
00358     if (Decorator & DECORATOR_CLOSEBUTTON)
00359     {
00360         CurrentWindowStyle |= WS_SYSMENU;
00361     }
00362
00363     if (Decorator & DECORATOR_MINIMIZEBUTTON)
00364     {
00365         CurrentWindowStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
00366     }
00367
00368     if (Decorator & DECORATOR_MAXIMIZEBUTTON)
00369     {
00370         CurrentWindowStyle |= WS_MAXIMIZEBOX | WS_SYSMENU;
00371     }
00372
00373     if (Decorator & DECORATOR_VERTICALSCROLLBAR)
00374     {
00375         CurrentWindowStyle |= WS_VSCROLL;
00376     }
00377
00378     if (Decorator & DECORATOR_HORIZONTALSCROLLBAR)
00379     {
00380         CurrentWindowStyle |= WS_HSCROLL;
00381     }
00382
00383     if (Decorator & DECORATOR_SIZEABLEBORDER)
00384     {
00385         CurrentWindowStyle |= WS_SIZEBOX;
00386     }
00387
00388     SetWindowLongPtr(WindowHandle, GWL_STYLE,
00389         CurrentWindowStyle);
00390 }
00391
00392 void FWindow::Windows_DisableDecorator(GLbitfield Decorator)
00393 {
00394     if (Decorator & DECORATOR_BORDER)
00395     {
00396         CurrentWindowStyle &= ~WS_BORDER;
00397     }
00398
00399     if (Decorator & DECORATOR_TITLEBAR)

```

```

00400     {
00401         CurrentWindowStyle &= ~WS_MAXIMIZEBOX;
00402     }
00403
00404     if (Decorator & DECORATOR_ICON)
00405     {
00406         CurrentWindowStyle &= ~WS_ICONIC;
00407     }
00408
00409     if (Decorator & DECORATOR_CLOSEBUTTON)
00410     {
00411         CurrentWindowStyle &= ~WS_SYSMENU;
00412     }
00413
00414     if (Decorator & DECORATOR_MINIMIZEBUTTON)
00415     {
00416         CurrentWindowStyle &= ~WS_MINIMIZEBOX;
00417     }
00418
00419     if (Decorator & DECORATOR_MAXIMIZEBUTTON)
00420     {
00421         CurrentWindowStyle &= ~WS_MAXIMIZEBOX;
00422     }
00423
00424     if (Decorator & DECORATOR_VERTICALSCROLLBAR)
00425     {
00426         CurrentWindowStyle &= ~WS_VSCROLL;
00427     }
00428
00429     if (Decorator & DECORATOR_HORIZONTALSCROLLBAR)
00430     {
00431         CurrentWindowStyle &= ~WS_HSCROLL;
00432     }
00433
00434     if (Decorator & DECORATOR_SIZEABLEBORDER)
00435     {
00436         CurrentWindowStyle &= ~WS_SIZEBOX;
00437     }
00438
00439     SetWindowLongPtr(WindowHandle, GWL_STYLE,
00440         CurrentWindowStyle | WS_VISIBLE);
00441 }
00442
00443 void FWindow::Windows_SetStyle(GLuint WindowType)
00444 {
00445     switch (WindowType)
00446     {
00447     case WINDOWSTYLE_DEFAULT:
00448     {
00449         EnableDecorator(DECORATOR_TITLEBAR | DECORATOR_BORDER |
00450             DECORATOR_CLOSEBUTTON | DECORATOR_MINIMIZEBUTTON | DECORATOR_MAXIMIZEBUTTON);
00451         break;
00452     }
00453
00454     case WINDOWSTYLE_POPUP:
00455     {
00456         EnableDecorator(0);
00457         break;
00458     }
00459
00460     case WINDOWSTYLE_BARE:
00461     {
00462         EnableDecorator(DECORATOR_TITLEBAR | DECORATOR_BORDER);
00463         break;
00464     }
00465
00466     default:
00467     {
00468         PrintErrorMessage(ERROR_INVALIDWINDOWSTYLE);
00469         break;
00470     }
00471     }
00472 }
00473
00474 #endif

```

## 4.11 WindowAPI\_Defs.h File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <list>

```



## Macros

- `#define KEYSTATE_DOWN 1`
- `#define KEYSTATE_UP 0`
- `#define KEY_ERROR -1`
- `#define KEY_FIRST 256 + 1`
- `#define KEY_F1 KEY_FIRST`
- `#define KEY_F2 KEY_FIRST + 1`
- `#define KEY_F3 KEY_FIRST + 2`
- `#define KEY_F4 KEY_FIRST + 3`
- `#define KEY_F5 KEY_FIRST + 4`
- `#define KEY_F6 KEY_FIRST + 5`
- `#define KEY_F7 KEY_FIRST + 6`
- `#define KEY_F8 KEY_FIRST + 7`
- `#define KEY_F9 KEY_FIRST + 8`
- `#define KEY_F10 KEY_FIRST + 9`
- `#define KEY_F11 KEY_FIRST + 10`
- `#define KEY_F12 KEY_FIRST + 11`
- `#define KEY_CAPSLOCK KEY_FIRST + 12`
- `#define KEY_LEFTSHIFT KEY_FIRST + 13`
- `#define KEY_RIGHTSHIFT KEY_FIRST + 14`
- `#define KEY_LEFTCONTROL KEY_FIRST + 15`
- `#define KEY_RIGHTCONTROL KEY_FIRST + 16`
- `#define KEY_LEFTWINDOW KEY_FIRST + 17`
- `#define KEY_RIGHTWINDOW KEY_FIRST + 18`
- `#define KEY_LEFTALT KEY_FIRST + 19`
- `#define KEY_RIGHTALT KEY_FIRST + 20`
- `#define KEY_ENTER KEY_FIRST + 21`
- `#define KEY_PRINTSCREEN KEY_FIRST + 22`
- `#define KEY_SCROLLLOCK KEY_FIRST + 23`
- `#define KEY_NUMLOCK KEY_FIRST + 24`
- `#define KEY_PAUSE KEY_FIRST + 25`
- `#define KEY_INSERT KEY_FIRST + 26`
- `#define KEY_HOME KEY_FIRST + 27`
- `#define KEY_END KEY_FIRST + 28`
- `#define KEY_PAGEUP KEY_FIRST + 28`
- `#define KEY_PAGEDOWN KEY_FIRST + 30`
- `#define KEY_ARROW_DOWN KEY_FIRST + 31`
- `#define KEY_ARROW_UP KEY_FIRST + 32`
- `#define KEY_ARROW_LEFT KEY_FIRST + 33`
- `#define KEY_ARROW_RIGHT KEY_FIRST + 34`
- `#define KEY_KEYPAD_DIVIDE KEY_FIRST + 35`
- `#define KEY_KEYPAD_MULTIPLY KEY_FIRST + 36`
- `#define KEY_KEYPAD_SUBTRACT KEY_FIRST + 37`
- `#define KEY_KEYPAD_ADD KEY_FIRST + 38`
- `#define KEY_KEYPAD_ENTER KEY_FIRST + 39`
- `#define KEY_KEYPAD_PERIOD KEY_FIRST + 40`
- `#define KEY_KEYPAD_0 KEY_FIRST + 41`
- `#define KEY_KEYPAD_1 KEY_FIRST + 42`
- `#define KEY_KEYPAD_2 KEY_FIRST + 43`
- `#define KEY_KEYPAD_3 KEY_FIRST + 44`
- `#define KEY_KEYPAD_4 KEY_FIRST + 45`

- #define [KEY\\_KEYPAD\\_5](#) [KEY\\_FIRST](#) + 46
- #define [KEY\\_KEYPAD\\_6](#) [KEY\\_FIRST](#) + 47
- #define [KEY\\_KEYPAD\\_7](#) [KEY\\_FIRST](#) + 48
- #define [KEY\\_KEYPAD\\_8](#) [KEY\\_FIRST](#) + 49
- #define [KEY\\_KEYPAD\\_9](#) [KEY\\_FIRST](#) + 50
- #define [KEY\\_BACKSPACE](#) [KEY\\_FIRST](#) + 51
- #define [KEY\\_TAB](#) [KEY\\_FIRST](#) + 52
- #define [KEY\\_DELETE](#) [KEY\\_FIRST](#) + 53
- #define [KEY\\_ESCAPE](#) [KEY\\_FIRST](#) + 54
- #define [KEY\\_LAST](#) [KEY\\_ESCAPE](#)
- #define [MOUSE\\_BUTTONUP](#) 0
- #define [MOUSE\\_BUTTONDOWN](#) 1
- #define [MOUSE\\_LEFTBUTTON](#) 0
- #define [MOUSE\\_RIGHTBUTTON](#) 1
- #define [MOUSE\\_MIDDLEBUTTON](#) 2
- #define [MOUSE\\_LAST](#) [MOUSE\\_MIDDLEBUTTON](#) + 1
- #define [MOUSE\\_SCROLL\\_DOWN](#) 0
- #define [MOUSE\\_SCROLL\\_UP](#) 1
- #define [WINDOWSTYLE\\_BARE](#) 1
- #define [WINDOWSTYLE\\_DEFAULT](#) 2
- #define [WINDOWSTYLE\\_POPUP](#) 3
- #define [WINDOWSTATE\\_NORMAL](#) 0
- #define [WINDOWSTATE\\_MAXIMIZED](#) 1
- #define [WINDOWSTATE\\_MINIMIZED](#) 2
- #define [WINDOWSTATE\\_FULLSCREEN](#) 3
- #define [DECORATOR\\_TITLEBAR](#) 0x01
- #define [DECORATOR\\_ICON](#) 0x02
- #define [DECORATOR\\_BORDER](#) 0x04
- #define [DECORATOR\\_MINIMIZEBUTTON](#) 0x08
- #define [DECORATOR\\_MAXIMIZEBUTTON](#) 0x010
- #define [DECORATOR\\_CLOSEBUTTON](#) 0x20
- #define [DECORATOR\\_VERTICALSCROLLBAR](#) 0x40
- #define [DECORATOR\\_HORIZONTALSCROLLBAR](#) 0x80
- #define [DECORATOR\\_SIZEABLEBORDER](#) 0x100
- #define [LINUX\\_DECORATOR\\_BORDER](#) 1L << 1
- #define [LINUX\\_DECORATOR\\_MOVE](#) 1L << 2
- #define [LINUX\\_DECORATOR\\_MINIMIZE](#) 1L << 3
- #define [LINUX\\_DECORATOR\\_MAXIMIZE](#) 1L << 4
- #define [LINUX\\_DECORATOR\\_CLOSE](#) 1L << 5
- #define [FOUNDATION\\_ERROR](#) 0
- #define [FOUNDATION\\_OKAY](#) 1
- #define [ERROR\\_NOCONTEXT](#) 0
- #define [ERROR\\_INVALIDWINDOWNAME](#) 1
- #define [ERROR\\_INVALIDWINDOWINDEX](#) 2
- #define [ERROR\\_INVALIDWINDOWSTATE](#) 3
- #define [ERROR\\_INVALIDRESOLUTION](#) 4
- #define [ERROR\\_INVALIDCONTEXT](#) 5
- #define [ERROR\\_EXISTINGCONTEXT](#) 6
- #define [ERROR\\_NOTINITIALIZED](#) 7
- #define [ERROR\\_ALREADYINITIALIZED](#) 8
- #define [ERROR\\_INVALIDTITLEBAR](#) 9
- #define [ERROR\\_INVALIDEVENT](#) 10
- #define [ERROR\\_WINDOWNOTFOUND](#) 11
- #define [ERROR\\_INVALIDWINDOWSTYLE](#) 12
- #define [ERROR\\_INVALIDWINDOW](#) 13

- `#define` [ERROR\\_FUNCTIONNOTIMPLEMENTED](#) 14
- `#define` [ERROR\\_LINUX\\_CANNOTCONNECTXSERVER](#) 15
- `#define` [ERROR\\_LINUX\\_INVALIDVISUALINFO](#) 16
- `#define` [ERROR\\_LINUX\\_CANNOTCREATEWINDOW](#) 17
- `#define` [ERROR\\_LINUX\\_FUNCTIONNOTIMPLEMENTED](#) 18
- `#define` [ERROR\\_WINDOWS\\_CANNOTCREATEWINDOW](#) 19
- `#define` [ERROR\\_WINDOWS\\_CANNOTINITIALIZE](#) 20
- `#define` [ERROR\\_WINDOWS\\_FUNCTIONNOTIMPLEMENTED](#) 21
- `#define` [WARNING\\_NOTCURRENTCONTEXT](#) 0
- `#define` [WARNING\\_NOGLEXTEENSIONS](#) 1
- `#define` [LINUX\\_FUNCTION](#) 1
- `#define` [LINUX\\_DECORATOR](#) 2

## Typedefs

- `typedef void(* OnKeyEvent )(GLuint Key, GLboolean KeyState)`
- `typedef void(* OnMouseButtonEvent )(GLuint Button, GLboolean ButtonState)`
- `typedef void(* OnMouseWheelEvent )(GLuint WheelDirection)`
- `typedef void(* OnDestroyedEvent )()`
- `typedef void(* OnMaximizedEvent )()`
- `typedef void(* OnMinimizedEvent )()`
- `typedef void(* OnFocusEvent )(GLboolean InFocus)`
- `typedef void(* OnMovedEvent )(GLuint X, GLuint Y)`
- `typedef void(* OnResizeEvent )(GLuint Width, GLuint Height)`
- `typedef void(* OnMouseMoveEvent )(GLuint WindowX, GLuint WindowY, GLuint ScreenX, GLuint ScreenY)`

## Functions

- static GLboolean [IsValidString](#) (const char \*String)
- static GLboolean [IsValidKeyEvent](#) ([OnKeyEvent](#) OnKeyPressed)
- static GLboolean [IsValidMouseWheelEvent](#) ([OnMouseWheelEvent](#) MouseWheelEvent)
- static GLboolean [IsValidDestroyedEvent](#) ([OnMaximizedEvent](#) OnMaximized)
- static GLboolean [IsValidFocusEvent](#) ([OnFocusEvent](#) OnFocus)
- static GLboolean [IsValidMovedEvent](#) ([OnMovedEvent](#) OnMoved)
- static GLboolean [IsValidMouseMoveEvent](#) ([OnMouseMoveEvent](#) OnMouseMove)
- static void [PrintWarningMessage](#) (GLuint WarningNumber)
- static void [PrintErrorMessage](#) (GLuint ErrorNumber)

### 4.11.1 Macro Definition Documentation

#### 4.11.1.1 `#define` DECORATOR\_BORDER 0x04

Definition at line 119 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.2 `#define` DECORATOR\_CLOSEBUTTON 0x20

Definition at line 122 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.3 `#define` DECORATOR\_HORIZONTALSCROLLBAR 0x80

Definition at line 124 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.4 `#define DECORATOR_ICON 0x02`

Definition at line 118 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.5 `#define DECORATOR_MAXIMIZEBUTTON 0x010`

Definition at line 121 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.6 `#define DECORATOR_MINIMIZEBUTTON 0x08`

Definition at line 120 of file [WindowAPI\\_Defs.h](#).

Referenced by [main\(\)](#).

#### 4.11.1.7 `#define DECORATOR_SIZEABLEBORDER 0x100`

Definition at line 125 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.8 `#define DECORATOR_TITLEBAR 0x01`

Definition at line 117 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.9 `#define DECORATOR_VERTICALSCROLLBAR 0x40`

Definition at line 123 of file [WindowAPI\\_Defs.h](#).

#### 4.11.1.10 `#define ERROR_ALREADYINITIALIZED 8`

Definition at line 144 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.11 `#define ERROR_EXISTINGCONTEXT 6`

Definition at line 142 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.12 `#define ERROR_FUNCTIONNOTIMPLEMENTED 14`

Definition at line 150 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.13 `#define ERROR_INVALIDCONTEXT 5`

Definition at line 141 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.14 `#define ERROR_INVALIDEVENT 10`

Definition at line 146 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#), [FWindow::SetOnDestroyed\(\)](#), [FWindow::SetOnFocus\(\)](#), [FWindow::SetOnMaximized\(\)](#), [FWindow::SetOnMinimized\(\)](#), [FWindow::SetOnMouseButtonEvent\(\)](#), [FWindow::SetOnMouseMove\(\)](#), [FWindow::SetOnMouseWheelEvent\(\)](#), [FWindow::SetOnMoved\(\)](#), and [FWindow::SetOnResize\(\)](#).

#### 4.11.1.15 `#define ERROR_INVALIDRESOLUTION 4`

Definition at line 140 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#), and [FWindow::SetResolution\(\)](#).

#### 4.11.1.16 `#define ERROR_INVALIDTITLEBAR 9`

Definition at line 145 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#), and [FWindow::SetTitleBar\(\)](#).

#### 4.11.1.17 `#define ERROR_INVALIDWINDOW 13`

Definition at line 149 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::AddWindow\(\)](#), and [PrintErrorMessage\(\)](#).

#### 4.11.1.18 `#define ERROR_INVALIDWINDOWINDEX 2`

Definition at line 138 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::DoesExist\(\)](#), [WindowManager::GetMousePositionInWindow\(\)](#), and [PrintErrorMessage\(\)](#).

#### 4.11.1.19 `#define ERROR_INVALIDWINDOWNAME 1`

Definition at line 137 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::DoesExist\(\)](#), [FWindow::FWindow\(\)](#), and [PrintErrorMessage\(\)](#).

#### 4.11.1.20 `#define ERROR_INVALIDWINDOWSTATE 3`

Definition at line 139 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.21 `#define ERROR_INVALIDWINDOWSTYLE 12`

Definition at line 148 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.22 `#define ERROR_LINUX_CANNOTCONNECTXSERVER 15`

Definition at line 151 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.23 `#define ERROR_LINUX_CANNOTCREATEWINDOW 17`

Definition at line 153 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.24 `#define ERROR_LINUX_FUNCTIONNOTIMPLEMENTED 18`

Definition at line 154 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.25 `#define ERROR_LINUX_INVALIDVISUALINFO 16`

Definition at line 152 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.26 `#define ERROR_NOCONTEXT 0`

Definition at line 136 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::DisableDecorator\(\)](#), [FWindow::EnableDecorator\(\)](#), [FWindow::Focus\(\)](#), [FWindow::FullScreen\(\)](#), [FWindow::GetIsCurrentContext\(\)](#), [FWindow::GetIsFullScreen\(\)](#), [FWindow::GetMousePosition\(\)](#), [FWindow::GetOpenGLExtensions\(\)](#), [FWindow::GetOpenGLVersion\(\)](#), [FWindow::GetPosition\(\)](#), [FWindow::GetResolution\(\)](#), [FWindow::GetWindowName\(\)](#), [FWindow::MakeCurrentContext\(\)](#), [FWindow::Maximize\(\)](#), [PrintErrorMessage\(\)](#), [FWindow::PrintOpenGLExtensions\(\)](#), [FWindow::PrintOpenGLVersion\(\)](#), [FWindow::Restore\(\)](#), [FWindow::SetCurrentState\(\)](#), [FWindow::SetMousePosition\(\)](#), [FWindow::SetPosition\(\)](#), [FWindow::SetResolution\(\)](#), [FWindow::SetStyle\(\)](#), [FWindow::SetSwapInterval\(\)](#), [FWindow::SetTitleBar\(\)](#), [FWindow::Shutdown\(\)](#), and [FWindow::SwapDrawBuffers\(\)](#).

#### 4.11.1.27 `#define ERROR_NOTINITIALIZED 7`

Definition at line 143 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::AddWindow\(\)](#), [WindowManager::GetMousePositionInScreen\(\)](#), [WindowManager::GetNumWindows\(\)](#), [WindowManager::GetScreenResolution\(\)](#), [WindowManager::GetWindowResolution\(\)](#), [WindowManager::PollForEvents\(\)](#), and [PrintErrorMessage\(\)](#).

#### 4.11.1.28 `#define ERROR_WINDOWNOTFOUND 11`

Definition at line 147 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::GetWindowByIndex\(\)](#), [WindowManager::GetWindowByName\(\)](#), and [PrintErrorMessage\(\)](#).

#### 4.11.1.29 `#define ERROR_WINDOWS_CANNOTCREATEWINDOW 19`

Definition at line 155 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

#### 4.11.1.30 `#define ERROR_WINDOWS_CANNOTINITIALIZE 20`

Definition at line 156 of file [WindowAPI\\_Defs.h](#).

4.11.1.31 `#define ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED 21`

Definition at line 157 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintErrorMessage\(\)](#).

4.11.1.32 `#define FOUNDATION_ERROR 0`

Definition at line 133 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::DisableDecorator\(\)](#), [WindowManager::DisableWindowDecorator\(\)](#), [WindowManager::DoesExist\(\)](#), [FWindow::EnableDecorator\(\)](#), [WindowManager::EnableWindowDecorator\(\)](#), [FWindow::Focus\(\)](#), [WindowManager::FocusWindow\(\)](#), [FWindow::GetIsFullScreen\(\)](#), [FWindow::GetMousePosition\(\)](#), [WindowManager::GetMousePositionInScreen\(\)](#), [WindowManager::GetMousePositionInWindow\(\)](#), [WindowManager::GetNumWindows\(\)](#), [FWindow::GetPosition\(\)](#), [FWindow::GetResolution\(\)](#), [WindowManager::GetScreenResolution\(\)](#), [WindowManager::GetWindowByIndex\(\)](#), [WindowManager::GetWindowIsFullScreen\(\)](#), [WindowManager::GetWindowIsInFocus\(\)](#), [WindowManager::GetWindowIsMaximized\(\)](#), [WindowManager::GetWindowIsMinimized\(\)](#), [WindowManager::GetWindowResolution\(\)](#), [WindowManager::GetWindowShouldClose\(\)](#), [FWindow::MakeCurrentContext\(\)](#), [FWindow::Maximize\(\)](#), [WindowManager::MaximizeWindow\(\)](#), [FWindow::Minimize\(\)](#), [WindowManager::MinimizeWindow\(\)](#), [WindowManager::PollForEvents\(\)](#), [FWindow::PrintOpenGLExtensions\(\)](#), [FWindow::PrintOpenGLVersion\(\)](#), [FWindow::Restore\(\)](#), [WindowManager::RestoreWindow\(\)](#), [FWindow::SetCurrentState\(\)](#), [WindowManager::SetFullScreen\(\)](#), [FWindow::SetIcon\(\)](#), [FWindow::SetMousePosition\(\)](#), [WindowManager::SetMousePositionInWindow\(\)](#), [FWindow::SetOnDestroyed\(\)](#), [FWindow::SetOnFocus\(\)](#), [FWindow::SetOnKeyEvent\(\)](#), [FWindow::SetOnMaximized\(\)](#), [FWindow::SetOnMinimized\(\)](#), [FWindow::SetOnMouseButtonEvent\(\)](#), [FWindow::SetOnMouseMove\(\)](#), [FWindow::SetOnMouseWheelEvent\(\)](#), [FWindow::SetOnMoved\(\)](#), [FWindow::SetOnResize\(\)](#), [FWindow::SetPosition\(\)](#), [FWindow::SetResolution\(\)](#), [FWindow::SetStyle\(\)](#), [FWindow::SetSwapInterval\(\)](#), [FWindow::SetTitleBar\(\)](#), [WindowManager::SetWindowOnDestroyed\(\)](#), [WindowManager::SetWindowOnFocus\(\)](#), [WindowManager::SetWindowOnKeyEvent\(\)](#), [WindowManager::SetWindowOnMaximized\(\)](#), [WindowManager::SetWindowOnMinimized\(\)](#), [WindowManager::SetWindowOnMouseButtonEvent\(\)](#), [WindowManager::SetWindowOnMouseMove\(\)](#), [WindowManager::SetWindowOnMouseWheelEvent\(\)](#), [WindowManager::SetWindowOnMoved\(\)](#), [WindowManager::SetWindowOnResize\(\)](#), [WindowManager::SetWindowPosition\(\)](#), [WindowManager::SetWindowStyle\(\)](#), [WindowManager::SetWindowSwapInterval\(\)](#), [WindowManager::SetWindowTitleBar\(\)](#), [FWindow::Shutdown\(\)](#), [FWindow::SwapDrawBuffers\(\)](#), [WindowManager::WindowGetKey\(\)](#), and [WindowManager::WindowSwapBuffers\(\)](#).

4.11.1.33 `#define FOUNDATION_OKAY 1`

Definition at line 134 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::DisableDecorator\(\)](#), [WindowManager::DoesExist\(\)](#), [FWindow::EnableDecorator\(\)](#), [FWindow::Focus\(\)](#), [FWindow::FullScreen\(\)](#), [FWindow::GetMousePosition\(\)](#), [WindowManager::GetMousePositionInScreen\(\)](#), [FWindow::GetPosition\(\)](#), [FWindow::GetResolution\(\)](#), [WindowManager::GetScreenResolution\(\)](#), [WindowManager::GetWindowResolution\(\)](#), [FWindow::MakeCurrentContext\(\)](#), [FWindow::Maximize\(\)](#), [FWindow::Minimize\(\)](#), [FWindow::PrintOpenGLExtensions\(\)](#), [FWindow::PrintOpenGLVersion\(\)](#), [FWindow::Restore\(\)](#), [FWindow::SetIcon\(\)](#), [FWindow::SetMousePosition\(\)](#), [FWindow::SetOnDestroyed\(\)](#), [FWindow::SetOnFocus\(\)](#), [FWindow::SetOnKeyEvent\(\)](#), [FWindow::SetOnMaximized\(\)](#), [FWindow::SetOnMinimized\(\)](#), [FWindow::SetOnMouseButtonEvent\(\)](#), [FWindow::SetOnMouseMove\(\)](#), [FWindow::SetOnMouseWheelEvent\(\)](#), [FWindow::SetOnMoved\(\)](#), [FWindow::SetOnResize\(\)](#), [FWindow::SetResolution\(\)](#), [FWindow::SetStyle\(\)](#), [FWindow::SetSwapInterval\(\)](#), [FWindow::SetTitleBar\(\)](#), [WindowManager::SetWindowOnFocus\(\)](#), [FWindow::Shutdown\(\)](#), and [FWindow::SwapDrawBuffers\(\)](#).

4.11.1.34 `#define KEY_ARROW_DOWN KEY_FIRST + 31`

the ArrowDown key

Definition at line 71 of file [WindowAPI\\_Defs.h](#).

4.11.1.35 `#define KEY_ARROW_LEFT KEY_FIRST + 33`

the ArrowLeft key

Definition at line 73 of file [WindowAPI\\_Defs.h](#).

4.11.1.36 `#define KEY_ARROW_RIGHT KEY_FIRST + 34`

the ArrowRight key

Definition at line 74 of file [WindowAPI\\_Defs.h](#).

4.11.1.37 `#define KEY_ARROW_UP KEY_FIRST + 32`

the ArrowUp key

Definition at line 72 of file [WindowAPI\\_Defs.h](#).

4.11.1.38 `#define KEY_BACKSPACE KEY_FIRST + 51`

the Backspace key

Definition at line 91 of file [WindowAPI\\_Defs.h](#).

4.11.1.39 `#define KEY_CAPSLOCK KEY_FIRST + 12`

the CapsLock key

Definition at line 52 of file [WindowAPI\\_Defs.h](#).

4.11.1.40 `#define KEY_DELETE KEY_FIRST + 53`

the Delete key

Definition at line 93 of file [WindowAPI\\_Defs.h](#).

4.11.1.41 `#define KEY_END KEY_FIRST + 28`

the End key

Definition at line 68 of file [WindowAPI\\_Defs.h](#).

4.11.1.42 `#define KEY_ENTER KEY_FIRST + 21`

the Enter/Return key

Definition at line 61 of file [WindowAPI\\_Defs.h](#).

4.11.1.43 `#define KEY_ERROR -1`

the key pressed is considered invalid

Definition at line 37 of file [WindowAPI\\_Defs.h](#).



4.11.1.44 `#define KEY_ESCAPE KEY_FIRST + 54`

the Escape key

Definition at line 94 of file [WindowAPI\\_Defs.h](#).

4.11.1.45 `#define KEY_F1 KEY_FIRST`

the F1 key

Definition at line 40 of file [WindowAPI\\_Defs.h](#).

4.11.1.46 `#define KEY_F10 KEY_FIRST + 9`

the F10 key

Definition at line 49 of file [WindowAPI\\_Defs.h](#).

4.11.1.47 `#define KEY_F11 KEY_FIRST + 10`

the F11 key

Definition at line 50 of file [WindowAPI\\_Defs.h](#).

4.11.1.48 `#define KEY_F12 KEY_FIRST + 11`

the F12 key

Definition at line 51 of file [WindowAPI\\_Defs.h](#).

4.11.1.49 `#define KEY_F2 KEY_FIRST + 1`

the F2 key

Definition at line 41 of file [WindowAPI\\_Defs.h](#).

4.11.1.50 `#define KEY_F3 KEY_FIRST + 2`

the F3 key

Definition at line 42 of file [WindowAPI\\_Defs.h](#).

4.11.1.51 `#define KEY_F4 KEY_FIRST + 3`

the F4 key

Definition at line 43 of file [WindowAPI\\_Defs.h](#).

4.11.1.52 `#define KEY_F5 KEY_FIRST + 4`

the F5 key

Definition at line 44 of file [WindowAPI\\_Defs.h](#).

4.11.1.53 `#define KEY_F6 KEY_FIRST + 5`

the F6 key

Definition at line 45 of file [WindowAPI\\_Defs.h](#).

4.11.1.54 `#define KEY_F7 KEY_FIRST + 6`

the F7 key

Definition at line 46 of file [WindowAPI\\_Defs.h](#).

4.11.1.55 `#define KEY_F8 KEY_FIRST + 7`

the F8 key

Definition at line 47 of file [WindowAPI\\_Defs.h](#).

4.11.1.56 `#define KEY_F9 KEY_FIRST + 8`

the F9 key

Definition at line 48 of file [WindowAPI\\_Defs.h](#).

4.11.1.57 `#define KEY_FIRST 256 + 1`

the first key that is not a char

Definition at line 39 of file [WindowAPI\\_Defs.h](#).

4.11.1.58 `#define KEY_HOME KEY_FIRST + 27`

the Home key

Definition at line 67 of file [WindowAPI\\_Defs.h](#).

4.11.1.59 `#define KEY_INSERT KEY_FIRST + 26`

the insert key

Definition at line 66 of file [WindowAPI\\_Defs.h](#).

4.11.1.60 `#define KEY_KEYPAD_0 KEY_FIRST + 41`

the Keypad 0 key

Definition at line 81 of file [WindowAPI\\_Defs.h](#).

4.11.1.61 `#define KEY_KEYPAD_1 KEY_FIRST + 42`

the Keypad 1 key

Definition at line 82 of file [WindowAPI\\_Defs.h](#).

4.11.1.62 `#define KEY_KEYPAD_2 KEY_FIRST + 43`

the Keypad 2 key

Definition at line 83 of file [WindowAPI\\_Defs.h](#).

4.11.1.63 `#define KEY_KEYPAD_3 KEY_FIRST + 44`

the Keypad 3 key

Definition at line 84 of file [WindowAPI\\_Defs.h](#).

4.11.1.64 `#define KEY_KEYPAD_4 KEY_FIRST + 45`

the Keypad 4 key

Definition at line 85 of file [WindowAPI\\_Defs.h](#).

4.11.1.65 `#define KEY_KEYPAD_5 KEY_FIRST + 46`

the Keypad 5 key

Definition at line 86 of file [WindowAPI\\_Defs.h](#).

4.11.1.66 `#define KEY_KEYPAD_6 KEY_FIRST + 47`

the Keypad 6 key

Definition at line 87 of file [WindowAPI\\_Defs.h](#).

4.11.1.67 `#define KEY_KEYPAD_7 KEY_FIRST + 48`

the Keypad 7 key

Definition at line 88 of file [WindowAPI\\_Defs.h](#).

4.11.1.68 `#define KEY_KEYPAD_8 KEY_FIRST + 49`

the keypad 8 key

Definition at line 89 of file [WindowAPI\\_Defs.h](#).

4.11.1.69 `#define KEY_KEYPAD_9 KEY_FIRST + 50`

the Keypad 9 key

Definition at line 90 of file [WindowAPI\\_Defs.h](#).

4.11.1.70 `#define KEY_KEYPAD_ADD KEY_FIRST + 38`

the Keypad Add key

Definition at line 78 of file [WindowAPI\\_Defs.h](#).

4.11.1.71 `#define KEY_KEYPAD_DIVIDE KEY_FIRST + 35`

the KeyPad Divide key

Definition at line 75 of file [WindowAPI\\_Defs.h](#).

4.11.1.72 `#define KEY_KEYPAD_ENTER KEY_FIRST + 39`

the Keypad Enter key

Definition at line 79 of file [WindowAPI\\_Defs.h](#).

4.11.1.73 `#define KEY_KEYPAD_MULTIPLY KEY_FIRST + 36`

the Keypad Multiply key

Definition at line 76 of file [WindowAPI\\_Defs.h](#).

4.11.1.74 `#define KEY_KEYPAD_PERIOD KEY_FIRST + 40`

the Keypad Period/Decimal key

Definition at line 80 of file [WindowAPI\\_Defs.h](#).

4.11.1.75 `#define KEY_KEYPAD_SUBTRACT KEY_FIRST + 37`

the Keypad Subtract key

Definition at line 77 of file [WindowAPI\\_Defs.h](#).

4.11.1.76 `#define KEY_LAST KEY_ESCAPE`

the last key to be supported

Definition at line 95 of file [WindowAPI\\_Defs.h](#).

4.11.1.77 `#define KEY_LEFTALT KEY_FIRST + 19`

the left Alternate key

Definition at line 59 of file [WindowAPI\\_Defs.h](#).

4.11.1.78 `#define KEY_LEFTCONTROL KEY_FIRST + 15`

the left Control key

Definition at line 55 of file [WindowAPI\\_Defs.h](#).

4.11.1.79 `#define KEY_LEFTSHIFT KEY_FIRST + 13`

the left Shift key

Definition at line 53 of file [WindowAPI\\_Defs.h](#).

4.11.1.80 `#define KEY_LEFTWINDOW KEY_FIRST + 17`

the left Window key

Definition at line 57 of file [WindowAPI\\_Defs.h](#).

4.11.1.81 `#define KEY_NUMLOCK KEY_FIRST + 24`

the NumLock key

Definition at line 64 of file [WindowAPI\\_Defs.h](#).

4.11.1.82 `#define KEY_PAGEDOWN KEY_FIRST + 30`

the PageDown key

Definition at line 70 of file [WindowAPI\\_Defs.h](#).

4.11.1.83 `#define KEY_PAGEUP KEY_FIRST + 28`

the PageUp key

Definition at line 69 of file [WindowAPI\\_Defs.h](#).

4.11.1.84 `#define KEY_PAUSE KEY_FIRST + 25`

the pause/break key

Definition at line 65 of file [WindowAPI\\_Defs.h](#).

4.11.1.85 `#define KEY_PRINTSCREEN KEY_FIRST + 22`

the PrintScreen key

Definition at line 62 of file [WindowAPI\\_Defs.h](#).

4.11.1.86 `#define KEY_RIGHTALT KEY_FIRST + 20`

the right Alternate key

Definition at line 60 of file [WindowAPI\\_Defs.h](#).

4.11.1.87 `#define KEY_RIGHTCONTROL KEY_FIRST + 16`

the right Control key

Definition at line 56 of file [WindowAPI\\_Defs.h](#).

4.11.1.88 `#define KEY_RIGHTSHIFT KEY_FIRST + 14`

the right Shift key

Definition at line 54 of file [WindowAPI\\_Defs.h](#).

4.11.1.89 `#define KEY_RIGHTWINDOW KEY_FIRST + 18`

the right Window key

Definition at line 58 of file [WindowAPI\\_Defs.h](#).

4.11.1.90 `#define KEY_SCROLLLOCK KEY_FIRST + 23`

the ScrollLock key

Definition at line 63 of file [WindowAPI\\_Defs.h](#).

4.11.1.91 `#define KEY_TAB KEY_FIRST + 52`

the Tab key

Definition at line 92 of file [WindowAPI\\_Defs.h](#).

4.11.1.92 `#define KEYSTATE_DOWN 1`

the key is currently up

Definition at line 34 of file [WindowAPI\\_Defs.h](#).

Referenced by [OnWindowKeyPressed\(\)](#).

4.11.1.93 `#define KEYSTATE_UP 0`

the key is currently down

Definition at line 35 of file [WindowAPI\\_Defs.h](#).

4.11.1.94 `#define LINUX_DECORATOR 2`

Definition at line 163 of file [WindowAPI\\_Defs.h](#).

4.11.1.95 `#define LINUX_DECORATOR_BORDER 1L << 1`

Definition at line 127 of file [WindowAPI\\_Defs.h](#).

4.11.1.96 `#define LINUX_DECORATOR_CLOSE 1L << 5`

Definition at line 131 of file [WindowAPI\\_Defs.h](#).

4.11.1.97 `#define LINUX_DECORATOR_MAXIMIZE 1L << 4`

Definition at line 130 of file [WindowAPI\\_Defs.h](#).

4.11.1.98 `#define LINUX_DECORATOR_MINIMIZE 1L << 3`

Definition at line 129 of file [WindowAPI\\_Defs.h](#).

4.11.1.99 `#define LINUX_DECORATOR_MOVE 1L << 2`

Definition at line 128 of file [WindowAPI\\_Defs.h](#).

4.11.1.100 `#define LINUX_FUNCTION 1`

Definition at line 162 of file [WindowAPI\\_Defs.h](#).

4.11.1.101 `#define MOUSE_BUTTONDOWN 1`

the mouse button is currently down

Definition at line 98 of file [WindowAPI\\_Defs.h](#).

4.11.1.102 `#define MOUSE_BUTTONUP 0`

the mouse button is currently up

Definition at line 97 of file [WindowAPI\\_Defs.h](#).

4.11.1.103 `#define MOUSE_LAST MOUSE_MIDDLEBUTTON + 1`

the last mouse button to be supported

Definition at line 103 of file [WindowAPI\\_Defs.h](#).

4.11.1.104 `#define MOUSE_LEFTBUTTON 0`

the left mouse button

Definition at line 100 of file [WindowAPI\\_Defs.h](#).

4.11.1.105 `#define MOUSE_MIDDLEBUTTON 2`

the middle mouse button / ScrollWheel

Definition at line 102 of file [WindowAPI\\_Defs.h](#).

4.11.1.106 `#define MOUSE_RIGHTBUTTON 1`

the right mouse button

Definition at line 101 of file [WindowAPI\\_Defs.h](#).

4.11.1.107 `#define MOUSE_SCROLL_DOWN 0`

the mouse wheel up

Definition at line 105 of file [WindowAPI\\_Defs.h](#).

4.11.1.108 `#define MOUSE_SCROLL_UP 1`

the mouse wheel down

Definition at line 106 of file [WindowAPI\\_Defs.h](#).

**4.11.1.109 #define WARNING\_NOGLEXTEENSIONS 1**

Definition at line 160 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintWarningMessage\(\)](#).

**4.11.1.110 #define WARNING\_NOTCURRENTCONTEXT 0**

Definition at line 159 of file [WindowAPI\\_Defs.h](#).

Referenced by [PrintWarningMessage\(\)](#).

**4.11.1.111 #define WINDOWSTATE\_FULLSCREEN 3**

the window is currently full screen

Definition at line 115 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::FullScreen\(\)](#), [FWindow::GetIsFullScreen\(\)](#), [FWindow::Restore\(\)](#), and [FWindow::SetCurrentState\(\)](#).

**4.11.1.112 #define WINDOWSTATE\_MAXIMIZED 1**

the window is currently maximized

Definition at line 113 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::GetIsMaximized\(\)](#), [FWindow::Maximize\(\)](#), [FWindow::Restore\(\)](#), and [FWindow::SetCurrentState\(\)](#).

**4.11.1.113 #define WINDOWSTATE\_MINIMIZED 2**

the window is currently minimized

Definition at line 114 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::GetIsMinimized\(\)](#), [FWindow::Minimize\(\)](#), and [FWindow::SetCurrentState\(\)](#).

**4.11.1.114 #define WINDOWSTATE\_NORMAL 0**

the window is in its default state

Definition at line 112 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::FullScreen\(\)](#), [FWindow::FWindow\(\)](#), [FWindow::Maximize\(\)](#), [FWindow::Minimize\(\)](#), and [FWindow::Restore\(\)](#).

**4.11.1.115 #define WINDOWSTYLE\_BARE 1**

the window has no decorators but the window border and title bar

Definition at line 108 of file [WindowAPI\\_Defs.h](#).

Referenced by [main\(\)](#).

**4.11.1.116 #define WINDOWSTYLE\_DEFAULT 2**

the default window style for the respective platform

Definition at line 109 of file [WindowAPI\\_Defs.h](#).



#### 4.11.1.117 `#define WINDOWSTYLE_POPUP 3`

the window has no decorators

Definition at line 110 of file [WindowAPI\\_Defs.h](#).

### 4.11.2 Typedef Documentation

#### 4.11.2.1 `typedef void(* OnDestroyedEvent)()`

To be called when the window is being destroyed

Definition at line 170 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.2 `typedef void(* OnFocusEvent)(GLboolean InFocus)`

To be called when the window has gained event focus

Definition at line 174 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.3 `typedef void(* OnKeyEvent)(GLuint Key, GLboolean KeyState)`

To be called when a key event has occurred

Definition at line 167 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.4 `typedef void(* OnMaximizedEvent)()`

To be called when the window has been maximized

Definition at line 171 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.5 `typedef void(* OnMinimizedEvent)()`

To be called when the window has been minimized

Definition at line 172 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.6 `typedef void(* OnMouseButtonEvent)(GLuint Button, GLboolean ButtonState)`

To be called when a Mouse button event has occurred

Definition at line 168 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.7 `typedef void(* OnMouseMoveEvent)(GLuint WindowX, GLuint WindowY, GLuint ScreenX, GLuint ScreenY)`

To be called when the mouse has been moved within the window

Definition at line 177 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.8 `typedef void(* OnMouseWheelEvent)(GLuint WheelDirection)`

To be called when a mouse wheel event has occurred.

Definition at line 169 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.9 typedef void(\* OnMovedEvent)(GLuint X, GLuint Y)

To be called when the window has been moved

Definition at line 175 of file [WindowAPI\\_Defs.h](#).

#### 4.11.2.10 typedef void(\* OnResizeEvent)(GLuint Width, GLuint Height)

To be called when the window has been resized

Definition at line 176 of file [WindowAPI\\_Defs.h](#).

### 4.11.3 Function Documentation

#### 4.11.3.1 static GLboolean IsValidDestroyedEvent ( OnMaximizedEvent OnMaximized ) [inline],[static]

Definition at line 196 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnDestroyed\(\)](#), [FWindow::SetOnMaximized\(\)](#), and [FWindow::SetOnMinimized\(\)](#).

```
00197 {
00198     return (OnMaximized != nullptr);
00199 }
```

#### 4.11.3.2 static GLboolean IsValidFocusEvent ( OnFocusEvent OnFocus ) [inline],[static]

Definition at line 201 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnFocus\(\)](#).

```
00202 {
00203     return (OnFocus != nullptr);
00204 }
```

#### 4.11.3.3 static GLboolean IsValidKeyEvent ( OnKeyEvent OnKeyPressed ) [inline],[static]

Definition at line 186 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnKeyEvent\(\)](#), and [FWindow::SetOnMouseButtonEvent\(\)](#).

```
00187 {
00188     return (OnKeyPressed != nullptr);
00189 }
```

#### 4.11.3.4 static GLboolean IsValidMouseMoveEvent ( OnMouseMoveEvent OnMouseMove ) [inline],[static]

Definition at line 211 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnMouseMove\(\)](#).

```
00212 {
00213     return (OnMouseMove != nullptr);
00214 }
```

**4.11.3.5** static GLboolean IsValidMouseEvent ( *OnMouseEvent MouseEvent* ) [inline],  
[static]

Definition at line 191 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnMouseEvent\(\)](#).

```
00192 {
00193     return (MouseEvent != nullptr);
00194 }
```

**4.11.3.6** static GLboolean IsValidMovedEvent ( *OnMovedEvent OnMoved* ) [inline],[static]

Definition at line 206 of file [WindowAPI\\_Defs.h](#).

Referenced by [FWindow::SetOnMoved\(\)](#), and [FWindow::SetOnResize\(\)](#).

```
00207 {
00208     return (OnMoved != nullptr);
00209 }
```

**4.11.3.7** static GLboolean IsValidString ( const char \* *String* ) [inline],[static]

Definition at line 180 of file [WindowAPI\\_Defs.h](#).

Referenced by [WindowManager::DoesExist\(\)](#), [FWindow::FWindow\(\)](#), [WindowManager::SetWindowOnFocus\(\)](#), and [WindowManager::SetWindowTitleBar\(\)](#).

```
00181 {
00182     return (String != nullptr);
00183 }
```

**4.11.3.8** static void PrintErrorMessage ( GLuint *ErrorNumber* ) [static]

Definition at line 242 of file [WindowAPI\\_Defs.h](#).

References [ERROR\\_ALREADYINITIALIZED](#), [ERROR\\_EXISTINGCONTEXT](#), [ERROR\\_FUNCTIONNOTIMPLEMENTED](#), [ERROR\\_INVALIDCONTEXT](#), [ERROR\\_INVALIDEVENT](#), [ERROR\\_INVALIDRESOLUTION](#), [ERROR\\_INVALIDTITLEBAR](#), [ERROR\\_INVALIDWINDOW](#), [ERROR\\_INVALIDWINDOWINDEX](#), [ERROR\\_INVALIDWINDOWNAME](#), [ERROR\\_INVALIDWINDOWSTATE](#), [ERROR\\_INVALIDWINDOWSTYLE](#), [ERROR\\_LINUX\\_CANNOTCONNECTXSERVER](#), [ERROR\\_LINUX\\_CANNOTCREATEWINDOW](#), [ERROR\\_LINUX\\_FUNCTIONNOTIMPLEMENTED](#), [ERROR\\_LINUX\\_INVALIDVISUALINFO](#), [ERROR\\_NOCONTEXT](#), [ERROR\\_NOTINITIALIZED](#), [ERROR\\_WINDOWNOTFOUND](#), [ERROR\\_WINDOWS\\_CANNOTCREATEWINDOW](#), and [ERROR\\_WINDOWS\\_FUNCTIONNOTIMPLEMENTED](#).

Referenced by [WindowManager::AddWindow\(\)](#), [FWindow::DisableDecorator\(\)](#), [WindowManager::DoesExist\(\)](#), [FWindow::EnableDecorator\(\)](#), [FWindow::Focus\(\)](#), [FWindow::FullScreen\(\)](#), [FWindow::FWindow\(\)](#), [FWindow::GetCurrentContext\(\)](#), [FWindow::GetIsFullScreen\(\)](#), [FWindow::GetMousePosition\(\)](#), [WindowManager::GetMousePositionInScreen\(\)](#), [WindowManager::GetMousePositionInWindow\(\)](#), [WindowManager::GetNumWindows\(\)](#), [FWindow::GetOpenGLExtensions\(\)](#), [FWindow::GetOpenGLVersion\(\)](#), [FWindow::GetPosition\(\)](#), [FWindow::GetResolution\(\)](#), [WindowManager::GetScreenResolution\(\)](#), [WindowManager::GetWindowByIndex\(\)](#), [WindowManager::GetWindowByName\(\)](#), [FWindow::GetWindowName\(\)](#), [WindowManager::GetWindowResolution\(\)](#), [FWindow::MakeCurrentContext\(\)](#), [FWindow::Maximize\(\)](#), [WindowManager::PollForEvents\(\)](#), [FWindow::PrintOpenGLExtensions\(\)](#), [FWindow::PrintOpenGLVersion\(\)](#), [FWindow::Restore\(\)](#), [FWindow::SetCurrentState\(\)](#), [FWindow::SetMousePosition\(\)](#), [FWindow::SetOnDestroyed\(\)](#), [FWindow::SetOnFocus\(\)](#), [FWindow::SetOnMaximized\(\)](#), [FWindow::SetOnMinimized\(\)](#), [FWindow::SetOnMouseButtonEvent\(\)](#), [FWindow::SetOnMouseMove\(\)](#), [FWindow::SetOnMouseWheelEvent\(\)](#), [FWindow::SetOnMoved\(\)](#), [FWindow::SetOnResize\(\)](#), [FWindow::SetPosition\(\)](#), [FWindow::SetResolution\(\)](#), [FWindow::SetStyle\(\)](#), [FWindow::SetSwapInterval\(\)](#), [FWindow::SetTitleBar\(\)](#), [FWindow::Shutdown\(\)](#), and [FWindow::SwapDrawBuffers\(\)](#).

```
00243 {
00244     switch(ErrorNumber)
00245     {
00246         case ERROR_NOCONTEXT:
00247         {
00248             printf("Error: An OpenGL context must first be created(initialize the window) \n");
00249             break;
00250         }
00251         case ERROR_INVALIDWINDOWNAME:
00252         {
00253             printf("Error: invald window name \n");
00254             break;
00255         }
00256         case ERROR_INVALIDWINDOWINDEX:
00257         {
00258             printf("Error: invalid window index \n");
00259             break;
00260         }
00261         case ERROR_INVALIDWINDOWSTATE:
00262         {
00263             printf("Error: invalid window state \n");
00264             break;
00265         }
00266         case ERROR_INVALIDRESOLUTION:
00267         {
00268             printf("Error: invalid resolution \n");
00269             break;
00270         }
00271         case ERROR_INVALIDCONTEXT:
00272         {
00273             printf("Error: Failed to create OpenGL context \n");
00274             break;
00275         }
00276         case ERROR_EXISTINGCONTEXT:
00277         {
00278             printf("Error: context already created \n");
00279             break;
00280         }
00281         case ERROR_NOTINITIALIZED:
00282         {
00283             printf("Error: Window manager not initialized \n");
00284             break;
00285         }
00286         case ERROR_ALREADYINITIALIZED:
00287         {
00288             printf("Error: window has already been initialized \n");
00289             break;
00290         }
00291         case ERROR_INVALIDTITLEBAR:
00292         {
00293             printf("Error: invalid title bar name (cannot be null or nullptr) \n");
00294             break;
00295         }
00296         case ERROR_INVALIDEVENT:
00297         {
00298             printf("Error: invalid event callback given \n");
00299             break;
00300         }
00301         case ERROR_WINDOWNOTFOUND:
00302         {
00303             printf("Error: window was not found \n");
00304             break;
00305         }
00306         case ERROR_INVALIDWINDOWSTYLE:
00307         {
00308             printf("Error: invalid window style given \n");
00309             break;
00310         }
00311         case ERROR_INVALIDWINDOW:
00312         {
00313             printf("Error: invalid window given \n");
00314             break;
00315         }
00316     }
00317 }
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
```

```

00330         case ERROR_FUNCTIONNOTIMPLEMENTED:
00331         {
00332             printf("Error: I'm sorry but this function has not been implemented yet :( \n");
00333             break;
00334         }
00335
00336         case ERROR_LINUX_CANNOTCONNECTXSERVER:
00337         {
00338             printf("Error: cannot connect to X server \n");
00339             break;
00340         }
00341
00342         case ERROR_LINUX_INVALIDVISUALINFO:
00343         {
00344             printf("Error: Invalid visual information given \n");
00345             break;
00346         }
00347
00348         case ERROR_LINUX_CANNOTCREATEWINDOW:
00349         {
00350             printf("Error: failed to create window \n");
00351             break;
00352         }
00353
00354         case ERROR_LINUX_FUNCTIONNOTIMPLEMENTED:
00355         {
00356             printf("Error: function not implemented on linux platform yet. sorry :( \n");
00357             break;
00358         }
00359
00360         case ERROR_WINDOWS_CANNOTCREATEWINDOW:
00361         {
00362             printf("Error: failed to create window \n");
00363             break;
00364         }
00365
00366         case ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED:
00367         {
00368             printf("Error: function not implemented on Windows platform yet. sorry ;( \n");
00369             break;
00370         }
00371
00372         default:
00373         {
00374             printf("Error: unspecified Error \n");
00375             break;
00376         }
00377     }
00378 }

```

#### 4.11.3.9 static void PrintWarningMessage ( GLuint WarningNumber ) [inline],[static]

Definition at line 217 of file [WindowAPI\\_Defs.h](#).

References [WARNING\\_NOGLEXTENSIONS](#), and [WARNING\\_NOTCURRENTCONTEXT](#).

```

00218 {
00219     switch(WarningNumber)
00220     {
00221         case WARNING_NOGLEXTENSIONS:
00222         {
00223             printf("Warning: no OpenGL extensions available \n");
00224             break;
00225         }
00226
00227         case WARNING_NOTCURRENTCONTEXT:
00228         {
00229             printf("Warning: window not the current OpenGL context being rendered to \n");
00230             break;
00231         }
00232
00233         default:
00234         {
00235             printf("Warning: unspecified warning \n");
00236             break;
00237         }
00238     }
00239 }

```

## 4.12 WindowAPI\_Defs.h

```

00001 /*****
00006 #ifndef WINDOWAPI_DEFS_H
00007 #define WINDOWAPI_DEFS_H
00008
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <list>
00012
00013 #if defined(_MSC_VER) || defined(_WIN32) || defined(_WIN64)
00014 #define CURRENT_OS_WINDOWS
00015 #include <windows.h>
00016 #include <gl/GL.h>
00017 #include "../dependencies/wglext.h"
00018 #endif
00019
00020 #if defined(__linux__) || defined(__GNUCC__) || defined(__GNUC__) || defined(__clang__)
00021 #define CURRENT_OS_LINUX
00022 #include <GL/gl.h>
00023 #include "../dependencies/glxext.h"
00024 #include <GL/glxext.h>
00025 #include <GL/gl.h>
00026 #include <GL/glu.h>
00027 #include <X11/X.h>
00028 #include <X11/Xlib.h>
00029 #include <X11/keysym.h>
00030 #include <X11/Xatom.h>
00031 #include <string>
00032 #endif
00033
00034 #define KEYSTATE_DOWN 1
00035 #define KEYSTATE_UP 0
00037 #define KEY_ERROR -1
00039 #define KEY_FIRST 256 + 1
00040 #define KEY_F1 KEY_FIRST
00041 #define KEY_F2 KEY_FIRST + 1
00042 #define KEY_F3 KEY_FIRST + 2
00043 #define KEY_F4 KEY_FIRST + 3
00044 #define KEY_F5 KEY_FIRST + 4
00045 #define KEY_F6 KEY_FIRST + 5
00046 #define KEY_F7 KEY_FIRST + 6
00047 #define KEY_F8 KEY_FIRST + 7
00048 #define KEY_F9 KEY_FIRST + 8
00049 #define KEY_F10 KEY_FIRST + 9
00050 #define KEY_F11 KEY_FIRST + 10
00051 #define KEY_F12 KEY_FIRST + 11
00052 #define KEY_CAPSLOCK KEY_FIRST + 12
00053 #define KEY_LEFTSHIFT KEY_FIRST + 13
00054 #define KEY_RIGHTSHIFT KEY_FIRST + 14
00055 #define KEY_LEFTCONTROL KEY_FIRST + 15
00056 #define KEY_RIGHTCONTROL KEY_FIRST + 16
00057 #define KEY_LEFTWINDOW KEY_FIRST + 17
00058 #define KEY_RIGHTWINDOW KEY_FIRST + 18
00059 #define KEY_LEFTALT KEY_FIRST + 19
00060 #define KEY_RIGHTALT KEY_FIRST + 20
00061 #define KEY_ENTER KEY_FIRST + 21
00062 #define KEY_PRINTSCREEN KEY_FIRST + 22
00063 #define KEY_SCROLLLOCK KEY_FIRST + 23
00064 #define KEY_NUMLOCK KEY_FIRST + 24
00065 #define KEY_PAUSE KEY_FIRST + 25
00066 #define KEY_INSERT KEY_FIRST + 26
00067 #define KEY_HOME KEY_FIRST + 27
00068 #define KEY_END KEY_FIRST + 28
00069 #define KEY_PAGEUP KEY_FIRST + 28
00070 #define KEY_PAGEDOWN KEY_FIRST + 30
00071 #define KEY_ARROW_DOWN KEY_FIRST + 31
00072 #define KEY_ARROW_UP KEY_FIRST + 32
00073 #define KEY_ARROW_LEFT KEY_FIRST + 33
00074 #define KEY_ARROW_RIGHT KEY_FIRST + 34
00075 #define KEY_KEYPAD_DIVIDE KEY_FIRST + 35
00076 #define KEY_KEYPAD_MULTIPLY KEY_FIRST + 36
00077 #define KEY_KEYPAD_SUBTRACT KEY_FIRST + 37
00078 #define KEY_KEYPAD_ADD KEY_FIRST + 38
00079 #define KEY_KEYPAD_ENTER KEY_FIRST + 39
00080 #define KEY_KEYPAD_PERIOD KEY_FIRST + 40
00081 #define KEY_KEYPAD_0 KEY_FIRST + 41
00082 #define KEY_KEYPAD_1 KEY_FIRST + 42
00083 #define KEY_KEYPAD_2 KEY_FIRST + 43
00084 #define KEY_KEYPAD_3 KEY_FIRST + 44
00085 #define KEY_KEYPAD_4 KEY_FIRST + 45
00086 #define KEY_KEYPAD_5 KEY_FIRST + 46
00087 #define KEY_KEYPAD_6 KEY_FIRST + 47
00088 #define KEY_KEYPAD_7 KEY_FIRST + 48
00089 #define KEY_KEYPAD_8 KEY_FIRST + 49
00090 #define KEY_KEYPAD_9 KEY_FIRST + 50

```

```

00091 #define KEY_BACKSPACE KEY_FIRST + 51
00092 #define KEY_TAB KEY_FIRST + 52
00093 #define KEY_DELETE KEY_FIRST + 53
00094 #define KEY_ESCAPE KEY_FIRST + 54
00095 #define KEY_LAST KEY_ESCAPE
00097 #define MOUSE_BUTTONUP 0
00098 #define MOUSE_BUTTONDOWN 1
00100 #define MOUSE_LEFTBUTTON 0
00101 #define MOUSE_RIGHTBUTTON 1
00102 #define MOUSE_MIDDLEBUTTON 2
00103 #define MOUSE_LAST MOUSE_MIDDLEBUTTON + 1
00105 #define MOUSE_SCROLL_DOWN 0
00106 #define MOUSE_SCROLL_UP 1
00108 #define WINDOWSTYLE_BARE 1
00109 #define WINDOWSTYLE_DEFAULT 2
00110 #define WINDOWSTYLE_POPUP 3
00112 #define WINDOWSTATE_NORMAL 0
00113 #define WINDOWSTATE_MAXIMIZED 1
00114 #define WINDOWSTATE_MINIMIZED 2
00115 #define WINDOWSTATE_FULLSCREEN 3
00117 #define DECORATOR_TITLEBAR 0x01
00118 #define DECORATOR_ICON 0x02
00119 #define DECORATOR_BORDER 0x04
00120 #define DECORATOR_MINIMIZEBUTTON 0x08
00121 #define DECORATOR_MAXIMIZEBUTTON 0x10
00122 #define DECORATOR_CLOSEBUTTON 0x20
00123 #define DECORATOR_VERTICALSCROLLBAR 0x40
00124 #define DECORATOR_HORIZONTALSCROLLBAR 0x80
00125 #define DECORATOR_SIZEABLEBORDER 0x100
00126
00127 #define LINUX_DECORATOR_BORDER 1L << 1
00128 #define LINUX_DECORATOR_MOVE 1L << 2
00129 #define LINUX_DECORATOR_MINIMIZE 1L << 3
00130 #define LINUX_DECORATOR_MAXIMIZE 1L << 4
00131 #define LINUX_DECORATOR_CLOSE 1L << 5
00132
00133 #define FOUNDATION_ERROR 0
00134 #define FOUNDATION_OKAY 1
00135
00136 #define ERROR_NOCONTEXT 0
00137 #define ERROR_INVALIDWINDOWNAME 1
00138 #define ERROR_INVALIDWINDOWINDEX 2
00139 #define ERROR_INVALIDWINDOWSTATE 3
00140 #define ERROR_INVALIDRESOLUTION 4
00141 #define ERROR_INVALIDCONTEXT 5
00142 #define ERROR_EXISTINGCONTEXT 6
00143 #define ERROR_NOTINITIALIZED 7
00144 #define ERROR_ALREADYINITIALIZED 8
00145 #define ERROR_INVALIDTITLEBAR 9
00146 #define ERROR_INVALIDEVENT 10
00147 #define ERROR_WINDOWNOTFOUND 11
00148 #define ERROR_INVALIDWINDOWSTYLE 12
00149 #define ERROR_INVALIDWINDOW 13
00150 #define ERROR_FUNCTIONNOTIMPLEMENTED 14
00151 #define ERROR_LINUX_CANNOTCONNECTXSERVER 15
00152 #define ERROR_LINUX_INVALIDVISUALINFO 16
00153 #define ERROR_LINUX_CANNOTCREATEWINDOW 17
00154 #define ERROR_LINUX_FUNCTIONNOTIMPLEMENTED 18
00155 #define ERROR_WINDOWS_CANNOTCREATEWINDOW 19
00156 #define ERROR_WINDOWS_CANNOTINITIALIZE 20
00157 #define ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED 21
00158
00159 #define WARNING_NOTCURRENTCONTEXT 0
00160 #define WARNING_NOGLEXTEENSIONS 1
00161
00162 #define LINUX_FUNCTION 1
00163 #define LINUX_DECORATOR 2
00164
00165
00166
00167 typedef void (*OnKeyEvent)(GLuint Key, GLboolean KeyState);
00168 typedef void (*OnMouseButtonEvent)(GLuint Button, GLboolean ButtonState);
00169 typedef void (*OnMouseWheelEvent)(GLuint WheelDirection);
00170 typedef void (*OnDestroyedEvent)();
00171 typedef void (*OnMaximizedEvent)();
00172 typedef void (*OnMinimizedEvent)();
00173 //typedef void (*OnRestoredEvent)(); //only really works on windows, Linux doesn't even have an atomic for
    it. might need to remove
00174 typedef void (*OnFocusEvent)(GLboolean InFocus);
00175 typedef void (*OnMovedEvent)(GLuint X, GLuint Y);
00176 typedef void (*OnResizeEvent)(GLuint Width, GLuint Height);
00177 typedef void (*OnMouseMoveEvent)(GLuint WindowX, GLuint WindowY, GLuint ScreenX, GLuint
    ScreenY);
00179 //return wether the given string is valid
00180 static inline GLboolean IsValidString(const char* String)
00181 {
00182     return (String != nullptr);

```

```

00183 }
00184
00185 //return whether the given event is valid
00186 static inline GLboolean IsValidKeyEvent(OnKeyEvent OnKeyPressed)
00187 {
00188     return (OnKeyPressed != nullptr);
00189 }
00190 //return whether the given event is valid
00191 static inline GLboolean IsValidMouseWheelEvent(
    OnMouseWheelEvent MouseWheelEvent)
00192 {
00193     return (MouseWheelEvent != nullptr);
00194 }
00195 //return whether the given event is valid
00196 static inline GLboolean IsValidDestroyedEvent(
    OnMaximizedEvent OnMaximized)
00197 {
00198     return (OnMaximized != nullptr);
00199 }
00200 //return whether the given event is valid
00201 static inline GLboolean IsValidFocusEvent(OnFocusEvent OnFocus)
00202 {
00203     return (OnFocus != nullptr);
00204 }
00205 //return whether the given event is valid
00206 static inline GLboolean IsValidMovedEvent(OnMovedEvent OnMoved)
00207 {
00208     return (OnMoved != nullptr);
00209 }
00210 //return whether the given event is valid
00211 static inline GLboolean IsValidMouseMoveEvent(
    OnMouseMoveEvent OnMouseMove)
00212 {
00213     return (OnMouseMove != nullptr);
00214 }
00215
00216 //print the warning message associated with the given warning number
00217 static inline void PrintWarningMessage(GLuint WarningNumber)
00218 {
00219     switch(WarningNumber)
00220     {
00221         case WARNING_NOGLEXTENSIONS:
00222         {
00223             printf("Warning: no OpenGL extensions available \n");
00224             break;
00225         }
00226         case WARNING_NOTCURRENTCONTEXT:
00227         {
00228             printf("Warning: window not the current OpenGL context being rendered to \n");
00229             break;
00230         }
00231         default:
00232         {
00233             printf("Warning: unspecified warning \n");
00234             break;
00235         }
00236     }
00237 }
00238 }
00239 }
00240
00241 //print out the error associated with the given error number
00242 static void PrintErrorMessage(GLuint ErrorNumber)
00243 {
00244     switch(ErrorNumber)
00245     {
00246         case ERROR_NOCONTEXT:
00247         {
00248             printf("Error: An OpenGL context must first be created(initialize the window) \n");
00249             break;
00250         }
00251         case ERROR_INVALIDWINDOWNAME:
00252         {
00253             printf("Error: invald window name \n");
00254             break;
00255         }
00256         case ERROR_INVALIDWINDOWINDEX:
00257         {
00258             printf("Error: invalid window index \n");
00259             break;
00260         }
00261         case ERROR_INVALIDWINDOWSTATE:
00262         {
00263             printf("Error: invalid window state \n");
00264         }
00265     }
00266 }

```



```

00267         break;
00268     }
00269
00270     case ERROR_INVALIDRESOLUTION:
00271     {
00272         printf("Error: invalid resolution \n");
00273         break;
00274     }
00275
00276     case ERROR_INVALIDCONTEXT:
00277     {
00278         printf("Error: Failed to create OpenGL context \n");
00279         break;
00280     }
00281
00282     case ERROR_EXISTINGCONTEXT:
00283     {
00284         printf("Error: context already created \n");
00285         break;
00286     }
00287
00288     case ERROR_NOTINITIALIZED:
00289     {
00290         printf("Error: Window manager not initialized \n");
00291         break;
00292     }
00293
00294     case ERROR_ALREADYINITIALIZED:
00295     {
00296         printf("Error: window has already been initialized \n");
00297         break;
00298     }
00299
00300     case ERROR_INVALIDTITLEBAR:
00301     {
00302         printf("Error: invalid title bar name (cannot be null or nullptr) \n");
00303         break;
00304     }
00305
00306     case ERROR_INVALIDEVENT:
00307     {
00308         printf("Error: invalid event callback given \n");
00309         break;
00310     }
00311
00312     case ERROR_WINDOWNOTFOUND:
00313     {
00314         printf("Error: window was not found \n");
00315         break;
00316     }
00317
00318     case ERROR_INVALIDWINDOWSTYLE:
00319     {
00320         printf("Error: invalid window style given \n");
00321         break;
00322     }
00323
00324     case ERROR_INVALIDWINDOW:
00325     {
00326         printf("Error: invalid window given \n");
00327         break;
00328     }
00329
00330     case ERROR_FUNCTIONNOTIMPLEMENTED:
00331     {
00332         printf("Error: I'm sorry but this function has not been implemented yet :( \n");
00333         break;
00334     }
00335
00336     case ERROR_LINUX_CANNOTCONNECTXSERVER:
00337     {
00338         printf("Error: cannot connect to X server \n");
00339         break;
00340     }
00341
00342     case ERROR_LINUX_INVALIDVISUALINFO:
00343     {
00344         printf("Error: Invalid visual information given \n");
00345         break;
00346     }
00347
00348     case ERROR_LINUX_CANNOTCREATEWINDOW:
00349     {
00350         printf("Error: failed to create window \n");
00351         break;
00352     }
00353

```



```

00073     }
00074 #endif
00075     GetInstance()->Windows.clear();
00076 }
00077 }
00078
00079 /*****
00092 FWindow* WindowManager::GetWindowByName(const char* WindowName)
00093 {
00094     if (DoesExist(WindowName))
00095     {
00096 #if defined(CURRENT_OS_WINDOWS)
00097         for each(auto CurrentWindow in GetInstance()->Windows)
00098         {
00099             if (CurrentWindow->Name == WindowName)
00100             {
00101                 return CurrentWindow;
00102             }
00103         }
00104 #endif
00105 #if defined(CURRENT_OS_LINUX)
00106         for (auto CurrentWindow : GetInstance()->Windows)
00107         {
00108             if (CurrentWindow->Name == WindowName)
00109             {
00110                 return CurrentWindow;
00111             }
00112         }
00113 #endif
00114 #endif
00115     PrintErrorMessage(ERROR_WINDOWNOTFOUND);
00116     return nullptr;
00117 }
00118 return nullptr;
00119 }
00120
00121 /*****
00134 FWindow* WindowManager::GetWindowByIndex(GLuint WindowIndex)
00135 {
00136     if (DoesExist(WindowIndex))
00137     {
00138 #if defined(CURRENT_OS_WINDOWS)
00139         for each (auto CurrentWindow in GetInstance()->Windows)
00140         {
00141             if (CurrentWindow->ID == WindowIndex)
00142             {
00143                 return CurrentWindow;
00144             }
00145         }
00146 #endif
00147 #if defined(CURRENT_OS_LINUX)
00148         for (auto CurrentWindow : GetInstance()->Windows)
00149         {
00150             if (CurrentWindow->ID == WindowIndex)
00151             {
00152                 return CurrentWindow;
00153             }
00154         }
00155 #endif
00156 #endif
00157     PrintErrorMessage(ERROR_WINDOWNOTFOUND);
00158     return nullptr;
00159 }
00160 return FOUNDATION_ERROR;
00161 }
00162 }
00163
00164 /*****
00177 WindowManager* WindowManager::AddWindow(
00178 FWindow* NewWindow)
00179 {
00180     if (GetInstance()->IsInitialized())
00181     {
00182         if (NewWindow != nullptr)
00183         {
00184             GetInstance()->Windows.push_back(NewWindow);
00185             NewWindow->ID = GetInstance()->Windows.size() - 1;
00186             NewWindow->Initialize();
00187             return GetInstance();
00188         }
00189         PrintErrorMessage(ERROR_INVALIDWINDOW);
00190         return nullptr;
00191     }
00192     PrintErrorMessage(ERROR_NOTINITIALIZED);
00193     return nullptr;
00194 }

```

```

00195 /*****
00206 WindowManager* WindowManager::GetInstance()
00207 {
00208     if(!WindowManager::Instance)
00209     {
00210         WindowManager::Instance = new WindowManager();
00211         return WindowManager::Instance;
00212     }
00213     else
00214     {
00215         return WindowManager::Instance;
00216     }
00217 }
00218 }
00219
00220 /*****
00233 GLboolean WindowManager::DoesExist(const char* WindowName)
00234 {
00235     if (GetInstance()->IsInitialized())
00236     {
00237         if (IsValidString(WindowName))
00238         {
00239             #if defined(CURRENT_OS_WINDOWS)
00240                 for each(auto iter in GetInstance()->Windows)
00241                 {
00242                     if (iter->Name == WindowName)
00243                     {
00244                         return GL_TRUE;
00245                     }
00246                 }
00247             #endif
00248             #if defined(CURRENT_OS_LINUX)
00249                 for (auto iter : GetInstance()->Windows)
00250                 {
00251                     if (iter->Name == WindowName)
00252                     {
00253                         return GL_TRUE;
00254                     }
00255                 }
00256             #endif
00257         }
00258         PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
00259         return GL_FALSE;
00260     }
00261     return GL_FALSE;
00262 }
00263 }
00264
00265 /*****
00278 GLboolean WindowManager::DoesExist(GLuint WindowIndex)
00279 {
00280     if (GetInstance()->IsInitialized())
00281     {
00282         if (WindowIndex <= (GetInstance()->Windows.size() - 1))
00283         {
00284             return FOUNDATION_OKAY;
00285         }
00286         PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
00287         return FOUNDATION_ERROR;
00288     }
00289     return FOUNDATION_ERROR;
00290 }
00291 }
00292
00293 /*****
00304 GLuint WindowManager::GetNumWindows()
00305 {
00306     if(GetInstance()->IsInitialized())
00307     {
00308         return GetInstance()->Windows.size();
00309     }
00310     PrintErrorMessage(ERROR_NOTINITIALIZED);
00311     return FOUNDATION_ERROR;
00312 }
00313 }
00314
00315 /*****
00324 void WindowManager::ShutDown()
00325 {
00326     #if defined(CURRENT_OS_WINDOWS)
00327         for each(auto CurrentWindow in GetInstance()->Windows)
00328         {
00329             delete CurrentWindow;
00330         }
00331         GetInstance()->Windows.clear();
00332     }
00333 }

```

```

00334 #endif
00335
00336 #if defined(CURRENT_OS_LINUX)
00337     for (auto CurrentWindow : GetInstance()->Windows)
00338     {
00339         delete CurrentWindow;
00340     }
00341
00342     GetInstance()->Windows.clear();
00343
00344     XCloseDisplay(GetInstance()->m_Display);
00345 #endif
00346
00347     delete Instance;
00348 }
00349
00350 /*****
00362 GLboolean WindowManager::GetMousePositionInScreen(GLuint& X, GLuint&
    Y)
00363 {
00364     if (GetInstance()->IsInitialized())
00365     {
00366         X = GetInstance()->ScreenMousePosition[0];
00367         Y = GetInstance()->ScreenMousePosition[1];
00368         return FOUNDATION_OKAY;
00369     }
00370
00371     PrintErrorMessage(ERROR_NOTINITIALIZED);
00372     return FOUNDATION_ERROR;
00373 }
00374 }
00375
00376 /*****
00387 GLuint* WindowManager::GetMousePositionInScreen()
00388 {
00389     if (GetInstance()->IsInitialized())
00390     {
00391         return GetInstance()->ScreenMousePosition;
00392     }
00393
00394     PrintErrorMessage(ERROR_NOTINITIALIZED);
00395     return nullptr;
00396 }
00397
00398 /*****
00410 GLboolean WindowManager::SetMousePositionInScreen(GLuint X, GLuint Y)
    )
00411 {
00412     GetInstance()->ScreenMousePosition[0] = X;
00413     GetInstance()->ScreenMousePosition[1] = Y;
00414     #if defined(CURRENT_OS_WINDOWS)
00415         return Windows_SetMousePositionInScreen(X, Y);
00416     #endif
00417
00418     #if defined(CURRENT_OS_LINUX)
00419         return Linux_SetMousePositionInScreen(X, Y);
00420     #endif
00421 }
00422
00423 /*****
00435 GLuint* WindowManager::GetScreenResolution()
00436 {
00437     if (GetInstance()->IsInitialized())
00438     {
00439         #if defined(CURRENT_OS_WINDOWS)
00440             RECT l_Screen;
00441             HWND m_Desktop = GetDesktopWindow();
00442             GetWindowRect(m_Desktop, &l_Screen);
00443
00444             GetInstance()->ScreenResolution[0] = l_Screen.right;
00445             GetInstance()->ScreenResolution[1] = l_Screen.bottom;
00446             return GetInstance()->ScreenResolution;
00447         #endif
00448     #endif
00449
00450     #if defined(CURRENT_OS_LINUX)
00451         GetInstance()->ScreenResolution[0] = WidthOfScreen(
            XDefaultScreenOfDisplay(GetInstance()->m_Display));
00452         GetInstance()->ScreenResolution[1] = HeightOfScreen(
            XDefaultScreenOfDisplay(GetInstance()->m_Display));
00453
00454         return GetInstance()->ScreenResolution;
00455     #endif
00456 }
00457
00458 PrintErrorMessage(ERROR_NOTINITIALIZED);
00459 return nullptr;

```

```

00460 }
00461
00462 /*****
00471 GLboolean WindowManager::PollForEvents()
00472 {
00473     if (GetInstance()->IsInitialized())
00474     {
00475         #if defined(CURRENT_OS_WINDOWS)
00476             return GetInstance()->Windows_PollForEvents();
00477         #endif
00478         #if defined (CURRENT_OS_LINUX)
00479             return GetInstance()->Linux_PollForEvents();
00480         #endif
00481     }
00482
00483     PrintErrorMessage(ERROR_NOTINITIALIZED);
00484     return FOUNDATION_ERROR;
00485 }
00486
00487
00488 /*****
00500 GLboolean WindowManager::GetScreenResolution(GLuint& Width, GLuint&
    Height)
00501 {
00502     if (GetInstance()->IsInitialized())
00503     {
00504         #if defined(CURRENT_OS_WINDOWS)
00505
00506             RECT l_Screen;
00507             HWND m_Desktop = GetDesktopWindow();
00508             GetWindowRect(m_Desktop, &l_Screen);
00509             Width = l_Screen.right;
00510             Height = l_Screen.bottom;
00511         #endif
00512         #if defined(CURRENT_OS_LINUX)
00513
00514             Width = WidthOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
00515             Height = HeightOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
00516
00517             GetInstance()->ScreenResolution[0] = Width;
00518             GetInstance()->ScreenResolution[1] = Height;
00519         #endif
00520     }
00521     return FOUNDATION_OKAY;
00522 }
00523
00524 PrintErrorMessage(ERROR_NOTINITIALIZED);
00525 return FOUNDATION_ERROR;
00526 }
00527
00528 /*****
00542 GLboolean WindowManager::GetWindowResolution(const char* WindowName,
    GLuint& Width, GLuint& Height)
00543 {
00544     if (GetInstance()->IsInitialized())
00545     {
00546         if (DoesExist(WindowName))
00547         {
00548             if (GetWindowByName(WindowName)->GetResolution(Width, Height))
00549             {
00550                 return FOUNDATION_OKAY;
00551             }
00552             return FOUNDATION_ERROR;
00553         }
00554         return FOUNDATION_ERROR;
00555     }
00556
00557     PrintErrorMessage(ERROR_NOTINITIALIZED);
00558     return FOUNDATION_ERROR;
00559 }
00560
00561 /*****
00574 GLboolean WindowManager::GetWindowResolution(GLuint WindowIndex, GLuint&
    Width, GLuint& Height)
00575 {
00576     if (DoesExist(WindowIndex))
00577     {
00578         GetWindowByIndex(WindowIndex)->GetResolution(Width, Height);
00579         return FOUNDATION_OKAY;
00580     }
00581
00582     PrintErrorMessage(ERROR_NOTINITIALIZED);
00583     return FOUNDATION_ERROR;
00584 }
00585
00586 /*****

```

```

00600 GLuint* WindowManager::GetWindowResolution(const char* WindowName)
00601 {
00602     if(DoesExist(WindowName))
00603     {
00604         return GetWindowByName(WindowName)->GetResolution();
00605     }
00606     return nullptr;
00607 }
00608
00609
00610 /*****
00624 GLuint* WindowManager::GetWindowResolution(GLuint WindowIndex)
00625 {
00626     if(DoesExist(WindowIndex))
00627     {
00628         return GetWindowByIndex(WindowIndex)->GetResolution();
00629     }
00630     return nullptr;
00631 }
00632
00633
00634 /*****
00647 GLboolean WindowManager::SetWindowResolution(const char* WindowName,
00648     GLuint Width, GLuint Height)
00649 {
00650     if(DoesExist(WindowName))
00651     {
00652         return GetWindowByName(WindowName)->SetResolution(Width, Height);
00653     }
00654     return GL_FALSE;
00655 }
00656
00657 /*****
00670 GLboolean WindowManager::SetWindowResolution(GLuint WindowIndex, GLuint
00671     Width, GLuint Height)
00672 {
00673     if(DoesExist(WindowIndex))
00674     {
00675         return GetWindowByIndex(WindowIndex)->SetResolution(Width, Height);
00676     }
00677     return GL_FALSE;
00678 }
00679
00680 /*****
00693 GLboolean WindowManager::GetWindowPosition(const char* WindowName, GLuint&
00694     X, GLuint& Y)
00695 {
00696     if(DoesExist(WindowName))
00697     {
00698         return GetWindowByName(WindowName)->GetPosition(X, Y);
00699     }
00700     return GL_FALSE;
00701 }
00702
00703 /*****
00716 GLboolean WindowManager::GetWindowPosition(GLuint WindowIndex, GLuint& X,
00717     GLuint& Y)
00718 {
00719     if(DoesExist(WindowIndex))
00720     {
00721         return GetWindowByIndex(WindowIndex)->GetPosition(X, Y);
00722     }
00723     return GL_FALSE;
00724 }
00725
00726 /*****
00741 GLuint* WindowManager::GetWindowPosition(const char* WindowName)
00742 {
00743     if(DoesExist(WindowName))
00744     {
00745         return GetWindowByName(WindowName)->GetPosition();
00746     }
00747     return nullptr;
00748 }
00749
00750
00751 /*****
00766 GLuint* WindowManager::GetWindowPosition(GLuint WindowIndex)
00767 {
00768     if(WindowIndex <= GetInstance()->Windows.size() - 1)
00769     {
00770         return GetWindowByIndex(WindowIndex)->GetPosition();
00771     }

```

```

00772
00773     return nullptr;
00774 }
00775
00776 /*****
00789 GLboolean WindowManager::SetWindowPosition(const char* WindowName, GLuint X
, GLuint Y)
00790 {
00791     if(DoesExist(WindowName))
00792     {
00793         return GetWindowByName(WindowName)->SetPosition(X, Y);
00794     }
00795
00796     return FOUNDATION_ERROR;
00797 }
00798
00799 /*****
00812 GLboolean WindowManager::SetWindowPosition(GLuint WindowIndex, GLuint X,
GLuint Y)
00813 {
00814     if(WindowIndex <= GetInstance()->Windows.size() -1)
00815     {
00816         return GetWindowByIndex(WindowIndex)->SetPosition(X, Y);
00817     }
00818
00819     return FOUNDATION_ERROR;
00820 }
00821
00822 /*****
00835 GLboolean WindowManager::GetMousePositionInWindow(const char*
WindowName, GLuint& X, GLuint& Y)
00836 {
00837     if(DoesExist(WindowName))
00838     {
00839         return GetWindowByName(WindowName)->GetMousePosition(X, Y);
00840     }
00841
00842     return FOUNDATION_ERROR;
00843 }
00844
00845 /*****
00858 GLboolean WindowManager::GetMousePositionInWindow(GLuint WindowIndex
, GLuint& X, GLuint& Y)
00859 {
00860     if(DoesExist(WindowIndex))
00861     {
00862         return GetWindowByIndex(WindowIndex)->GetMousePosition(X, Y);
00863     }
00864
00865     return FOUNDATION_ERROR;
00866 }
00867
00868 /*****
00883 GLuint* WindowManager::GetMousePositionInWindow(const char*
WindowName)
00884 {
00885     if(DoesExist(WindowName))
00886     {
00887         return GetWindowByName(WindowName)->GetMousePosition();
00888     }
00889
00890     return nullptr;
00891 }
00892
00893 /*****
00908 GLuint* WindowManager::GetMousePositionInWindow(GLuint WindowIndex)
00909 {
00910     if(DoesExist(WindowIndex))
00911     {
00912         return GetWindowByIndex(WindowIndex)->GetMousePosition();
00913     }
00914     PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
00915     return nullptr;
00916 }
00917
00918 /*****
00931 GLboolean WindowManager::SetMousePositionInWindow(const char*
WindowName, GLuint X, GLuint Y)
00932 {
00933     if(DoesExist(WindowName))
00934     {
00935         return GetWindowByName(WindowName)->SetMousePosition(X, Y);
00936     }
00937
00938     return FOUNDATION_ERROR;
00939 }
00940

```



```

00941 /*****
00954 GLboolean WindowManager::SetMousePositionInWindow(GLuint WindowIndex
, GLuint X, GLuint Y)
00955 {
00956     if(DoesExist(WindowIndex))
00957     {
00958         return GetWindowByIndex(WindowIndex)->SetMousePosition(X, Y);
00959     }
00960     return FOUNDATION_ERROR;
00961 }
00962
00963
00964 /*****
00978 GLboolean WindowManager::WindowGetKey(const char* WindowName, GLuint Key)
00979 {
00980     if(DoesExist(WindowName))
00981     {
00982         return GetWindowByName(WindowName)->GetKeyState(Key);
00983     }
00984     return FOUNDATION_ERROR;
00985 }
00986
00987
00988 /*****
01002 GLboolean WindowManager::WindowGetKey(GLuint WindowIndex, GLuint Key)
01003 {
01004     if(DoesExist(WindowIndex))
01005     {
01006         return GetWindowByIndex(WindowIndex)->GetKeyState(Key);
01007     }
01008     return FOUNDATION_ERROR;
01009 }
01010
01011
01012 /*****
01025 GLboolean WindowManager::GetWindowShouldClose(const char* WindowName)
01026 {
01027     if(DoesExist(WindowName))
01028     {
01029         return GetWindowByName(WindowName)->GetShouldClose();
01030     }
01031     return FOUNDATION_ERROR;
01032 }
01033
01034
01035 /*****
01048 GLboolean WindowManager::GetWindowShouldClose(GLuint WindowIndex)
01049 {
01050     if(DoesExist(WindowIndex))
01051     {
01052         return GetWindowByIndex(WindowIndex)->GetShouldClose();
01053     }
01054     return FOUNDATION_ERROR;
01055 }
01056
01057
01058 /*****
01069 GLboolean WindowManager::WindowSwapBuffers(const char* WindowName)
01070 {
01071     if(DoesExist(WindowName))
01072     {
01073         return GetWindowByName(WindowName)->SwapDrawBuffers();
01074     }
01075     return FOUNDATION_ERROR;
01076 }
01077
01078
01079 /*****
01090 GLboolean WindowManager::WindowSwapBuffers(GLuint WindowIndex)
01091 {
01092     if(DoesExist(WindowIndex))
01093     {
01094         return GetWindowByIndex(WindowIndex)->SwapDrawBuffers();
01095     }
01096     return FOUNDATION_ERROR;
01097 }
01098
01099
01100 /*****
01113 GLboolean WindowManager::GetWindowIsFullScreen(const char* WindowName)
01114 {
01115     if(DoesExist(WindowName))
01116     {
01117         return GetWindowByName(WindowName)->GetIsFullScreen();
01118     }
01119     return FOUNDATION_ERROR;
01120 }

```

```

01121 }
01122
01123 /*****
01136 GLboolean WindowManager::GetWindowIsFullScreen(GLuint WindowIndex)
01137 {
01138     if(WindowIndex <= GetInstance()->Windows.size() -1)
01139     {
01140         return GetWindowByIndex(WindowIndex)->GetIsFullScreen();
01141     }
01142     return FOUNDATION_ERROR;
01143 }
01144
01145 /*****
01158 GLboolean WindowManager::SetFullScreen(const char* WindowName, GLboolean
ShouldBeFullscreen)
01159 {
01160     if(DoesExist(WindowName))
01161     {
01162         return GetWindowByName(WindowName)->FullScreen(ShouldBeFullscreen);
01163     }
01164     return FOUNDATION_ERROR;
01165 }
01166
01167 /*****
01180 GLboolean WindowManager::SetFullScreen(GLuint WindowIndex, GLboolean
ShouldBeFullscreen)
01181 {
01182     if (DoesExist(WindowIndex))
01183     {
01184         return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeFullscreen);
01185     }
01186     return FOUNDATION_ERROR;
01187 }
01188
01189 /*****
01203 GLboolean WindowManager::GetWindowIsMinimized(const char* WindowName)
01204 {
01205     if(DoesExist(WindowName))
01206     {
01207         return GetWindowByName(WindowName)->GetIsMinimized();
01208     }
01209     return FOUNDATION_ERROR;
01210 }
01211
01212 /*****
01226 GLboolean WindowManager::GetWindowIsMinimized(GLuint WindowIndex)
01227 {
01228     if(DoesExist(WindowIndex))
01229     {
01230         return GetWindowByIndex(WindowIndex)->GetIsMinimized();
01231     }
01232     return FOUNDATION_ERROR;
01233 }
01234
01235 /*****
01248 GLboolean WindowManager::MinimizeWindow(const char* WindowName, GLboolean
ShouldBeMinimized)
01249 {
01250     if (DoesExist(WindowName))
01251     {
01252         return GetWindowByName(WindowName)->FullScreen(ShouldBeMinimized);
01253     }
01254     return FOUNDATION_ERROR;
01255 }
01256
01257 /*****
01270 GLboolean WindowManager::MinimizeWindow(GLuint WindowIndex, GLboolean
ShouldBeMinimized)
01271 {
01272     if (DoesExist(WindowIndex))
01273     {
01274         return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMinimized);
01275     }
01276     return FOUNDATION_ERROR;
01277 }
01278
01279 /*****
01293 GLboolean WindowManager::GetWindowIsMaximized(const char* WindowName)
01294 {
01295     if(DoesExist(WindowName))

```

```

01296     {
01297         return GetWindowByName(WindowName)->GetIsMaximized();
01298     }
01299
01300     return FOUNDATION_ERROR;
01301 }
01302
01303 /*****
01316 GLboolean WindowManager::GetWindowIsMaximized(GLuint WindowIndex)
01317 {
01318     if(DoesExist(WindowIndex))
01319     {
01320         return GetWindowByIndex(WindowIndex)->GetIsMaximized();
01321     }
01322
01323     return FOUNDATION_ERROR;
01324 }
01325
01326 /*****
01338 GLboolean WindowManager::MaximizeWindow(const char* WindowName, GLboolean
ShouldBeMaximized)
01339 {
01340     if (DoesExist(WindowName))
01341     {
01342         return GetWindowByName(WindowName)->FullScreen(ShouldBeMaximized);
01343     }
01344
01345     return FOUNDATION_ERROR;
01346 }
01347
01348 /*****
01360 GLboolean WindowManager::MaximizeWindow(GLuint WindowIndex, GLboolean
ShouldBeMaximized)
01361 {
01362     if (DoesExist(WindowIndex))
01363     {
01364         return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMaximized);
01365     }
01366
01367     return FOUNDATION_ERROR;
01368 }
01369
01370 /*****
01383 const char* WindowManager::GetWindowName(GLuint WindowIndex)
01384 {
01385     if(DoesExist(WindowIndex))
01386     {
01387         return GetWindowByIndex(WindowIndex)->GetWindowName();
01388     }
01389
01390     return nullptr;
01391 }
01392
01393 /*****
01406 GLuint WindowManager::GetWindowIndex(const char* WindowName)
01407 {
01408     if(DoesExist(WindowName))
01409     {
01410         return GetWindowByName(WindowName)->ID;
01411     }
01412
01413     return 0;
01414 }
01415
01416 /*****
01428 GLboolean WindowManager::SetWindowTitleBar(const char* WindowName, const
char* NewTitle)
01429 {
01430     if(DoesExist(WindowName) && IsValidString(NewTitle))
01431     {
01432         return GetWindowByName(WindowName)->SetTitleBar(NewTitle);
01433     }
01434
01435     return FOUNDATION_ERROR;
01436 }
01437
01438 /*****
01450 GLboolean WindowManager::SetWindowTitleBar(GLuint WindowIndex, const char*
NewTitle)
01451 {
01452     if(DoesExist(WindowIndex) && IsValidString(NewTitle))
01453     {
01454         return GetWindowByIndex(WindowIndex)->SetTitleBar(NewTitle);
01455     }
01456
01457     return FOUNDATION_ERROR;
01458 }

```

```

01459
01460 /*****
01473 GLboolean WindowManager::GetWindowIsInFocus(const char* WindowName)
01474 {
01475     if (DoesExist(WindowName))
01476     {
01477         return GetWindowByName(WindowName)->GetInFocus();
01478     }
01479     return FOUNDATION_ERROR;
01480 }
01481
01482 /*****
01496 GLboolean WindowManager::GetWindowIsInFocus(GLuint WindowIndex)
01497 {
01498     if (DoesExist(WindowIndex))
01499     {
01500         return GetWindowByIndex(WindowIndex)->GetInFocus();
01501     }
01502     return FOUNDATION_ERROR;
01503 }
01504
01505 /*****
01518 GLboolean WindowManager::FocusWindow(const char* WindowName, GLboolean
ShouldBeFocused)
01519 {
01520     if (DoesExist(WindowName))
01521     {
01522         return GetWindowByName(WindowName)->Focus(ShouldBeFocused);
01523     }
01524     return FOUNDATION_ERROR;
01525 }
01526
01527 /*****
01540 GLboolean WindowManager::FocusWindow(GLuint WindowIndex, GLboolean
ShouldBeFocused)
01541 {
01542     if (DoesExist(WindowIndex))
01543     {
01544         return GetWindowByIndex(WindowIndex)->Focus(ShouldBeFocused);
01545     }
01546     return FOUNDATION_ERROR;
01547 }
01548
01549 /*****
01561 GLboolean WindowManager::RestoreWindow(const char* WindowName)
01562 {
01563     if (DoesExist(WindowName))
01564     {
01565         return GetWindowByName(WindowName)->Restore();
01566     }
01567     return FOUNDATION_ERROR;
01568     //implement window focusing
01569 }
01570
01571 /*****
01582 GLboolean WindowManager::RestoreWindow(GLuint WindowIndex)
01583 {
01584     if (DoesExist(WindowIndex))
01585     {
01586         return GetWindowByIndex(WindowIndex)->Restore();
01587     }
01588     return FOUNDATION_ERROR;
01589     //implement window focusing
01590 }
01591
01592 /*****
01605 GLboolean WindowManager::SetWindowSwapInterval(const char* WindowName,
GLint a_SyncSetting)
01606 {
01607     if (DoesExist(WindowName))
01608     {
01609         return GetWindowByName(WindowName)->SetSwapInterval(a_SyncSetting);
01610     }
01611     return FOUNDATION_ERROR;
01612 }
01613
01614 /*****
01627 GLboolean WindowManager::SetWindowSwapInterval(GLuint WindowIndex,
GLint a_SyncSetting)
01628 {
01629     if (DoesExist(WindowIndex))

```

```

01630     {
01631         return GetWindowByIndex(WindowIndex)->SetSwapInterval(a_SyncSetting)
01632     ;
01633     }
01634     return FOUNDATION_ERROR;
01635 }
01636
01637 GLboolean WindowManager::SetWindowStyle(const char* WindowName, GLuint
WindowStyle)
01638 {
01639     if (DoesExist(WindowName))
01640     {
01641         return GetWindowByName(WindowName)->SetStyle(WindowStyle);
01642     }
01643     return FOUNDATION_ERROR;
01644 }
01645
01646 GLboolean WindowManager::SetWindowStyle(GLuint WindowIndex, GLuint WindowStyle
)
01647 {
01648     if (DoesExist(WindowIndex))
01649     {
01650         return GetWindowByIndex(WindowIndex)->SetStyle(WindowStyle);
01651     }
01652     return FOUNDATION_ERROR;
01653 }
01654
01655 GLboolean WindowManager::EnableWindowDecorator(const char* WindowName,
GLbitfield Decorators)
01656 {
01657     if (DoesExist(WindowName))
01658     {
01659         return GetWindowByName(WindowName)->EnableDecorator(Decorators);
01660     }
01661     return FOUNDATION_ERROR;
01662 }
01663
01664 GLboolean WindowManager::EnableWindowDecorator(GLuint WindowIndex,
GLbitfield Decorators)
01665 {
01666     if (DoesExist(WindowIndex))
01667     {
01668         return GetWindowByIndex(WindowIndex)->EnableDecorator(Decorators);
01669     }
01670     return FOUNDATION_ERROR;
01671 }
01672
01673 GLboolean WindowManager::DisableWindowDecorator(const char* WindowName
, GLbitfield Decorators)
01674 {
01675     if (DoesExist(WindowName))
01676     {
01677         return GetWindowByName(WindowName)->DisableDecorator(Decorators);
01678     }
01679     return FOUNDATION_ERROR;
01680 }
01681
01682 GLboolean WindowManager::DisableWindowDecorator(GLuint WindowIndex,
GLbitfield Decorators)
01683 {
01684     if (DoesExist(WindowIndex))
01685     {
01686         return GetWindowByIndex(WindowIndex)->DisableDecorator(Decorators);
01687     }
01688     return FOUNDATION_ERROR;
01689 }
01690
01691 /*****
01700 GLboolean WindowManager::SetWindowOnKeyEvent(const char* WindowName,
OnKeyEvent OnKey)
01701 {
01702     if (DoesExist(WindowName))
01703     {
01704         return GetWindowByName(WindowName)->SetOnKeyEvent(OnKey);
01705     }
01706     return FOUNDATION_ERROR;
01707 }
01708
01709

```

```

01720 /*****
01732 GLboolean WindowManager::SetWindowOnKeyEvent (GLuint WindowIndex,
OnKeyEvent OnKey)
01733 {
01734     if (DoesExist (WindowIndex))
01735     {
01736         return GetWindowByIndex (WindowIndex) ->SetOnKeyEvent (OnKey);
01737     }
01738     return FOUNDATION_ERROR;
01739 }
01740
01741
01742 /*****
01754 GLboolean WindowManager::SetWindowOnMouseButtonEvent (const char*
WindowName, OnMouseButtonEvent OnMouseButton)
01755 {
01756     if (DoesExist (WindowName))
01757     {
01758         return GetWindowByName (WindowName) ->SetOnMouseButtonEvent (
OnMouseButton);
01759     }
01760     return FOUNDATION_ERROR;
01761 }
01762
01763
01764 /*****
01776 GLboolean WindowManager::SetWindowOnMouseButtonEvent (GLuint
WindowIndex, OnMouseButtonEvent OnMouseButton)
01777 {
01778     if (DoesExist (WindowIndex))
01779     {
01780         return GetWindowByIndex (WindowIndex) ->
SetOnMouseButtonEvent (OnMouseButton);
01781     }
01782     return FOUNDATION_ERROR;
01783 }
01784
01785
01786 /*****
01798 GLboolean WindowManager::SetWindowOnMouseWheelEvent (const char*
WindowName, OnMouseWheelEvent OnMouseWheel)
01799 {
01800     if (DoesExist (WindowName))
01801     {
01802         return GetWindowByName (WindowName) ->SetOnMouseWheelEvent (
OnMouseWheel);
01803     }
01804     return FOUNDATION_ERROR;
01805 }
01806
01807
01808 /*****
01820 GLboolean WindowManager::SetWindowOnMouseWheelEvent (GLuint
WindowIndex, OnMouseWheelEvent OnMouseWheel)
01821 {
01822     if (DoesExist (WindowIndex))
01823     {
01824         return GetWindowByIndex (WindowIndex) ->
SetOnMouseWheelEvent (OnMouseWheel);
01825     }
01826     return FOUNDATION_ERROR;
01827 }
01828
01829
01830 /*****
01842 GLboolean WindowManager::SetWindowOnDestroyed (const char* WindowName,
OnDestroyedEvent OnDestroyed)
01843 {
01844     if (DoesExist (WindowName))
01845     {
01846         return GetWindowByName (WindowName) ->SetOnDestroyed (OnDestroyed);
01847     }
01848     return FOUNDATION_ERROR;
01849 }
01850
01851
01852 /*****
01864 GLboolean WindowManager::SetWindowOnDestroyed (GLuint WindowIndex,
OnDestroyedEvent OnDestroyed)
01865 {
01866     if (DoesExist (WindowIndex))
01867     {
01868         return GetWindowByIndex (WindowIndex) ->SetOnDestroyed (OnDestroyed);
01869     }
01870     return FOUNDATION_ERROR;
01871 }
01872

```

```

01873
01874 /*****
01886 GLboolean WindowManager::SetWindowOnMaximized(const char* WindowName,
OnMaximizedEvent OnMaximized)
01887 {
01888     if (DoesExist(WindowName))
01889     {
01890         return GetWindowByName(WindowName)->SetOnMaximized(OnMaximized);
01891     }
01892
01893     return FOUNDATION_ERROR;
01894 }
01895
01896 /*****
01908 GLboolean WindowManager::SetWindowOnMaximized(GLuint WindowIndex,
OnMaximizedEvent OnMaximized)
01909 {
01910     if (DoesExist(WindowIndex))
01911     {
01912         return GetWindowByIndex(WindowIndex)->SetOnMaximized(OnMaximized);
01913     }
01914
01915     return FOUNDATION_ERROR;
01916 }
01917
01918 /*****
01930 GLboolean WindowManager::SetWindowOnMinimized(const char* WindowName,
OnMinimizedEvent OnMinimized)
01931 {
01932     if (DoesExist(WindowName))
01933     {
01934         return GetWindowByName(WindowName)->SetOnMinimized(OnMinimized);
01935     }
01936
01937     return FOUNDATION_ERROR;
01938 }
01939
01940 /*****
01952 GLboolean WindowManager::SetWindowOnMinimized(GLuint WindowIndex,
OnMinimizedEvent OnMinimized)
01953 {
01954     if (DoesExist(WindowIndex))
01955     {
01956         return GetWindowByIndex(WindowIndex)->SetOnMinimized(OnMinimized);
01957     }
01958
01959     return FOUNDATION_ERROR;
01960 }
01961
01962 void WindowManager::SetWindowOnRestored(const char* WindowName, OnRestoredEvent OnRestored)
01963 {
01964     if (DoesExist(WindowName))
01965     {
01966         GetWindowByName(WindowName)->SetOnRestored(OnRestored);
01967     }
01968 }
01969
01970 void WindowManager::SetWindowOnRestored(GLuint WindowIndex, OnRestoredEvent OnRestored)
01971 {
01972     if (DoesExist(WindowIndex))
01973     {
01974         GetWindowByIndex(WindowIndex)->SetOnRestored(OnRestored);
01975     }
01976 }*/
01977
01978 /*****
01990 GLboolean WindowManager::SetWindowOnFocus(const char* WindowName,
OnFocusEvent OnFocus)
01991 {
01992     if (IsValidString(WindowName))
01993     {
01994         GetWindowByName(WindowName)->FocusEvent = OnFocus;
01995         return FOUNDATION_OKAY;
01996     }
01997
01998     return FOUNDATION_ERROR;
01999 }
02000
02001 /*****
02013 GLboolean WindowManager::SetWindowOnFocus(GLuint WindowIndex,
OnFocusEvent OnFocus)
02014 {
02015     if (DoesExist(WindowIndex))
02016     {
02017         GetWindowByIndex(WindowIndex)->FocusEvent = OnFocus;
02018         return FOUNDATION_OKAY;
02019     }

```

```

02020
02021     return FOUNDATION_ERROR;
02022 }
02023
02024 /*****
02036 GLboolean WindowManager::SetWindowOnMoved(const char* WindowName,
02037     OnMovedEvent OnMoved)
02038 {
02039     if (DoesExist(WindowName))
02040     {
02041         return GetWindowByName(WindowName)->SetOnMoved(OnMoved);
02042     }
02043     return FOUNDATION_ERROR;
02044 }
02045
02046 /*****
02058 GLboolean WindowManager::SetWindowOnMoved(GLuint WindowIndex,
02059     OnMovedEvent OnMoved)
02060 {
02061     if (DoesExist(WindowIndex))
02062     {
02063         return GetWindowByIndex(WindowIndex)->SetOnMoved(OnMoved);
02064     }
02065     return FOUNDATION_ERROR;
02066 }
02067
02068 /*****
02080 GLboolean WindowManager::SetWindowOnResize(const char* WindowName,
02081     OnResizeEvent OnResize)
02082 {
02083     if (DoesExist(WindowName))
02084     {
02085         return GetWindowByName(WindowName)->SetOnResize(OnResize);
02086     }
02087     return FOUNDATION_ERROR;
02088 }
02089
02090 /*****
02102 GLboolean WindowManager::SetWindowOnResize(GLuint WindowIndex,
02103     OnResizeEvent OnResize)
02104 {
02105     if (DoesExist(WindowIndex))
02106     {
02107         return GetWindowByIndex(WindowIndex)->SetOnResize(OnResize);
02108     }
02109     return FOUNDATION_ERROR;
02110 }
02111
02112 /*****
02124 GLboolean WindowManager::SetWindowOnMouseMove(const char* WindowName,
02125     OnMouseMoveEvent OnMouseMove)
02126 {
02127     if (DoesExist(WindowName))
02128     {
02129         return GetWindowByName(WindowName)->SetOnMouseMove(OnMouseMove);
02130     }
02131     return FOUNDATION_ERROR;
02132 }
02133
02134 /*****
02146 GLboolean WindowManager::SetWindowOnMouseMove(GLuint WindowIndex,
02147     OnMouseMoveEvent OnMouseMove)
02148 {
02149     if (DoesExist(WindowIndex))
02150     {
02151         return GetWindowByIndex(WindowIndex)->SetOnMouseMove(OnMouseMove);
02152     }
02153     return FOUNDATION_ERROR;
02154 }
02155
02156 WindowManager* WindowManager::Instance = 0;

```

## 4.15 WindowManager.h File Reference

```

#include "WindowAPI_Defs.h"
#include "Window.h"

```



## Classes

- class [WindowManager](#)

## 4.16 WindowManager.h

```

00001 #ifndef WINDOW_MANAGER_H
00002 #define WINDOW_MANAGER_H
00003
00004
00005 #include "WindowAPI_Defs.h"
00006 #include "Window.h"
00007
00008 class FWindow;
00009
00010 class WindowManager
00011 {
00012     friend FWindow;
00013     public:
00014
00015     WindowManager();
00016     ~WindowManager();
00020     static void ShutDown();
00021
00023     static FWindow* GetWindowByName(const char* WindowName);
00024     static FWindow* GetWindowByIndex(GLuint WindowIndex);
00025
00030     static WindowManager* AddWindow(FWindow* NewWindow);
00031
00032     //return the total amount of windows the manager has
00033     static GLuint GetNumWindows();
00034
00035     //gets and sets for the mouse position in the screen
00036     static GLboolean GetMousePositionInScreen(GLuint& X, GLuint& Y);
00037     static GLuint* GetMousePositionInScreen();
00038     static GLboolean SetMousePositionInScreen(GLuint X, GLuint Y);
00039
00040     // get the screen resolution for the screen that is being drawn to
00041     static GLuint* GetScreenResolution();
00042     static GLboolean GetScreenResolution(GLuint& Width, GLuint& Height);
00043
00044     //these are another way to set and get window variables
00045     //apart from the functions that are available to the user
00046     //via each window
00047
00048     //sets and gets for window resolution
00049     static GLboolean GetWindowResolution(const char* WindowName, GLuint& Width,
    GLuint& Height);
00050     static GLboolean GetWindowResolution(GLuint WindowIndex, GLuint& Width, GLuint&
    Height);
00051     static GLuint* GetWindowResolution(const char* WindowName);
00052     static GLuint* GetWindowResolution(GLuint WindowIndex);
00053     static GLboolean SetWindowResolution(const char* WindowName, GLuint Width,
    GLuint Height);
00054     static GLboolean SetWindowResolution(GLuint WindowIndex, GLuint Width, GLuint
    Height);
00055
00056     //sets and gets for window position
00057     static GLboolean GetWindowPosition(const char* WindowName, GLuint& X, GLuint& Y);
00058     static GLboolean GetWindowPosition(GLuint WindowIndex, GLuint& X, GLuint& Y);
00059     static GLuint* GetWindowPosition(const char* WindowName);
00060     static GLuint* GetWindowPosition(GLuint WindowIndex);
00061     static GLboolean SetWindowPosition(const char* WindowName, GLuint X, GLuint Y);
00062     static GLboolean SetWindowPosition(GLuint WindowIndex, GLuint X, GLuint Y);
00063
00064     //sets and gets for the mouse position in window
00065     static GLboolean GetMousePositionInWindow(const char* WindowName, GLuint& X
    , GLuint& Y);
00066     static GLboolean GetMousePositionInWindow(GLuint WindowIndex, GLuint& X,
    GLuint& Y);
00067     static GLuint* GetMousePositionInWindow(const char* WindowName);
00068     static GLuint* GetMousePositionInWindow(GLuint WindowIndex);
00069     static GLboolean SetMousePositionInWindow(const char* WindowName, GLuint X,
    GLuint Y);
00070     static GLboolean SetMousePositionInWindow(GLuint WindowIndex, GLuint X,
    GLuint Y);
00071
00072     //gets for window keys

```

```

00073     static GLboolean WindowGetKey(const char* WindowName, GLuint Key);
00074     static GLboolean WindowGetKey(GLuint WindowIndex, GLuint Key);
00075
00076     //gets for window should close
00077     static GLboolean GetWindowShouldClose(const char* WindowName);
00078     static GLboolean GetWindowShouldClose(GLuint WindowIndex);
00079
00080     //swap buffers
00081     static GLboolean WindowSwapBuffers(const char* WindowName);
00082     static GLboolean WindowSwapBuffers(GLuint WindowIndex);
00083
00084     //sets and gets for fullscreen
00085     static GLboolean SetFullScreen(const char* WindowName, GLboolean NewState);
00086     static GLboolean SetFullScreen(GLuint WindowIndex, GLboolean NewState);
00087     static GLboolean GetWindowIsFullScreen(const char* WindowName);
00088     static GLboolean GetWindowIsFullScreen(GLuint WindowIndex);
00089
00090     //gets and sets for minimized
00091     static GLboolean GetWindowIsMinimized(const char* WindowName);
00092     static GLboolean GetWindowIsMinimized(GLuint WindowIndex);
00093     static GLboolean MinimizeWindow(const char* WindowName, GLboolean NewState);
00094     static GLboolean MinimizeWindow(GLuint WindowIndex, GLboolean NewState);
00095
00096     //gets and sets for maximised state
00097     static GLboolean GetWindowIsMaximized(const char* WindowName);
00098     static GLboolean GetWindowIsMaximized(GLuint WindowIndex);
00099     static GLboolean MaximizeWindow(const char* WindowName, GLboolean NewState);
00100     static GLboolean MaximizeWindow(GLuint WindowIndex, GLboolean NewState);
00101
00102     //gets and sets for window name and index
00103     static const char* GetWindowName(GLuint WindowIndex);
00104     static GLuint GetWindowIndex(const char* WindowName);
00105
00106     static GLboolean SetWindowTitleBar(const char* WindowName, const char* NewName);
00107     static GLboolean SetWindowTitleBar(GLuint WindowIndex, const char* NewName);
00108
00109     static GLboolean SetWindowIcon(const char* WindowName, const char* Icon, GLuint Width,
    GLuint Height);
00110     static GLboolean SetWindowIcon(GLuint WindowIndex, const char* Icon, GLuint Width,
    GLuint Height);
00111
00112     //gets and sets window is in focus(Linux only?)
00113     static GLboolean GetWindowIsInFocus(const char* WindowName);
00114     static GLboolean GetWindowIsInFocus(GLuint WindowIndex);
00115     static GLboolean FocusWindow(const char* WindowName, GLboolean NewState);
00116     static GLboolean FocusWindow(GLuint WindowIndex, GLboolean NewState);
00117
00118     //gets and sets for restoring the window
00119     static GLboolean RestoreWindow(const char* WindowName);
00120     static GLboolean RestoreWindow(GLuint WindowIndex);
00121
00122     //get window obscurity. I feel like this is completely useless
00123     //static GLboolean GetWindowIsObscured(const char* WindowName);
00124     //static GLboolean GetWindowIsObscured(GLuint WindowIndex);
00125
00126     //enable vertical sync on selected window
00127     static GLboolean SetWindowSwapInterval(const char* WindowName, GLint
    EnableSync);
00128     static GLboolean SetWindowSwapInterval(GLuint WindowIndex, GLint EnableSync);
00129
00130     //initialize the window manager
00131     static GLboolean Initialize();
00132     static GLboolean IsInitialized();
00133
00134     //ask the window to poll for window events
00135     static GLboolean PollForEvents();
00136
00137     //remove a window from the manager
00138     static GLboolean RemoveWindow(FWindow* WindowToBeRemoved);
00139
00140     //set the style of the given window
00141     static GLboolean SetWindowStyle(const char* WindowName, GLuint WindowStyle);
00142     static GLboolean SetWindowStyle(GLuint WindowIndex, GLuint WindowStyle);
00143
00144     //enable the given decorators of the given window
00145     static GLboolean EnableWindowDecorator(const char* WindowName, GLbitfield
    Decorators);
00146     static GLboolean EnableWindowDecorator(GLuint WindowIndex, GLbitfield
    Decorators);
00147
00148     //disable the given decorators of the given window
00149     static GLboolean DisableWindowDecorator(const char* WindowName, GLbitfield
    Decorators);
00150     static GLboolean DisableWindowDecorator(GLuint WindowIndex, GLbitfield
    Decorators);
00151
00152     //set callbacks for the selected window

```

```

00153     static GLboolean SetWindowOnKeyEvent(const char* WindowName,
00154     OnKeyEvent OnKey);
00154     static GLboolean SetWindowOnKeyEvent(GLuint WindowIndex,
00155     OnKeyEvent OnKey);
00155
00156     static GLboolean SetWindowOnMouseButtonEvent(const char* WindowName,
00157     OnMouseButtonEvent a_OnMouseButtonEvent);
00157     static GLboolean SetWindowOnMouseButtonEvent(GLuint WindowIndex,
00158     OnMouseButtonEvent a_OnMouseButtonEvent);
00158
00159     static GLboolean SetWindowOnMouseWheelEvent(const char* WindowName,
00160     OnMouseWheelEvent OnMouseWheelEvent);
00160     static GLboolean SetWindowOnMouseWheelEvent(GLuint WindowIndex,
00161     OnMouseWheelEvent OnMouseWheelEvent);
00161
00162     static GLboolean SetWindowOnDestroyed(const char* WindowName,
00163     OnDestroyedEvent OnDestroyed);
00163     static GLboolean SetWindowOnDestroyed(GLuint WindowIndex,
00164     OnDestroyedEvent OnDestroyed);
00164
00165     static GLboolean SetWindowOnMaximized(const char* WindowName,
00166     OnMaximizedEvent OnMaximized);
00166     static GLboolean SetWindowOnMaximized(GLuint WindowIndex,
00167     OnMaximizedEvent OnMaximized);
00167
00168     static GLboolean SetWindowOnMinimized(const char* WindowName,
00169     OnMinimizedEvent a_OnMinimized);
00169     static GLboolean SetWindowOnMinimized(GLuint WindowIndex,
00170     OnMinimizedEvent a_OnMinimized);
00170
00171     // static void SetWindowOnRestored(const char* WindowName, OnRestoredEvent OnRestored);
00172     //static void SetWindowOnRestored(GLuint WindowIndex, OnRestoredEvent OnRestored);
00173
00174     static GLboolean SetWindowOnFocus(const char* WindowName,
00175     OnFocusEvent OnFocus);
00175     static GLboolean SetWindowOnFocus(GLuint WindowIndex,
00176     OnFocusEvent OnFocus);
00176
00177     static GLboolean SetWindowOnMoved(const char* WindowName,
00178     OnMovedEvent OnMoved);
00178     static GLboolean SetWindowOnMoved(GLuint WindowIndex,
00179     OnMovedEvent OnMoved);
00179
00180     static GLboolean SetWindowOnResize(const char* WindowName,
00181     OnResizeEvent OnResize);
00181     static GLboolean SetWindowOnResize(GLuint WindowIndex,
00182     OnResizeEvent OnResize);
00182
00183     static GLboolean SetWindowOnMouseMove(const char* WindowName,
00184     OnMouseMoveEvent OnMouseMove);
00184     static GLboolean SetWindowOnMouseMove(GLuint WindowIndex,
00185     OnMouseMoveEvent OnMouseMove);
00185
00186     private:
00187
00188         //make sure the window exists in the window manager
00189         static GLboolean DoesExist(const char* WindowName);
00189         static GLboolean DoesExist(GLuint WindowIndex);
00190
00191         //get a static reference to the window manager
00192         static WindowManager* GetInstance();
00193
00194         std::list<FWindow*> Windows;
00195         static WindowManager* Instance;
00196         GLuint ScreenResolution[2];
00197         GLuint ScreenMousePosition[2];
00198         GLboolean Initialized;
00199
00200 #if defined(CURRENT_OS_WINDOWS)
00201     LRESULT CALLBACK WindowProcedure(HWND WindowHandle, GLuint Message, WPARAM WordParam, LPARAM
00202     LongParam);
00203
00204     static LRESULT CALLBACK StaticWindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam,
00205     LPARAM LongParam);
00206
00207     static FWindow* GetWindowByHandle(HWND WindowHandle);
00208
00209     static GLboolean Windows_PollForEvents();
00210     static GLboolean Windows_Initialize();
00211     static GLboolean Windows_Shutdown();
00212     static GLboolean Windows_SetMousePositionInScreen(GLuint X, GLuint Y);
00213
00214     static void CreateTerminal();
00215     static GLuint Windows_TranslateKey(WPARAM WordParam, LPARAM LongParam);
00216
00217     HDC DeviceContextHandle;
00218     MSG Message;
00219
00220 #endif

```

```

00221
00222 #if defined(CURRENT_OS_LINUX)
00223     static FWindow* GetWindowByHandle(Window WindowHandle);
00224     static FWindow* GetWindowByEvent(XEvent Event);
00225
00226     static GLboolean Linux_Initialize();
00227     static void Linux_Shutdown();
00228
00229     static GLboolean Linux_PollForEvents();
00230     static GLboolean Linux_SetMousePositionInScreen(GLuint X, GLuint Y);
00231     static Display* GetDisplay();
00232
00233     static GLuint Linux_TranslateKey(GLuint KeySym);
00234     static const char* Linux_GetEventType(XEvent Event);
00235
00236     Display* m_Display;
00237     XEvent m_Event;
00238 #endif
00239 };
00240 #endif

```

## 4.17 WindowManager\_Linux.cpp File Reference

```

#include "WindowManager.h"
#include <limits.h>

```

## 4.18 WindowManager\_Linux.cpp

```

00001 /*****
00007 #include "WindowManager.h"
00008
00009 #include <limits.h>
00010 #if defined(CURRENT_OS_LINUX)
00011
00012 /*****
00025 FWindow* WindowManager::GetWindowByHandle(Window WindowHandle)
00026 {
00027     if(GetInstance()->IsInitialized())
00028     {
00029         for (auto Iter : GetInstance()->Windows)
00030         {
00031             if (Iter->GetWindowHandle() == WindowHandle)
00032             {
00033                 return Iter;
00034             }
00035         }
00036         return nullptr;
00037     }
00038     PrintErrorMessage(ERROR_NOTINITIALIZED);
00041     return nullptr;
00042 }
00043
00044 /*****
00053 GLboolean WindowManager::Linux_Initialize()
00054 {
00055     GetInstance()->m_Display = XOpenDisplay(0);
00056     if(!GetInstance()->m_Display)
00057     {
00058         PrintErrorMessage(ERROR_LINUX_CANNOTCONNECTXSERVER);
00059     };
00060     return FOUNDATION_ERROR;
00061 }
00062
00063 GetInstance()->ScreenResolution[0] = WidthOfScreen(XScreenOfDisplay(
GetInstance()->m_Display,
DefaultScreen(GetInstance()->m_Display)));
00064
00065 GetInstance()->ScreenResolution[1] = HeightOfScreen(XScreenOfDisplay(
GetInstance()->m_Display,
DefaultScreen(GetInstance()->m_Display)));
00066
00067 GetInstance()->Initialized = GL_TRUE;
00070
00071     return FOUNDATION_OKAY;

```

```

00072 }
00073
00074 /*****
00086 GLboolean WindowManager::Linux_SetMousePositionInScreen(GLuint X, GLuint Y)
00087 {
00088     if(GetInstance()->IsInitialized())
00089     {
00090         XWarpPointer(GetInstance()->m_Display, None,
00091             XDefaultRootWindow(GetInstance()->m_Display), 0, 0,
00092             GetScreenResolution()[0],
00093             GetScreenResolution()[1],
00094             X, Y);
00095         return FOUNDATION_OKAY;
00096     }
00097
00098     PrintErrorMessage(ERROR_NOTINITIALIZED);
00099     return FOUNDATION_ERROR;
00100 }
00101
00102 /*****
00111 void WindowManager::Linux_Shutdown()
00112 {
00113     XCloseDisplay(GetInstance()->m_Display);
00114 }
00115
00116 /*****
00129 FWindow* WindowManager::GetWindowByEvent(XEvent Event)
00130 {
00131     if(GetInstance()->IsInitialized())
00132     {
00133         switch(Event.type)
00134         {
00135             case Expose:
00136             {
00137                 return GetWindowByHandle(Event.xexpose.window);
00138             }
00139
00140             case DestroyNotify:
00141             {
00142                 return GetWindowByHandle(Event.xdestroywindow.window);
00143             }
00144
00145             case CreateNotify:
00146             {
00147                 return GetWindowByHandle(Event.xcreatewindow.window);
00148             }
00149
00150             case KeyPress:
00151             {
00152                 return GetWindowByHandle(Event.xkey.window);
00153             }
00154
00155             case KeyRelease:
00156             {
00157                 return GetWindowByHandle(Event.xkey.window);
00158             }
00159
00160             case ButtonPress:
00161             {
00162                 return GetWindowByHandle(Event.xbutton.window);
00163             }
00164
00165             case ButtonRelease:
00166             {
00167                 return GetWindowByHandle(Event.xbutton.window);
00168             }
00169
00170             case MotionNotify:
00171             {
00172                 return GetWindowByHandle(Event.xmotion.window);
00173             }
00174
00175             case FocusIn:
00176             {
00177                 return GetWindowByHandle(Event.xfocus.window);
00178             }
00179
00180             case FocusOut:
00181             {
00182                 return GetWindowByHandle(Event.xfocus.window);
00183             }
00184
00185             case ResizeRequest:
00186             {
00187                 return GetWindowByHandle(Event.xresizerequest.window);
00188             }
00189

```

```

00190         case ConfigureNotify:
00191         {
00192             return GetWindowByHandle(Event.xconfigure.window);
00193         }
00194
00195         case PropertyNotify:
00196         {
00197             return GetWindowByHandle(Event.xproperty.window);
00198         }
00199
00200         case GravityNotify:
00201         {
00202             return GetWindowByHandle(Event.xgravity.window);
00203         }
00204
00205         case ClientMessage:
00206         {
00207             return GetWindowByHandle(Event.xclient.window);
00208         }
00209
00210         case VisibilityNotify:
00211         {
00212             return GetWindowByHandle(Event.xvisibility.window);
00213         }
00214
00215         default:
00216         {
00217             return nullptr;
00218         }
00219     }
00220 }
00221 PrintErrorMessage(ERROR_NOTINITIALIZED);
00222 return nullptr;
00223 }
00224
00225 /*****
00236 Display* WindowManager::GetDisplay()
00237 {
00238     return GetInstance()->m_Display;
00239 }
00240
00241 /*****
00250 GLboolean WindowManager::Linux_PollForEvents()
00251 {
00252     if(GetInstance()->IsInitialized())
00253     {
00254         XNextEvent(GetInstance()->m_Display, &GetInstance()->m_Event);
00255
00256         XEvent l_Event = GetInstance()->m_Event;
00257         FWindow* l_Window = GetWindowByEvent(l_Event);
00258
00259         switch (l_Event.type)
00260         {
00261             case Expose:
00262             {
00263                 break;
00264             }
00265
00266             case DestroyNotify:
00267             {
00268
00269                 if(IsValidDestroyedEvent(l_Window->
DestroyedEvent))
00270                 {
00271                     l_Window->DestroyedEvent();
00272                 }
00273
00274                 l_Window->Shutdown();
00275                 break;
00276             }
00277
00278             /*case CreateNotify:
00279             {
00280                 printf("FWindow was created\n");
00281                 l_Window->InitializeGL();
00282
00283                 if(Foundation_Tools::IsValid(l_Window->m_OnCreated))
00284                 {
00285                     l_Window->m_OnCreated();
00286                 }
00287
00288                 break;
00289             }*/
00290
00291             case KeyPress:
00292             {
00293                 GLuint l_FunctionKeysym = XKeycodeToKeysym(

```

```

00294         GetInstance()->m_Display, l_Event.xkey.keycode, 1);
00295
00296     if(l_FunctionKeysym <= 255)
00297     {
00298         l_Window->Keys[l_FunctionKeysym] = KEYSTATE_DOWN;
00299         if(IsValidKeyEvent(l_Window->KeyEvent()))
00300         {
00301             l_Window->KeyEvent(l_FunctionKeysym, KEYSTATE_DOWN);
00302         }
00303     }
00304
00305     else
00306     {
00307         l_Window->Keys[
00308             Linux_TranslateKey(l_FunctionKeysym)] = KEYSTATE_DOWN;
00309
00310         if(IsValidKeyEvent(l_Window->KeyEvent()))
00311         {
00312             l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
00313 KEYSTATE_DOWN);
00314         }
00315     }
00316     break;
00317 }
00318
00319 case KeyRelease:
00320 {
00321     GLboolean l_IsRetriggered = GL_FALSE;
00322     if(XEventsQueued(GetInstance()->m_Display, QueuedAfterReading))
00323     {
00324         XEvent l_NextEvent;
00325         XPeekevent(GetInstance()->m_Display, &l_NextEvent);
00326
00327         if(l_NextEvent.type == KeyPress &&
00328             l_NextEvent.xkey.time == l_Event.xkey.time &&
00329             l_NextEvent.xkey.keycode == l_Event.xkey.keycode)
00330         {
00331             XNextEvent(GetInstance()->m_Display, &l_Event);
00332             l_IsRetriggered = GL_TRUE;
00333         }
00334     }
00335
00336     if(!l_IsRetriggered)
00337     {
00338         GLuint l_FunctionKeysym = XKeycodeToKeysym(GetInstance()->m_Display,
00339             l_Event.xkey.keycode, 1);
00340
00341         if(l_FunctionKeysym <= 255)
00342         {
00343             l_Window->Keys[l_FunctionKeysym] = KEYSTATE_UP;
00344
00345             if(IsValidKeyEvent(l_Window->KeyEvent()))
00346             {
00347                 l_Window->KeyEvent(l_FunctionKeysym, KEYSTATE_UP);
00348             }
00349         }
00350
00351         else
00352         {
00353             l_Window->Keys[
00354                 Linux_TranslateKey(l_FunctionKeysym)] = KEYSTATE_UP;
00355
00356             if(IsValidKeyEvent(l_Window->KeyEvent()))
00357             {
00358                 l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
00359 KEYSTATE_UP);
00360             }
00361         }
00362
00363         if(IsValidKeyEvent(l_Window->KeyEvent()))
00364         {
00365             l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
00366 KEYSTATE_UP);
00367         }
00368     }
00369     break;
00370 }
00371 case ButtonPress:
00372 {
00373     switch(l_Event.xbutton.button)
00374     {
00375         case 1:
00376         {
00377             l_Window->MouseButton[MOUSE_LEFTBUTTON] =

```

```

    MOUSE_BUTTONDOWN;
00378
00379         if(IsValidKeyEvent(l_Window->
MouseButtonEvent))
    {
00380             l_Window->MouseButtonEvent (
00381     MOUSE_LEFTBUTTON, MOUSE_BUTTONDOWN);
00382     }
00383     break;
00384     }
00385
00386     case 2:
00387     {
00388         l_Window->MouseButton[MOUSE_MIDDLEBUTTON] =
    MOUSE_BUTTONDOWN;
00389
00390         if(IsValidKeyEvent(l_Window->
MouseButtonEvent))
    {
00391             l_Window->MouseButtonEvent (
00392     MOUSE_MIDDLEBUTTON, MOUSE_BUTTONDOWN);
00393     }
00394     break;
00395     }
00396
00397     case 3:
00398     {
00399         l_Window->MouseButton[MOUSE_RIGHTBUTTON] =
    MOUSE_BUTTONDOWN;
00400
00401         if(IsValidKeyEvent(l_Window->
MouseButtonEvent))
    {
00402             l_Window->MouseButtonEvent (
00403     MOUSE_RIGHTBUTTON, MOUSE_BUTTONDOWN);
00404     }
00405     break;
00406     }
00407
00408     case 4:
00409     {
00410         l_Window->MouseButton[MOUSE_SCROLL_UP] =
    MOUSE_BUTTONDOWN;
00411
00412         if(IsValidMouseEvent(l_Window->
MouseWheelEvent))
    {
00413             l_Window->MouseWheelEvent (
00414     MOUSE_SCROLL_UP);
00415     }
00416     break;
00417     }
00418
00419     case 5:
00420     {
00421         l_Window->MouseButton[MOUSE_SCROLL_DOWN] =
    MOUSE_BUTTONDOWN;
00422
00423         if(IsValidMouseEvent(l_Window->
MouseWheelEvent))
    {
00424             l_Window->MouseWheelEvent (
00425     MOUSE_SCROLL_DOWN);
00426     }
00427     break;
00428     }
00429
00430     default:
00431     {
00432         break;
00433     }
00434     }
00435     break;
00436 }
00437
00438 case ButtonRelease:
00439 {
00440     switch(l_Event.xbutton.button)
00441     {
00442     case 1:
00443     {
00444         l_Window->MouseButton[MOUSE_LEFTBUTTON] =
    MOUSE_BUTTONUP;
00445
00446         if(IsValidKeyEvent(l_Window->
MouseButtonEvent))

```



```

00448         {
00449             l_Window->MouseButtonEvent (
00450                 MOUSE_LEFTBUTTON, MOUSE_BUTTONUP);
00451             }
00452             break;
00453         }
00454         case 2:
00455         {
00456             l_Window->MouseButton[MOUSE_MIDDLEBUTTON] =
00457                 MOUSE_BUTTONUP;
00458             if(IsValidKeyEvent(l_Window->
00459                 MouseButtonEvent))
00459             {
00460                 l_Window->MouseButtonEvent (
00461                     MOUSE_MIDDLEBUTTON, MOUSE_BUTTONUP);
00462                 }
00463                 break;
00464             }
00465             case 3:
00466             {
00467                 l_Window->MouseButton[MOUSE_RIGHTBUTTON] =
00468                     MOUSE_BUTTONUP;
00469                 if(IsValidKeyEvent(l_Window->
00470                     MouseButtonEvent))
00470                 {
00471                     l_Window->MouseButtonEvent (
00472                         MOUSE_RIGHTBUTTON, MOUSE_BUTTONUP);
00473                     }
00474                     break;
00475                 }
00476                 case 4:
00477                 {
00478                     l_Window->MouseButton[MOUSE_SCROLL_UP] =
00479                         MOUSE_BUTTONDOWN;
00480                     break;
00481                 }
00482                 case 5:
00483                 {
00484                     l_Window->MouseButton[MOUSE_SCROLL_DOWN] =
00485                         MOUSE_BUTTONDOWN;
00486                     break;
00487                 }
00488                 default:
00489                 {
00490                     break;
00491                 }
00492             }
00493             break;
00494         }
00495     }
00496     //when the mouse/pointer device is moved
00497     case MotionNotify:
00498     {
00499         //set the windows mouse position to match the event
00500         l_Window->MousePosition[0] =
00501             l_Event.xmotion.x;
00502
00503         l_Window->MousePosition[1] =
00504             l_Event.xmotion.y;
00505
00506         GetInstance()->ScreenMousePosition[0] = l_Event.xmotion.x_root;
00507         GetInstance()->ScreenMousePosition[1] = l_Event.xmotion.y_root;
00508
00509         if(IsValidMouseMoveEvent(l_Window->
00510             MouseMoveEvent))
00511         {
00512             l_Window->MouseMoveEvent(l_Event.xmotion.x,
00513                 l_Event.xmotion.y, l_Event.xmotion.x_root,
00514                 l_Event.xmotion.y_root);
00515             }
00516             break;
00517         }
00518     }
00519     //when the window goes out of focus
00520     case FocusOut:
00521     {
00522         l_Window->InFocus = GL_FALSE;
00523         if(IsValidFocusEvent(l_Window->FocusEvent))
00524         {
00525             l_Window->FocusEvent (

```

```

00526         l_Window->InFocus);
00527     }
00528     break;
00529 }
00530
00531 //when the window is back in focus (use to restore?)
00532 case FocusIn:
00533 {
00534     l_Window->InFocus = GL_TRUE;
00535
00536     if(IsValidFocusEvent(l_Window->FocusEvent))
00537     {
00538         l_Window->FocusEvent(l_Window->InFocus);
00539     }
00540     break;
00541 }
00542
00543 //when a request to resize the window is made either by
00544 //dragging out the window or programmatically
00545 case ResizeRequest:
00546 {
00547     glViewport(0, 0,
00548         l_Window->GetResolution()[0],
00549         l_Window->GetResolution()[1]);
00550
00551     glMatrixMode(GL_PROJECTION);
00552     glLoadIdentity();
00553
00554     break;
00555 }
00556
00557 //when a request to configure the window is made
00558 case ConfigureNotify:
00559 {
00560     glViewport(0, 0, l_Event.xconfigure.width,
00561         l_Event.xconfigure.height);
00562
00563     //check if window was resized
00564     if((GLuint)l_Event.xconfigure.width != l_Window->Resolution[0]
00565         || (GLuint)l_Event.xconfigure.height != l_Window->Resolution[1])
00566     {
00567         if(IsValidMovedEvent(l_Window->ResizeEvent))
00568         {
00569             l_Window->ResizeEvent(l_Event.xconfigure.width, l_Event.xconfigure.height);
00570         }
00571
00572         l_Window->Resolution[0] = l_Event.xconfigure.width;
00573         l_Window->Resolution[1] = l_Event.xconfigure.height;
00574     }
00575
00576     //check if window was moved
00577     if((GLuint)l_Event.xconfigure.x != l_Window->Position[0]
00578         || (GLuint)l_Event.xconfigure.y != l_Window->Position[1])
00579     {
00580         if(IsValidMovedEvent(l_Window->MovedEvent))
00581         {
00582             l_Window->MovedEvent(l_Event.xconfigure.x, l_Event.xconfigure.y);
00583         }
00584
00585         l_Window->Position[0] = l_Event.xconfigure.x;
00586         l_Window->Position[1] = l_Event.xconfigure.y;
00587     }
00588     break;
00589 }
00590
00591 case PropertyNotify:
00592 {
00593     //this is needed in order to read from the windows WM_STATE Atomic
00594     //to determine if the property notify event was caused by a client
00595     //iconify event(minimizing the window) or a maximise event
00596
00597     Atom l_Type;
00598     GLint l_Format;
00599     ulong l_NumItems, l_BytesAfter;
00600     unsigned char* l_Properties = nullptr;
00601
00602     XGetWindowProperty(WindowManager::GetDisplay(), l_Event.xproperty.window,
00603         l_Window->AtomState,
00604         0, LONG_MAX, GL_FALSE, AnyPropertyType,
00605         &l_Type, &l_Format, &l_NumItems, &l_BytesAfter,
00606         &l_Properties);
00607
00608     if(l_Properties && (l_Format == 32))
00609     {
00610         for(GLuint l_CurrentItem = 0; l_CurrentItem < l_NumItems; l_CurrentItem++)
00611         {
00612             long l_Property = ((long*)(l_Properties))[l_CurrentItem];

```

```

00613
00614         if(l_Property == l_Window->AtomHidden)
00615         {
00616             if(IsValidDestroyedEvent(l_Window->
MinimizedEvent))
00617             {
00618                 l_Window->MinimizedEvent();
00619             }
00620         }
00621
00622         if(l_Property == l_Window->AtomMaxVert ||
00623            l_Property == l_Window->AtomMaxVert)
00624         {
00625             if(IsValidDestroyedEvent(l_Window->
MaximizedEvent))
00626             {
00627                 l_Window->MaximizedEvent();
00628             }
00629         }
00630
00631         if(l_Property == l_Window->AtomDemandsAttention)
00632         {
00633             //printf("window demands attention \n");
00634         }
00635     }
00636 }
00637
00638     break;
00639 }
00640
00641 case GravityNotify:
00642 {
00643     //this is only supposed to pop up when the parent of this window(if any) has something happen
00644     //to it so that this window can react to said event as well.
00645     break;
00646 }
00647
00648 case ClientMessage:
00649 {
00650     const char* l_AtomName = XGetAtomName(WindowManager::GetDisplay(), l_Event.xclient.message_type
);
00651     if(IsValidString(l_AtomName))
00652     {
00653         //print the name of the Atom
00654         //printf("%s\n", l_AtomName);
00655     }
00656
00657     if((Atom)l_Event.xclient.data.l[0] == l_Window->AtomClose)
00658     {
00659         //printf("window closed\n");
00660         l_Window->ShouldClose = GL_TRUE;
00661         l_Window->DestroyedEvent();
00662         l_Window->Shutdown();
00663         //XDestroyWindow(GetInstance()->m_Display, l_Event.xclient.window);
00664         break;
00665     }
00666
00667     if((Atom)l_Event.xclient.data.l[1] == l_Window->AtomFullScreen)
00668     {
00669         //printf("resized window \n");
00670         break;
00671     }
00672     break;
00673 }
00674
00675 /*case VisibilityNotify:
00676 {
00677     if(l_Event.xvisibility.state == VisibilityUnobscured)
00678     {
00679         //printf("window not obscured \n");
00680         l_Window->m_IsObscured = GL_FALSE;
00681     }
00682
00683     else
00684     {
00685         //printf("window obscured\n");
00686         l_Window->m_IsObscured = GL_TRUE;
00687     }
00688 }*/
00689
00690 default:
00691 {
00692     break;
00693 }
00694 }
00695 return FOUNDATION_OKAY;
00696 }

```

```
00697     PrintErrorMessage(ERROR_NOTINITIALIZED);
00698     return FOUNDATION_ERROR;
00699 }
00700 }
00701
00702 const char* WindowManager::Linux_GetEventType(XEvent Event)
00703 {
00704     switch (Event.type)
00705     {
00706         case MotionNotify:
00707         {
00708             return "Motion Notify Event\n";
00709         }
00710
00711         case ButtonPress:
00712         {
00713             return "Button Press Event\n";
00714         }
00715
00716         case ButtonRelease:
00717         {
00718             return "Button Release Event\n";
00719         }
00720
00721         case ColormapNotify:
00722         {
00723             return "Color Map Notify event \n";
00724         }
00725
00726         case EnterNotify:
00727         {
00728             return "Enter Notify Event\n";
00729         }
00730
00731         case LeaveNotify:
00732         {
00733             return "Leave Notify Event\n";
00734         }
00735
00736         case Expose:
00737         {
00738             return "Expose Event\n";
00739         }
00740
00741         case GraphicsExpose:
00742         {
00743             return "Graphics expose event\n";
00744         }
00745
00746         case NoExpose:
00747         {
00748             return "No Expose Event\n";
00749         }
00750
00751         case FocusIn:
00752         {
00753             return "Focus In Event\n";
00754         }
00755
00756         case FocusOut:
00757         {
00758             return "Focus Out Event\n";
00759         }
00760
00761         case KeymapNotify:
00762         {
00763             return "Key Map Notify Event\n";
00764         }
00765
00766         case KeyPress:
00767         {
00768             return "Key Press Event\n";
00769         }
00770
00771         case KeyRelease:
00772         {
00773             return "Key Release Event\n";
00774         }
00775
00776         case PropertyNotify:
00777         {
00778             return "Property Notify Event\n";
00779         }
00780
00781         case ResizeRequest:
00782         {
00783             return "Resize Property Event\n";
```

```

00784     }
00785
00786     case CirculateNotify:
00787     {
00788         return "Circulate Notify Event\n";
00789     }
00790
00791     case ConfigureNotify:
00792     {
00793         return "configure Notify Event\n";
00794     }
00795
00796     case DestroyNotify:
00797     {
00798         return "Destroy Notify Request\n";
00799     }
00800
00801     case GravityNotify:
00802     {
00803         return "Gravity Notify Event \n";
00804     }
00805
00806     case MapNotify:
00807     {
00808         return "Map Notify Event\n";
00809     }
00810
00811     case ReparentNotify:
00812     {
00813         return "Reparent Notify Event\n";
00814     }
00815
00816     case UnmapNotify:
00817     {
00818         return "Unmap notify event\n";
00819     }
00820
00821     case MapRequest:
00822     {
00823         return "Map request event\n";
00824     }
00825
00826     case ClientMessage:
00827     {
00828         return "Client Message Event\n";
00829     }
00830
00831     case MappingNotify:
00832     {
00833         return "Mapping notify event\n";
00834     }
00835
00836     case SelectionClear:
00837     {
00838         return "Selection Clear event\n";
00839     }
00840
00841     case SelectionNotify:
00842     {
00843         return "Selection Notify Event\n";
00844     }
00845
00846     case SelectionRequest:
00847     {
00848         return "Selection Request event\n";
00849     }
00850
00851     case VisibilityNotify:
00852     {
00853         return "Visibility Notify Event\n";
00854     }
00855
00856     default:
00857     {
00858         return 0;
00859     }
00860 }
00861 }
00862
00863 GLuint WindowManager::Linux_TranslateKey(GLuint KeySym)
00864 {
00865     switch (KeySym)
00866     {
00867         case XK_Escape:
00868         {
00869             return KEY_ESCAPE;
00870         }
00871     }

```

```
00871
00872     case XK_Home:
00873     {
00874         return KEY_HOME;
00875     }
00876
00877     case XK_Left:
00878     {
00879         return KEY_ARROW_LEFT;
00880     }
00881
00882     case XK_Right:
00883     {
00884         return KEY_ARROW_RIGHT;
00885     }
00886
00887     case XK_Up:
00888     {
00889         return KEY_ARROW_UP;
00890     }
00891
00892     case XK_Down:
00893     {
00894         return KEY_ARROW_DOWN;
00895     }
00896
00897     case XK_Page_Up:
00898     {
00899         return KEY_PAGEUP;
00900     }
00901
00902     case XK_Page_Down:
00903     {
00904         return KEY_PAGEDOWN;
00905     }
00906
00907     case XK_End:
00908     {
00909         return KEY_END;
00910     }
00911
00912     case XK_Print:
00913     {
00914         return KEY_PRINTSCREEN;
00915     }
00916
00917     case XK_Insert:
00918     {
00919         return KEY_INSERT;
00920     }
00921
00922     case XK_Num_Lock:
00923     {
00924         return KEY_NUMLOCK;
00925     }
00926
00927     case XK_KP_Multiply:
00928     {
00929         return KEY_KEYPAD_MULTIPLY;
00930     }
00931
00932     case XK_KP_Add:
00933     {
00934         return KEY_KEYPAD_ADD;
00935     }
00936
00937     case XK_KP_Subtract:
00938     {
00939         return KEY_KEYPAD_SUBTRACT;
00940     }
00941
00942     case XK_KP_Decimal:
00943     {
00944         return KEY_KEYPAD_PERIOD;
00945     }
00946
00947     case XK_KP_Divide:
00948     {
00949         return KEY_KEYPAD_DIVIDE;
00950     }
00951
00952     case XK_KP_0:
00953     {
00954         return KEY_KEYPAD_0;
00955     }
00956
00957     case XK_KP_1:
```

```
00958     {
00959         return KEY_KEYPAD_1;
00960     }
00961
00962     case XK_KP_2:
00963     {
00964         return KEY_KEYPAD_2;
00965     }
00966
00967     case XK_KP_3:
00968     {
00969         return KEY_KEYPAD_3;
00970     }
00971
00972     case XK_KP_4:
00973     {
00974         return KEY_KEYPAD_4;
00975     }
00976
00977     case XK_KP_5:
00978     {
00979         return KEY_KEYPAD_5;
00980     }
00981
00982     case XK_KP_6:
00983     {
00984         return KEY_KEYPAD_6;
00985     }
00986
00987     case XK_KP_7:
00988     {
00989         return KEY_KEYPAD_7;
00990     }
00991
00992     case XK_KP_8:
00993     {
00994         return KEY_KEYPAD_8;
00995     }
00996
00997     case XK_KP_9:
00998     {
00999         return KEY_KEYPAD_9;
01000     }
01001
01002     case XK_F1:
01003     {
01004         return KEY_F1;
01005     }
01006
01007     case XK_F2:
01008     {
01009         return KEY_F2;
01010     }
01011
01012     case XK_F3:
01013     {
01014         return KEY_F3;
01015     }
01016
01017     case XK_F4:
01018     {
01019         return KEY_F4;
01020     }
01021
01022     case XK_F5:
01023     {
01024         return KEY_F5;
01025     }
01026
01027     case XK_F6:
01028     {
01029         return KEY_F6;
01030     }
01031
01032     case XK_F7:
01033     {
01034         return KEY_F7;
01035     }
01036
01037     case XK_F8:
01038     {
01039         return KEY_F8;
01040     }
01041
01042     case XK_F9:
01043     {
01044         return KEY_F9;
```

```
01045     }
01046
01047     case XK_F10:
01048     {
01049         return KEY_F10;
01050     }
01051
01052     case XK_F11:
01053     {
01054         return KEY_F11;
01055     }
01056
01057     case XK_F12:
01058     {
01059         return KEY_F12;
01060     }
01061
01062     case XK_Shift_L:
01063     {
01064         return KEY_LEFTSHIFT;
01065     }
01066
01067     case XK_Shift_R:
01068     {
01069         return KEY_RIGHTSHIFT;
01070     }
01071
01072     case XK_Control_R:
01073     {
01074         return KEY_RIGHTCONTROL;
01075     }
01076
01077     case XK_Control_L:
01078     {
01079         return KEY_LEFTCONTROL;
01080     }
01081
01082     case XK_Caps_Lock:
01083     {
01084         return KEY_CAPSLOCK;
01085     }
01086
01087     case XK_Alt_L:
01088     {
01089         return KEY_LEFTALT;
01090     }
01091
01092     case XK_Alt_R:
01093     {
01094         return KEY_RIGHTALT;
01095     }
01096
01097     default:
01098     {
01099         return 0;
01100     }
01101 }
01102 }
01103
01104 #endif
```

#### 4.19 WindowManager\_Windows.cpp File Reference

```
#include "WindowManager.h"
```

## 4.20 WindowManager\_Windows.cpp

```
00001 /*****
00007 #include "WindowManager.h"
00008
00009 #if defined(CURRENT_OS_WINDOWS)
00010
00011 /*****
00020 GLboolean WindowManager::Windows_Initialize()
00021 {
00022     CreateTerminal();
00023     RECT l_Desktop;
```



```

00024
00025     HWND l_DesktopHandle = GetDesktopWindow();
00026
00027
00028     if (l_DesktopHandle)
00029     {
00030         GetWindowRect(l_DesktopHandle, &l_Desktop);
00031
00032         GetInstance()->ScreenResolution[0] = l_Desktop.right;
00033         GetInstance()->ScreenResolution[1] = l_Desktop.bottom;
00034
00035         GetInstance()->Initialized = GL_TRUE;
00036         return FOUNDATION_OKAY;
00037     }
00038
00039     PrintErrorMessage(ERROR_WINDOWS_CANNOTINITIALIZE);
00040     return FOUNDATION_ERROR;
00041 }
00042
00043 /*****
00056 FWindow* WindowManager::GetWindowByHandle(HWND WindowHandle)
00057 {
00058     if(GetInstance()->IsInitialized())
00059     {
00060 #if defined(CURRENT_OS_WINDOWS)
00061
00062         for each(auto CurrentWindow in GetInstance()->Windows)
00063         {
00064             if (CurrentWindow->WindowHandle == WindowHandle)
00065             {
00066                 return CurrentWindow;
00067             }
00068         }
00069 #endif
00070 #endif
00071
00072 #if defined(CURRENT_OS_LINUX)
00073         for (auto CurrentWindow : GetInstance()->Windows)
00074         {
00075             if (CurrentWindow->WindowHandle == WindowHandle)
00076             {
00077                 return CurrentWindow;
00078             }
00079         }
00080 #endif
00081 #endif
00082
00083     return nullptr;
00084 }
00085
00086     return nullptr;
00087 }
00088
00089 LRESULT CALLBACK WindowManager::WindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam, LPARAM
LongParam)
00090 {
00091     FWindow* l_Window = GetWindowByHandle(WindowHandle);
00092     switch (Message)
00093     {
00094     case WM_CREATE:
00095     {
00096         GetWindowByIndex(GetInstance()->Windows.size() - 1)->WindowHandle
= WindowHandle;
00097         l_Window = GetWindowByHandle(WindowHandle);
00098
00099         l_Window->InitializeGL();
00100         break;
00101     }
00102
00103     case WM_DESTROY:
00104     {
00105         l_Window->ShouldClose = GL_TRUE;
00106
00107         if (IsValidDestroyedEvent(l_Window->DestroyedEvent))
00108         {
00109             l_Window->DestroyedEvent();
00110         }
00111
00112         l_Window->Shutdown();
00113         return 0;
00114     }
00115
00116     case WM_MOVE:
00117     {
00118         l_Window->Position[0] = LOWORD(LongParam);
00119         l_Window->Position[1] = HIWORD(LongParam);
00119
00120         if (IsValidMovedEvent(l_Window->MovedEvent))

```

```

00121         {
00122             l_Window->MovedEvent(l_Window->Position[0], l_Window->Position[1]);
00123         }
00124
00125         break;
00126     }
00127
00128     case WM_MOVING:
00129     {
00130         l_Window->Position[0] = LOWORD(LongParam);
00131         l_Window->Position[1] = HIWORD(LongParam);
00132
00133         if (IsValidMovedEvent(l_Window->MovedEvent))
00134         {
00135             l_Window->MovedEvent(l_Window->Position[0], l_Window->Position[1]);
00136         }
00137         break;
00138     }
00139
00140     case WM_SIZE:
00141     {
00142         l_Window->Resolution[0] = (GLuint)LOWORD(LongParam);
00143         l_Window->Resolution[1] = (GLuint)HIWORD(LongParam);
00144
00145         switch (WordParam)
00146         {
00147             case SIZE_MAXIMIZED:
00148             {
00149                 if (IsValidDestroyedEvent(l_Window->MaximizedEvent))
00150                 {
00151                     l_Window->MaximizedEvent();
00152                 }
00153
00154                 break;
00155             }
00156
00157             case SIZE_MINIMIZED:
00158             {
00159                 if (IsValidDestroyedEvent(l_Window->MinimizedEvent))
00160                 {
00161                     l_Window->MinimizedEvent();
00162                 }
00163                 break;
00164             }
00165
00166             /*case SIZE_RESTORED:
00167             {
00168                 if (IsValid(l_Window->MaximizedEvent))
00169                 {
00170                     l_Window->RestoredEvent();
00171                 }
00172                 break;
00173             }*/
00174
00175             default:
00176             {
00177                 if (IsValidMovedEvent(l_Window->ResizeEvent))
00178                 {
00179                     l_Window->ResizeEvent(l_Window->Resolution[0],
00180                                           l_Window->Resolution[1]);
00181                 }
00182
00183                 break;
00184             }
00185         }
00186         break;
00187     }
00188
00189     case WM_SIZING:
00190     {
00191
00192         l_Window->Resolution[0] = (GLuint)LOWORD(LongParam);
00193         l_Window->Resolution[1] = (GLuint)HIWORD(LongParam);
00194
00195         if (IsValidMovedEvent(l_Window->ResizeEvent))
00196         {
00197             l_Window->ResizeEvent(l_Window->Resolution[0],
00198                                   l_Window->Resolution[1]);
00199         }
00200
00201
00202         break;
00203     }
00204
00205     case WM_KEYDOWN:
00206     {
00207         GLuint l_TranslatedKey = 0;

```

```

00208
00209         switch (HIWORD(LongParam))
00210         {
00211             case 29:
00212             {
00213                 l_Window->Keys[KEY_LEFTCONTROL] = KEYSTATE_DOWN;
00214                 l_TranslatedKey = KEY_LEFTCONTROL;
00215                 break;
00216             }
00217
00218             case 285:
00219             {
00220                 l_Window->Keys[KEY_RIGHTCONTROL] = KEYSTATE_DOWN;
00221                 l_TranslatedKey = KEY_RIGHTCONTROL;
00222                 break;
00223             }
00224
00225             case 42:
00226             {
00227                 l_Window->Keys[KEY_LEFTSHIFT] = KEYSTATE_DOWN;
00228                 l_TranslatedKey = KEY_LEFTSHIFT;
00229                 break;
00230             }
00231
00232             case 54:
00233             {
00234                 l_Window->Keys[KEY_RIGHTSHIFT] = KEYSTATE_DOWN;
00235                 l_TranslatedKey = KEY_RIGHTSHIFT;
00236                 break;
00237             }
00238
00239             default:
00240             {
00241                 l_TranslatedKey = Windows_TranslateKey(WordParam, LongParam);
00242                 l_Window->Keys[l_TranslatedKey] = KEYSTATE_DOWN;
00243                 break;
00244             }
00245         }
00246
00247         if (IsValidKeyEvent(l_Window->KeyEvent))
00248         {
00249             l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_DOWN);
00250         }
00251         break;
00252     }
00253
00254     case WM_KEYUP:
00255     {
00256         GLuint l_TranslatedKey = 0;
00257
00258         switch (HIWORD(LongParam))
00259         {
00260             case 49181:
00261             {
00262                 l_Window->Keys[KEY_LEFTCONTROL] = KEYSTATE_UP;
00263                 l_TranslatedKey = KEY_LEFTCONTROL;
00264                 break;
00265             }
00266
00267             case 49437:
00268             {
00269                 l_Window->Keys[KEY_RIGHTCONTROL] = KEYSTATE_UP;
00270                 l_TranslatedKey = KEY_RIGHTCONTROL;
00271                 break;
00272             }
00273
00274             case 49194:
00275             {
00276                 l_Window->Keys[KEY_LEFTSHIFT] = KEYSTATE_UP;
00277                 l_TranslatedKey = KEY_LEFTSHIFT;
00278                 break;
00279             }
00280
00281             case 49206:
00282             {
00283                 l_Window->Keys[KEY_RIGHTSHIFT] = KEYSTATE_UP;
00284                 l_TranslatedKey = KEY_RIGHTSHIFT;
00285                 break;
00286             }
00287
00288             default:
00289             {
00290                 l_TranslatedKey = Windows_TranslateKey(WordParam, LongParam);
00291                 l_Window->Keys[l_TranslatedKey] = KEYSTATE_UP;
00292                 break;
00293             }
00294         }

```

```

00295
00296     if (IsValidKeyEvent(l_Window->KeyEvent))
00297     {
00298         l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_UP);
00299     }
00300     break;
00301 }
00302
00303 case WM_SYSKEYDOWN:
00304 {
00305     GLuint l_TranslatedKey = 0;
00306     switch (HIWORD(LongParam))
00307     {
00308     case 8248:
00309     {
00310         l_Window->Keys[KEY_LEFTALT] = KEYSTATE_DOWN;
00311         l_TranslatedKey = KEY_LEFTALT;
00312         break;
00313     }
00314
00315     case 8504:
00316     {
00317         l_Window->Keys[KEY_RIGHTALT] = KEYSTATE_DOWN;
00318         l_TranslatedKey = KEY_RIGHTALT;
00319     }
00320
00321     default:
00322     {
00323         break;
00324     }
00325     }
00326
00327     if (IsValidKeyEvent(l_Window->KeyEvent))
00328     {
00329         l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_DOWN);
00330     }
00331
00332     break;
00333 }
00334
00335 case WM_SYSKEYUP:
00336 {
00337     GLuint l_TranslatedKey = 0;
00338     switch (HIWORD(LongParam))
00339     {
00340     case 49208:
00341     {
00342         l_Window->Keys[KEY_LEFTALT] = KEYSTATE_UP;
00343         l_TranslatedKey = KEY_LEFTALT;
00344         break;
00345     }
00346
00347     case 49464:
00348     {
00349         l_Window->Keys[KEY_RIGHTALT] = KEYSTATE_UP;
00350         l_TranslatedKey = KEY_RIGHTALT;
00351         break;
00352     }
00353
00354     default:
00355     {
00356         break;
00357     }
00358
00359     }
00360
00361     if (IsValidKeyEvent(l_Window->KeyEvent))
00362     {
00363         l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_UP);
00364     }
00365
00366     break;
00367 }
00368
00369 case WM_MOUSEMOVE:
00370 {
00371     l_Window->MousePosition[0] = (GLuint)LOWORD(LongParam);
00372     l_Window->MousePosition[1] = (GLuint)HIWORD(LongParam);
00373
00374     POINT l_Point;
00375     l_Point.x = l_Window->MousePosition[0];
00376     l_Point.y = l_Window->MousePosition[1];
00377
00378     ClientToScreen(WindowHandle, &l_Point);
00379
00380     //printf("%i %i \n", l_Point.x, l_Point.y);
00381

```

```

00382         if (IsValidMouseMoveEvent (l_Window->MouseMoveEvent))
00383         {
00384             l_Window->MouseMoveEvent (l_Window->MousePosition[0],
00385                                     l_Window->MousePosition[1], l_Point.x, l_Point.y);
00386         }
00387         break;
00388     }
00389
00390     case WM_LBUTTONDOWN:
00391     {
00392         l_Window->MouseButton [MOUSE_LEFTBUTTON] =
MOUSE_BUTTONDOWN;
00393
00394         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00395         {
00396             l_Window->MouseButtonEvent (MOUSE_LEFTBUTTON,
MOUSE_BUTTONDOWN);
00397         }
00398         break;
00399     }
00400
00401     case WM_LBUTTONUP:
00402     {
00403         l_Window->MouseButton [MOUSE_LEFTBUTTON] = MOUSE_BUTTONUP;
00404
00405         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00406         {
00407             l_Window->MouseButtonEvent (MOUSE_LEFTBUTTON,
MOUSE_BUTTONUP);
00408         }
00409         break;
00410     }
00411
00412     case WM_RBUTTONDOWN:
00413     {
00414         l_Window->MouseButton [MOUSE_RIGHTBUTTON] =
MOUSE_BUTTONDOWN;
00415
00416         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00417         {
00418             l_Window->MouseButtonEvent (MOUSE_RIGHTBUTTON,
MOUSE_BUTTONDOWN);
00419         }
00420         break;
00421     }
00422
00423     case WM_RBUTTONUP:
00424     {
00425         l_Window->MouseButton [MOUSE_RIGHTBUTTON] =
MOUSE_BUTTONUP;
00426
00427         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00428         {
00429             l_Window->MouseButtonEvent (MOUSE_RIGHTBUTTON,
MOUSE_BUTTONUP);
00430         }
00431         break;
00432     }
00433
00434     case WM_MBUTTONDOWN:
00435     {
00436         l_Window->MouseButton [MOUSE_MIDDLEBUTTON] =
MOUSE_BUTTONDOWN;
00437
00438         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00439         {
00440             l_Window->MouseButtonEvent (MOUSE_MIDDLEBUTTON,
MOUSE_BUTTONDOWN);
00441         }
00442         break;
00443     }
00444
00445     case WM_MBUTTONUP:
00446     {
00447         l_Window->MouseButton [MOUSE_MIDDLEBUTTON] =
MOUSE_BUTTONUP;
00448
00449         if (IsValidKeyEvent (l_Window->MouseButtonEvent))
00450         {
00451             l_Window->MouseButtonEvent (MOUSE_MIDDLEBUTTON,
MOUSE_BUTTONUP);
00452         }
00453         break;
00454     }
00455
00456     case WM_MOUSEWHEEL:
00457     {

```

```

00458         if ((WordParam % WHEEL_DELTA) > 0)
00459         {
00460             if (IsValidMouseEvent(l_Window->MouseEvent))
00461             {
00462                 l_Window->MouseEvent(MOUSE_SCROLL_DOWN);
00463             }
00464         }
00465     else
00466     {
00467         if (IsValidMouseEvent(l_Window->MouseEvent))
00468         {
00469             l_Window->MouseEvent(MOUSE_SCROLL_UP);
00470         }
00471     }
00472 }
00473 break;
00474 }
00475 }
00476 }
00477 default:
00478 {
00479     return DefWindowProc(WindowHandle, Message, WordParam, LongParam);
00480 }
00481 }
00482 return 0;
00483 }
00484
00485 LRESULT CALLBACK WindowManager::StaticWindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam,
LPARAM LongParam)
00486 {
00487     return WindowManager::GetInstance()->WindowProcedure(WindowHandle, Message,
WordParam, LongParam);
00488 }
00489
00490 GLboolean WindowManager::Windows_PollForEvents()
00491 {
00492     if (GetInstance()->IsInitialized())
00493     {
00494         GetMessage(&GetInstance()->Message, 0, 0, 0);
00495         TranslateMessage(&GetInstance()->Message);
00496         DispatchMessage(&GetInstance()->Message);
00497         return FOUNDATION_OKAY;
00498     }
00499     else
00500     {
00501         PrintErrorMessage(ERROR_NOTINITIALIZED);
00502         return FOUNDATION_ERROR;
00503     }
00504 }
00505 }
00506 }
00507
00508 void WindowManager::CreateTerminal()
00509 {
00510     int hConHandle;
00511     long lStdHandle;
00512     FILE *fp;
00513
00514     // allocate a console for this app
00515     AllocConsole();
00516
00517     // redirect unbuffered STDOUT to the console
00518     lStdHandle = (long)GetStdHandle(STD_OUTPUT_HANDLE);
00519     hConHandle = _open_osfhandle(lStdHandle, _O_TEXT);
00520     fp = _fdopen(hConHandle, "w");
00521     *stdout = *fp;
00522
00523     setvbuf(stdout, nullptr, _IONBF, 0);
00524 }
00525
00526 GLboolean WindowManager::Windows_SetMousePositionInScreen(GLuint X, GLuint Y)
00527 {
00528     if (GetInstance()->IsInitialized())
00529     {
00530         POINT l_MousePoint;
00531         l_MousePoint.x = X;
00532         l_MousePoint.y = Y;
00533         SetCursorPos(l_MousePoint.x, l_MousePoint.y);
00534     }
00535
00536     PrintErrorMessage(ERROR_NOTINITIALIZED);
00537     return FOUNDATION_ERROR;
00538 }
00539
00540 GLuint WindowManager::Windows_TranslateKey(WPARAM WordParam, LPARAM LongParam)
00541 {
00542     switch (WordParam)

```

```
00543     {
00544         case VK_ESCAPE:
00545         {
00546             return KEY_ESCAPE;
00547         }
00548
00549         case VK_F1:
00550         {
00551             return KEY_F1;
00552         }
00553
00554         case VK_F2:
00555         {
00556             return KEY_F2;
00557         }
00558
00559         case VK_F3:
00560         {
00561             return KEY_F3;
00562         }
00563
00564         case VK_F4:
00565         {
00566             return KEY_F4;
00567         }
00568
00569         case VK_F5:
00570         {
00571             return KEY_F5;
00572         }
00573
00574         case VK_F6:
00575         {
00576             return KEY_F6;
00577         }
00578
00579         case VK_F7:
00580         {
00581             return KEY_F7;
00582         }
00583
00584         case VK_F8:
00585         {
00586             return KEY_F8;
00587         }
00588
00589         case VK_F9:
00590         {
00591             return KEY_F9;
00592         }
00593
00594         case VK_F10:
00595         {
00596             return KEY_F10;
00597         }
00598
00599         case VK_F11:
00600         {
00601             return KEY_F11;
00602         }
00603
00604         case VK_F12:
00605         {
00606             return KEY_F12;
00607         }
00608
00609         case VK_BACK:
00610         {
00611             return KEY_BACKSPACE;
00612         }
00613
00614         case VK_TAB:
00615         {
00616             return KEY_TAB;
00617         }
00618
00619         case VK_CAPITAL:
00620         {
00621             return KEY_CAPSLOCK;
00622         }
00623
00624         case VK_RETURN:
00625         {
00626             return KEY_ENTER;
00627         }
00628
00629         case VK_PRINT:
```

```
00630     {
00631         return KEY_PRINTSCREEN;
00632     }
00633
00634     case VK_SCROLL:
00635     {
00636         return KEY_SCROLLLOCK;
00637     }
00638
00639     case VK_PAUSE:
00640     {
00641         return KEY_PAUSE;
00642     }
00643
00644     case VK_INSERT:
00645     {
00646         return KEY_INSERT;
00647     }
00648
00649     case VK_HOME:
00650     {
00651         return KEY_HOME;
00652     }
00653
00654     case VK_DELETE:
00655     {
00656         return KEY_DELETE;
00657     }
00658
00659     case VK_END:
00660     {
00661         return KEY_END;
00662     }
00663
00664     case VK_PRIOR:
00665     {
00666         return KEY_PAGEUP;
00667     }
00668
00669     case VK_NEXT:
00670     {
00671         return KEY_PAGEDOWN;
00672     }
00673
00674     case VK_DOWN:
00675     {
00676         return KEY_ARROW_DOWN;
00677     }
00678
00679     case VK_UP:
00680     {
00681         return KEY_ARROW_UP;
00682     }
00683
00684     case VK_LEFT:
00685     {
00686         return KEY_ARROW_LEFT;
00687     }
00688
00689     case VK_RIGHT:
00690     {
00691         return KEY_ARROW_RIGHT;
00692     }
00693
00694     case VK_DIVIDE:
00695     {
00696         return KEY_KEYPAD_DIVIDE;
00697     }
00698
00699     case VK_MULTIPLY:
00700     {
00701         return KEY_KEYPAD_MULTIPLY;
00702     }
00703
00704     case VK_SUBTRACT:
00705     {
00706         return KEY_KEYPAD_DIVIDE;
00707     }
00708
00709     case VK_ADD:
00710     {
00711         return KEY_KEYPAD_ADD;
00712     }
00713
00714     case VK_DECIMAL:
00715     {
00716         return KEY_KEYPAD_PERIOD;
```



```
00717     }
00718
00719     case VK_NUMPAD0:
00720     {
00721         return KEY_KEYPAD_0;
00722     }
00723
00724     case VK_NUMPAD1:
00725     {
00726         return KEY_KEYPAD_1;
00727     }
00728
00729     case VK_NUMPAD2:
00730     {
00731         return KEY_KEYPAD_2;
00732     }
00733
00734     case VK_NUMPAD3:
00735     {
00736         return KEY_KEYPAD_3;
00737     }
00738
00739     case VK_NUMPAD4:
00740     {
00741         return KEY_KEYPAD_4;
00742     }
00743
00744     case VK_NUMPAD5:
00745     {
00746         return KEY_KEYPAD_5;
00747     }
00748
00749     case VK_NUMPAD6:
00750     {
00751         return KEY_KEYPAD_6;
00752     }
00753
00754     case VK_NUMPAD7:
00755     {
00756         return KEY_KEYPAD_7;
00757     }
00758
00759     case VK_NUMPAD8:
00760     {
00761         return KEY_KEYPAD_8;
00762     }
00763
00764     case VK_NUMPAD9:
00765     {
00766         return KEY_KEYPAD_9;
00767     }
00768
00769     case VK_LWIN:
00770     {
00771         return KEY_LEFTWINDOW;
00772     }
00773
00774     case VK_RWIN:
00775     {
00776         return KEY_RIGHTWINDOW;
00777     }
00778
00779     default:
00780     {
00781         return WordParam;
00782     }
00783 }
00784 }
00785 #endif
```

# Index

- ~FWindow
  - FWindow, [9](#)
- ~WindowManager
  - WindowManager, [46](#)
- AddWindow
  - WindowManager, [46](#)
- ColourBits
  - FWindow, [37](#)
- ContextCreated
  - FWindow, [37](#)
- CurrentState
  - FWindow, [37](#)
- CurrentSwapInterval
  - FWindow, [38](#)
- CurrentWindowStyle
  - FWindow, [38](#)
- DECORATOR\_BORDER
  - WindowAPI\_Defs.h, [127](#)
- DECORATOR\_ICON
  - WindowAPI\_Defs.h, [127](#)
- DepthBits
  - FWindow, [38](#)
- DestroyedEvent
  - FWindow, [38](#)
- DisableDecorator
  - FWindow, [9](#)
- DisableWindowDecorator
  - WindowManager, [47](#)
- DoesExist
  - WindowManager, [47](#), [48](#)
- ERROR\_NOCONTEXT
  - WindowAPI\_Defs.h, [130](#)
- EXTSwapControlSupported
  - FWindow, [38](#)
- EnableDecorator
  - FWindow, [9](#)
- EnableWindowDecorator
  - WindowManager, [49](#)
- Example.cpp, [95](#), [96](#)
  - main, [95](#)
  - OnWindowKeyPressed, [96](#)
- FOUNDATION\_ERROR
  - WindowAPI\_Defs.h, [131](#)
- FOUNDATION\_OKAY
  - WindowAPI\_Defs.h, [131](#)
- FWindow, [5](#)
- ~FWindow, [9](#)
- ColourBits, [37](#)
- ContextCreated, [37](#)
- CurrentState, [37](#)
- CurrentSwapInterval, [38](#)
- CurrentWindowStyle, [38](#)
- DepthBits, [38](#)
- DestroyedEvent, [38](#)
- DisableDecorator, [9](#)
- EXTSwapControlSupported, [38](#)
- EnableDecorator, [9](#)
- FWindow, [8](#)
- Focus, [10](#)
- FocusEvent, [38](#)
- FullScreen, [10](#)
- FWindow, [8](#)
- GetContextHasBeenCreated, [11](#)
- GetCurrentState, [12](#)
- GetInFocus, [12](#)
- GetIsCurrentContext, [12](#)
- GetIsFullScreen, [13](#)
- GetIsMaximized, [13](#)
- GetIsMinimized, [14](#)
- GetKeyState, [14](#)
- GetMousePosition, [15](#)
- GetOpenGLExtensions, [16](#)
- GetOpenGLVersion, [16](#)
- GetPosition, [16](#), [17](#)
- GetResolution, [17](#), [18](#)
- GetShouldClose, [18](#)
- GetWindowName, [19](#)
- ID, [38](#)
- InFocus, [38](#)
- InitGLExtensions, [19](#)
- Initialize, [20](#)
- InitializeEvents, [20](#)
- InitializeGL, [21](#)
- Initialized, [39](#)
- IsCurrentContext, [39](#)
- KeyEvent, [39](#)
- Keys, [39](#)
- MESASwapControlSupported, [39](#)
- MakeCurrentContext, [21](#)
- Maximize, [22](#)
- MaximizedEvent, [39](#)
- Minimize, [23](#)
- MinimizedEvent, [39](#)
- MouseButton, [39](#)
- MouseButtonEvent, [40](#)

- MouseMoveEvent, [40](#)
- MousePosition, [40](#)
- MouseWheelEvent, [40](#)
- MovedEvent, [40](#)
- Name, [40](#)
- Position, [40](#)
- PrintOpenGLExtensions, [23](#)
- PrintOpenGLVersion, [24](#)
- ResizeEvent, [40](#)
- Resolution, [41](#)
- Restore, [24](#)
- SGISwapControlSupported, [41](#)
- SetCurrentState, [25](#)
- SetIcon, [26](#)
- SetMousePosition, [26](#)
- SetOnDestroyed, [27](#)
- SetOnFocus, [27](#)
- SetOnKeyEvent, [28](#)
- SetOnMaximized, [29](#)
- SetOnMinimized, [29](#)
- SetOnMouseButtonEvent, [30](#)
- SetOnMouseMove, [30](#)
- SetOnMouseWheelEvent, [31](#)
- SetOnMoved, [32](#)
- SetOnResize, [32](#)
- SetPosition, [33](#)
- SetResolution, [33](#)
- SetStyle, [34](#)
- SetSwapInterval, [34](#)
- SetTitleBar, [35](#)
- ShouldClose, [41](#)
- Shutdown, [36](#)
- StencilBits, [41](#)
- SwapDrawBuffers, [36](#)
- WindowManager, [37](#)
- WindowManager, [93](#)
- Focus
  - FWindow, [10](#)
- FocusEvent
  - FWindow, [38](#)
- FocusWindow
  - WindowManager, [49](#), [50](#)
- FullScreen
  - FWindow, [10](#)
- GetContextHasBeenCreated
  - FWindow, [11](#)
- GetCurrentState
  - FWindow, [12](#)
- GetInFocus
  - FWindow, [12](#)
- GetInstance
  - WindowManager, [50](#)
- GetIsCurrentContext
  - FWindow, [12](#)
- GetIsFullScreen
  - FWindow, [13](#)
- GetIsMaximized
  - FWindow, [13](#)
- GetIsMinimized
  - FWindow, [14](#)
- GetKeyState
  - FWindow, [14](#)
- GetMousePosition
  - FWindow, [15](#)
- GetMousePositionInScreen
  - WindowManager, [51](#), [52](#)
- GetMousePositionInWindow
  - WindowManager, [52–54](#)
- GetNumWindows
  - WindowManager, [54](#)
- GetOpenGLExtensions
  - FWindow, [16](#)
- GetOpenGLVersion
  - FWindow, [16](#)
- GetPosition
  - FWindow, [16](#), [17](#)
- GetResolution
  - FWindow, [17](#), [18](#)
- GetScreenResolution
  - WindowManager, [55](#)
- GetShouldClose
  - FWindow, [18](#)
- GetWindowByIndex
  - WindowManager, [56](#)
- GetWindowByName
  - WindowManager, [57](#)
- GetWindowIndex
  - WindowManager, [58](#)
- GetWindowsFullScreen
  - WindowManager, [59](#)
- GetWindowsInFocus
  - WindowManager, [60](#)
- GetWindowsMaximized
  - WindowManager, [61](#)
- GetWindowsMinimized
  - WindowManager, [62](#)
- GetWindowName
  - FWindow, [19](#)
  - WindowManager, [63](#)
- GetWindowPosition
  - WindowManager, [63–65](#)
- GetWindowResolution
  - WindowManager, [65–67](#)
- GetWindowShouldClose
  - WindowManager, [67](#), [68](#)
- ID
  - FWindow, [38](#)
- InFocus
  - FWindow, [38](#)
- InitGLExtensions
  - FWindow, [19](#)
- Initialize
  - FWindow, [20](#)
  - WindowManager, [68](#)
- InitializeEvents
  - FWindow, [20](#)

InitializeGL  
     FWindow, [21](#)  
 Initialized  
     FWindow, [39](#)  
     WindowManager, [93](#)  
 Instance  
     WindowManager, [93](#)  
 IsCurrentContext  
     FWindow, [39](#)  
 IsInitialized  
     WindowManager, [69](#)  
 IsValidDestroyedEvent  
     WindowAPI\_Defs.h, [142](#)  
 IsValidFocusEvent  
     WindowAPI\_Defs.h, [142](#)  
 IsValidKeyEvent  
     WindowAPI\_Defs.h, [142](#)  
 IsValidMouseMoveEvent  
     WindowAPI\_Defs.h, [142](#)  
 IsValidMouseWheelEvent  
     WindowAPI\_Defs.h, [142](#)  
 IsValidMovedEvent  
     WindowAPI\_Defs.h, [143](#)  
 IsValidString  
     WindowAPI\_Defs.h, [143](#)  
  
 KEY\_ARROW\_DOWN  
     WindowAPI\_Defs.h, [131](#)  
 KEY\_ARROW\_LEFT  
     WindowAPI\_Defs.h, [131](#)  
 KEY\_ARROW\_RIGHT  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_ARROW\_UP  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_BACKSPACE  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_CAPSLOCK  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_DELETE  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_END  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_ENTER  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_ERROR  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_ESCAPE  
     WindowAPI\_Defs.h, [132](#)  
 KEY\_F1  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F10  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F11  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F12  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F2  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F3  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F4  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F5  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F6  
     WindowAPI\_Defs.h, [133](#)  
 KEY\_F7  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_F8  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_F9  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_FIRST  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_HOME  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_INSERT  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_KEYPAD\_0  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_KEYPAD\_1  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_KEYPAD\_2  
     WindowAPI\_Defs.h, [134](#)  
 KEY\_KEYPAD\_3  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_4  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_5  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_6  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_7  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_8  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_9  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_ADD  
     WindowAPI\_Defs.h, [135](#)  
 KEY\_KEYPAD\_ENTER  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_LAST  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_LEFTALT  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_LEFTCONTROL  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_LEFTSHIFT  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_LEFTWINDOW  
     WindowAPI\_Defs.h, [136](#)  
 KEY\_NUMLOCK  
     WindowAPI\_Defs.h, [137](#)  
 KEY\_PAGEDOWN  
     WindowAPI\_Defs.h, [137](#)  
 KEY\_PAGEUP

- WindowAPI\_Defs.h, [137](#)
- KEY\_PAUSE
  - WindowAPI\_Defs.h, [137](#)
- KEY\_PRINTSCREEN
  - WindowAPI\_Defs.h, [137](#)
- KEY\_RIGHTALT
  - WindowAPI\_Defs.h, [137](#)
- KEY\_RIGHTCONTROL
  - WindowAPI\_Defs.h, [137](#)
- KEY\_RIGHTSHIFT
  - WindowAPI\_Defs.h, [137](#)
- KEY\_RIGHTWINDOW
  - WindowAPI\_Defs.h, [137](#)
- KEY\_SCROLLLOCK
  - WindowAPI\_Defs.h, [138](#)
- KEY\_TAB
  - WindowAPI\_Defs.h, [138](#)
- KEYSTATE\_DOWN
  - WindowAPI\_Defs.h, [138](#)
- KEYSTATE\_UP
  - WindowAPI\_Defs.h, [138](#)
- KeyEvent
  - FWindow, [39](#)
- Keys
  - FWindow, [39](#)
- LINUX\_DECORATOR
  - WindowAPI\_Defs.h, [138](#)
- LINUX\_FUNCTION
  - WindowAPI\_Defs.h, [139](#)
- MESASwapControlSupported
  - FWindow, [39](#)
- MOUSE\_BUTTONDOWN
  - WindowAPI\_Defs.h, [139](#)
- MOUSE\_BUTTONUP
  - WindowAPI\_Defs.h, [139](#)
- MOUSE\_LAST
  - WindowAPI\_Defs.h, [139](#)
- MOUSE\_LEFTBUTTON
  - WindowAPI\_Defs.h, [139](#)
- MOUSE\_RIGHTBUTTON
  - WindowAPI\_Defs.h, [139](#)
- MOUSE\_SCROLL\_UP
  - WindowAPI\_Defs.h, [139](#)
- main
  - Example.cpp, [95](#)
- MakeCurrentContext
  - FWindow, [21](#)
- Maximize
  - FWindow, [22](#)
- MaximizeWindow
  - WindowManager, [69, 70](#)
- MaximizedEvent
  - FWindow, [39](#)
- Minimize
  - FWindow, [23](#)
- MinimizeWindow
  - WindowManager, [70, 71](#)
- MinimizedEvent
  - FWindow, [39](#)
- MouseButton
  - FWindow, [39](#)
- MouseButtonEvent
  - FWindow, [40](#)
- MouseMoveEvent
  - FWindow, [40](#)
- MousePosition
  - FWindow, [40](#)
- MouseWheelEvent
  - FWindow, [40](#)
- MovedEvent
  - FWindow, [40](#)
- Name
  - FWindow, [40](#)
- OnDestroyedEvent
  - WindowAPI\_Defs.h, [141](#)
- OnFocusEvent
  - WindowAPI\_Defs.h, [141](#)
- OnKeyEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMaximizedEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMinimizedEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMouseButtonEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMouseMoveEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMouseWheelEvent
  - WindowAPI\_Defs.h, [141](#)
- OnMovedEvent
  - WindowAPI\_Defs.h, [141](#)
- OnResizeEvent
  - WindowAPI\_Defs.h, [142](#)
- OnWindowKeyPressed
  - Example.cpp, [96](#)
- PollForEvents
  - WindowManager, [71](#)
- Position
  - FWindow, [40](#)
- PrintErrorMessage
  - WindowAPI\_Defs.h, [143](#)
- PrintOpenGLExtensions
  - FWindow, [23](#)
- PrintOpenGLVersion
  - FWindow, [24](#)
- PrintWarningMessage
  - WindowAPI\_Defs.h, [145](#)
- RemoveWindow
  - WindowManager, [72](#)
- ResizeEvent
  - FWindow, [40](#)
- Resolution

- FWindow, [41](#)
- Restore
  - FWindow, [24](#)
- RestoreWindow
  - WindowManager, [72](#)
- SGISwapControlSupported
  - FWindow, [41](#)
- ScreenMousePosition
  - WindowManager, [93](#)
- ScreenResolution
  - WindowManager, [93](#)
- SetCurrentState
  - FWindow, [25](#)
- SetFullScreen
  - WindowManager, [73](#)
- SetIcon
  - FWindow, [26](#)
- SetMousePosition
  - FWindow, [26](#)
- SetMousePositionInScreen
  - WindowManager, [74](#)
- SetMousePositionInWindow
  - WindowManager, [74](#), [75](#)
- SetOnDestroyed
  - FWindow, [27](#)
- SetOnFocus
  - FWindow, [27](#)
- SetOnKeyEvent
  - FWindow, [28](#)
- SetOnMaximized
  - FWindow, [29](#)
- SetOnMinimized
  - FWindow, [29](#)
- SetOnMouseButtonEvent
  - FWindow, [30](#)
- SetOnMouseMove
  - FWindow, [30](#)
- SetOnMouseWheelEvent
  - FWindow, [31](#)
- SetOnMoved
  - FWindow, [32](#)
- SetOnResize
  - FWindow, [32](#)
- SetPosition
  - FWindow, [33](#)
- SetResolution
  - FWindow, [33](#)
- SetStyle
  - FWindow, [34](#)
- SetSwapInterval
  - FWindow, [34](#)
- SetTitleBar
  - FWindow, [35](#)
- SetWindowIcon
  - WindowManager, [75](#)
- SetWindowOnDestroyed
  - WindowManager, [75](#), [76](#)
- SetWindowOnFocus
  - WindowManager, [76](#), [77](#)
- SetWindowOnKeyEvent
  - WindowManager, [77](#), [78](#)
- SetWindowOnMaximized
  - WindowManager, [78](#), [79](#)
- SetWindowOnMinimized
  - WindowManager, [79](#), [80](#)
- SetWindowOnMouseButtonEvent
  - WindowManager, [80](#), [81](#)
- SetWindowOnMouseMove
  - WindowManager, [81](#), [82](#)
- SetWindowOnMouseWheelEvent
  - WindowManager, [82](#), [83](#)
- SetWindowOnMoved
  - WindowManager, [83](#), [84](#)
- SetWindowOnResize
  - WindowManager, [84](#), [85](#)
- SetWindowPosition
  - WindowManager, [85](#), [86](#)
- SetWindowResolution
  - WindowManager, [86](#), [87](#)
- SetWindowStyle
  - WindowManager, [87](#), [88](#)
- SetWindowSwapInterval
  - WindowManager, [88](#)
- SetWindowTitleBar
  - WindowManager, [89](#)
- SetWindowIcon
  - WindowManager, [75](#)
- ShouldClose
  - FWindow, [41](#)
- ShutDown
  - WindowManager, [90](#)
- Shutdown
  - FWindow, [36](#)
- StencilBits
  - FWindow, [41](#)
- SwapDrawBuffers
  - FWindow, [36](#)
- WINDOWSTYLE\_BARE
  - WindowAPI\_Defs.h, [140](#)
- WINDOWSTYLE\_POPUP
  - WindowAPI\_Defs.h, [140](#)
- Window.cpp, [97](#)
- Window.h, [107](#)
- Window\_Linux.cpp, [112](#)
- Window\_Windows.cpp, [119](#)
- WindowAPI\_Defs.h, [124](#), [146](#)
  - DECORATOR\_ICON, [127](#)
  - IsValidDestroyedEvent, [142](#)
  - IsValidFocusEvent, [142](#)
  - IsValidKeyEvent, [142](#)
  - IsValidMouseMoveEvent, [142](#)
  - IsValidMouseWheelEvent, [142](#)
  - IsValidMovedEvent, [143](#)
  - IsValidString, [143](#)
  - KEY\_ARROW\_UP, [132](#)
  - KEY\_BACKSPACE, [132](#)

- KEY\_CAPSLOCK, [132](#)
- KEY\_DELETE, [132](#)
- KEY\_END, [132](#)
- KEY\_ENTER, [132](#)
- KEY\_ERROR, [132](#)
- KEY\_ESCAPE, [132](#)
- KEY\_F1, [133](#)
- KEY\_F10, [133](#)
- KEY\_F11, [133](#)
- KEY\_F12, [133](#)
- KEY\_F2, [133](#)
- KEY\_F3, [133](#)
- KEY\_F4, [133](#)
- KEY\_F5, [133](#)
- KEY\_F6, [133](#)
- KEY\_F7, [134](#)
- KEY\_F8, [134](#)
- KEY\_F9, [134](#)
- KEY\_FIRST, [134](#)
- KEY\_HOME, [134](#)
- KEY\_INSERT, [134](#)
- KEY\_KEYPAD\_0, [134](#)
- KEY\_KEYPAD\_1, [134](#)
- KEY\_KEYPAD\_2, [134](#)
- KEY\_KEYPAD\_3, [135](#)
- KEY\_KEYPAD\_4, [135](#)
- KEY\_KEYPAD\_5, [135](#)
- KEY\_KEYPAD\_6, [135](#)
- KEY\_KEYPAD\_7, [135](#)
- KEY\_KEYPAD\_8, [135](#)
- KEY\_KEYPAD\_9, [135](#)
- KEY\_LAST, [136](#)
- KEY\_LEFTALT, [136](#)
- KEY\_LEFTSHIFT, [136](#)
- KEY\_LEFTWINDOW, [136](#)
- KEY\_NUMLOCK, [137](#)
- KEY\_PAGEDOWN, [137](#)
- KEY\_PAGEUP, [137](#)
- KEY\_PAUSE, [137](#)
- KEY\_RIGHTALT, [137](#)
- KEY\_RIGHTSHIFT, [137](#)
- KEY\_SCROLLLOCK, [138](#)
- KEY\_TAB, [138](#)
- KEYSTATE\_DOWN, [138](#)
- KEYSTATE\_UP, [138](#)
- LINUX\_FUNCTION, [139](#)
- MOUSE\_BUTTONUP, [139](#)
- MOUSE\_LAST, [139](#)
- OnDestroyedEvent, [141](#)
- OnFocusEvent, [141](#)
- OnKeyEvent, [141](#)
- OnMaximizedEvent, [141](#)
- OnMinimizedEvent, [141](#)
- OnMouseButtonEvent, [141](#)
- OnMouseMoveEvent, [141](#)
- OnMouseWheelEvent, [141](#)
- OnMovedEvent, [141](#)
- OnResizeEvent, [142](#)
- PrintErrorMessage, [143](#)
- PrintWarningMessage, [145](#)
- WindowGetKey
- WindowManager, [91](#)
- WindowManager, [41](#)
  - ~WindowManager, [46](#)
  - AddWindow, [46](#)
  - DisableWindowDecorator, [47](#)
  - DoesExist, [47](#), [48](#)
  - EnableWindowDecorator, [49](#)
  - FWindow, [93](#)
  - FocusWindow, [49](#), [50](#)
  - FWindow, [37](#)
  - GetInstance, [50](#)
  - GetMousePositionInScreen, [51](#), [52](#)
  - GetMousePositionInWindow, [52–54](#)
  - GetNumWindows, [54](#)
  - GetScreenResolution, [55](#)
  - GetWindowByIndex, [56](#)
  - GetWindowByName, [57](#)
  - GetWindowIndex, [58](#)
  - GetWindowsIsFullScreen, [59](#)
  - GetWindowsIsInFocus, [60](#)
  - GetWindowsIsMaximized, [61](#)
  - GetWindowsIsMinimized, [62](#)
  - GetWindowName, [63](#)
  - GetWindowPosition, [63–65](#)
  - GetWindowResolution, [65–67](#)
  - GetWindowShouldClose, [67](#), [68](#)
  - Initialize, [68](#)
  - Initialized, [93](#)
  - Instance, [93](#)
  - IsInitialized, [69](#)
  - MaximizeWindow, [69](#), [70](#)
  - MinimizeWindow, [70](#), [71](#)
  - PollForEvents, [71](#)
  - RemoveWindow, [72](#)
  - RestoreWindow, [72](#)
  - ScreenMousePosition, [93](#)
  - ScreenResolution, [93](#)
  - SetFullScreen, [73](#)
  - SetMousePositionInScreen, [74](#)
  - SetMousePositionInWindow, [74](#), [75](#)
  - SetWindowIcon, [75](#)
  - SetWindowOnDestroyed, [75](#), [76](#)
  - SetWindowOnFocus, [76](#), [77](#)
  - SetWindowOnKeyEvent, [77](#), [78](#)
  - SetWindowOnMaximized, [78](#), [79](#)
  - SetWindowOnMinimized, [79](#), [80](#)
  - SetWindowOnMouseButtonEvent, [80](#), [81](#)
  - SetWindowOnMouseMove, [81](#), [82](#)
  - SetWindowOnMouseWheelEvent, [82](#), [83](#)
  - SetWindowOnMoved, [83](#), [84](#)
  - SetWindowOnResize, [84](#), [85](#)
  - SetWindowPosition, [85](#), [86](#)
  - SetWindowResolution, [86](#), [87](#)
  - SetWindowStyle, [87](#), [88](#)
  - SetWindowSwapInterval, [88](#)

- SetWindowTitleBar, [89](#)
- SetwindowIcon, [75](#)
- ShutDown, [90](#)
- WindowGetKey, [91](#)
- WindowManager, [46](#)
- WindowSwapBuffers, [92](#)
- WindowManager, [46](#)
- Windows, [93](#)
- WindowManager.cpp, [150](#)
- WindowManager.h, [164](#), [165](#)
- WindowManager\_Linux.cpp, [168](#)
- WindowManager\_Windows.cpp, [180](#)
- WindowSwapBuffers
  - WindowManager, [92](#)
- Windows
  - WindowManager, [93](#)