Foundation window management API

0.3

Generated by Doxygen 1.8.6

Wed Jan 21 2015 22:42:21

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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 foe 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

WindowName	Name of the window.
Width	The width.
Height	The height.
ColourBits	The colour bits.
DepthBits	The depth bits.
StencilBits	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.

```
                           :
    Name(WindowName),
    ColourBits(ColourBits),
    DepthBits(DepthBits),
    StencilBits(StencilBits)
{
    Resolution[0] = Width;
    Resolution[1] = Height;
    Position[0] = 0;
    Position[1] = 0;
    ShouldClose = GL_FALSE;
    EXTSwapControlSupported = GL_FALSE;
    SGISwapControlSupported = GL_FALSE;
    MESASwapControlSupported = GL_FALSE;

    if(!IsValidString(WindowName))
    {
        PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
        exit(0);
    }

    InitializeEvents();

    CurrentState = WINDOWSTATE_NORMAL;
    ContextCreated = GL_FALSE;
    IsCurrentContext = GL_FALSE;
}
```

3.1.2.2 FWindow::∼FWindow ( )

FWindow Destructor.

Author

Ziyad

Date

29/11/2014

Definition at line 73 of file Window.cpp.

References Shutdown().

```
{
    Shutdown();
}
```

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 Print-ErrorMessage().

Referenced by WindowManager::DisableWindowDecorator().

```
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_DisableDecorator(Decorator);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_DisableDecorator(Decorator);
#endif
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_EnableDecorator(Decorator);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_EnableDecorator(Decorator);
#endif

        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

put the window into event focus.

Author

Ziyad

Date

29/11/2014

Parameters

NewState	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().

```
{
    if (ContextCreated)
    {
        InFocus = ShouldBeInFocus;

#if defined(CURRENT_OS_LINUX)
        Linux_Focus(ShouldBeInFocus);
#endif

#if defined(CURRENT_OS_WINDOWS)
        Windows_Focus();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

toggle the full screen mode for the window.

Author

Ziyad

Date

29/11/2014

Parameters

ShouldBe-Fullscreen	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(), SetCurrent-State(), and WindowManager::SetFullScreen().

```
{
    if(ContextCreated)
    {
        if (ShouldBeFullscreen)
        {
            CurrentState = WINDOWSTATE_FULLSCREEN;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_LINUX)
        Linux_FullScreen(ShouldBeFullscreen);
#endif

#if defined(CURRENT_OS_WINDOWS)
        Windows_FullScreen();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_OKAY;
}
```

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.

```
{
    return ContextCreated;
}
```

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.

```
{
    return CurrentState;
}
```

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::GetWindowIsInFocus().

```
{
    return InFocus;
}
```

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().

```
{
    if(ContextCreated)
    {
        return IsCurrentContext;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return GL_FALSE;
}
```

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().

```
{
    if(ContextCreated)
    {
        return (CurrentState == WINDOWSTATE_FULLSCREEN);
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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::GetWindowIsMaximized().

```
{
    return (CurrentState == WINDOWSTATE_MAXIMIZED) ;
}
```

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::GetWindowIsMinimized().

```
{
    return (CurrentState == WINDOWSTATE_MINIMIZED);
}
```

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

Gets key state.

Author

Ziyad

Date

29/11/2014

Parameters

Key	The key.

Returns

The key state.

Definition at line 182 of file Window.cpp.

References Keys.

Referenced by WindowManager::WindowGetKey().

```
{
    return Keys[Key];
}
```

3.1.3.13 GLboolean FWindow::GetMousePosition (GLuint & *X*, GLuint & *Y*)

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

Author

Ziyad

Date

29/11/2014

Parameters

in,out	*X*	The X position of the mouse.
in,out	*Y*	The Y position of the mouse.

< .

Definition at line 667 of file Window.cpp.

References ContextCreated, ERROR_NOCONTEXT, FOUNDATION_ERROR, FOUNDATION_OKAY, Mouse-Position, and PrintErrorMessage().

Referenced by WindowManager::GetMousePositionInWindow().

```
{
    if (ContextCreated)
    {
        X = MousePosition[0];
        Y = MousePosition[1];
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (ContextCreated)
    {
        return MousePosition;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}
```

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().

```
{
    if(ContextCreated)
    {
        return (const char*)glGetString(GL_EXTENSIONS);
    }

    else
    {
        PrintErrorMessage(ERROR_NOCONTEXT);
        return nullptr;
    }
}
```

3.1.3.16 const char ∗ FWindow::GetOpenGLVersion ( )

Definition at line 1019 of file Window.cpp.

References ContextCreated, ERROR_NOCONTEXT, and PrintErrorMessage().

```
{
    if(ContextCreated)
    {
        return (const char*)glGetString(GL_VERSION);
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}
```

3.1.3.17 GLboolean FWindow::GetPosition (GLuint & *X*, GLuint & *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().

```
{
    if (ContextCreated)
    {
        X = Position[0];
        Y = Position[1];

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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.

```
{
    return Position;
}
```

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`	*Width*	The width.
`in,out`	*Height*	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().

```
{
    return ShouldClose;
}
```

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().

```
{
    if (ContextCreated)
    {
        return Name;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}
```

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.

```
{
#if defined(CURRENT_OS_WINDOWS)
    Windows_InitGLExtensions();
#endif

#if defined(CURRENT_OS_LINUX)
    Linux_InitGLExtensions();
#endif
}
```

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().

```
{
#if defined(CURRENT_OS_WINDOWS)
    return Windows_Initialize();
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_Initialize();
#endif
}
```

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().

```
{
    KeyEvent = nullptr;
    MouseButtonEvent = nullptr;
    MouseWheelEvent = nullptr;
    DestroyedEvent = nullptr;
    MaximizedEvent = nullptr;
    MinimizedEvent = nullptr;
//  RestoredEvent = nullptr;
    MovedEvent = nullptr;
    MouseMoveEvent = nullptr;
}
```

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.

```
{
#if defined(CURRENT_OS_WINDOWS)
    return Windows_InitializeGL();
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_InitializeGL();
#endif
}
```

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 foe 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, IsCurrent-Context, and PrintErrorMessage().

Referenced by main().

```
{
    if(ContextCreated)
    {
        IsCurrentContext = true;
#if defined(CURRENT_OS_WINDOWS)
        wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);
#endif

#if defined(CURRENT_OS_LINUX)
        glXMakeCurrent(WindowManager::GetDisplay(), WindowHandle, Context);
#endif
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

Maximizes the window depending on New state.

Author

Ziyad

Date

29/11/2014

Parameters

NewState	Whether to minimize the window.

Returns

A GLboolean.

Definition at line 493 of file Window.cpp.

References ContextCreated, CurrentState, ERROR_NOCONTEXT, FOUNDATION_ERROR, FOUNDATION_OK-AY, PrintErrorMessage(), WINDOWSTATE_MAXIMIZED, and WINDOWSTATE_NORMAL.

Referenced by Restore(), and SetCurrentState().

```
{
    if(ContextCreated)
    {
        if(NewState)
        {
            CurrentState = WINDOWSTATE_MAXIMIZED;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_WINDOWS)
         Windows_Maximize();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Maximize(NewState);
#endif
         return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

NewState	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().

```
{
    if(ContextCreated)
    {
        if(NewState)
        {
            CurrentState = WINDOWSTATE_MINIMIZED;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_WINDOWS)
        Windows_Minimize();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Minimize(NewState);
#endif

        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}
```

3.1.3.30 GLboolean FWindow::PrintOpenGLExtensions ( )

Definition at line 1029 of file Window.cpp.

References ContextCreated, ERROR_NOCONTEXT, FOUNDATION_ERROR, FOUNDATION_OKAY, and Print-ErrorMessage().

```
{
    if(ContextCreated)
    {
        printf("%s \n", (const char*)glGetString(GL_EXTENSIONS));
        return FOUNDATION_OKAY;
    }

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}
```

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 Print-ErrorMessage().

```
{
    if(ContextCreated)
    {
        printf("%s\n", glGetString(GL_VERSION));
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (ContextCreated)
    {
        switch (CurrentState)
        {
        case WINDOWSTATE_MAXIMIZED:
        {
            Maximize(GL_FALSE);
            break;
        }

        case WINDOWSTATE_FULLSCREEN:
        {
            FullScreen(GL_FALSE);
            break;
        }
        }

        CurrentState = WINDOWSTATE_NORMAL;
#if defined(CURRENT_OS_WINDOWS)
        Windows_Restore();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Restore();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

Sets the current state of the window.

Author

Ziyad

Date

29/11/2014

Parameters

NewState	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 
00310     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 }
```

3.1.3.34 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 }
```

3.1.3.35 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, Mouse-Position, and PrintErrorMessage().

Referenced by WindowManager::SetMousePositionInWindow().

```
{
    if (ContextCreated)
    {
        MousePosition[0] = X;
        MousePosition[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetMousePosition(X, Y);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetMousePosition(X, Y);
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

Sets on destroyed.

Author

Ziyad

Date

29/11/2014

Parameters

OnDestroyed	The on destroyed event.

Returns

A GLboolean.

Definition at line 1202 of file Window.cpp.

References DestroyedEvent, ERROR_INVALIDEVENT, FOUNDATION_ERROR, FOUNDATION_OKAY, IsValid-DestroyedEvent(), and PrintErrorMessage().

Referenced by WindowManager::SetWindowOnDestroyed().

```
{
    if(IsValidDestroyedEvent(OnDestroyed))
    {
        DestroyedEvent = OnDestroyed;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on focus.

Author

Ziyad

Date

29/11/2014

Parameters

OnFocus	The on focus event.

Returns

A GLboolean.

Definition at line 1284 of file Window.cpp.

References ERROR_INVALIDEVENT, FocusEvent, FOUNDATION_ERROR, FOUNDATION_OKAY, IsValidFocus-Event(), and PrintErrorMessage().

```
{
    if(IsValidFocusEvent(OnFocus))
    {
        FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on key event.

Author

Ziyad

Date

29/11/2014

Parameters

OnKey	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().

```
{
    if (IsValidKeyEvent(OnKey))
    {
        KeyEvent = OnKey;
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}
```

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

Sets on maximized.

Author

Ziyad

Date

29/11/2014

Parameters

OnMaximized	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().

```
{
    if(IsValidDestroyedEvent(OnMaximized))
    {
        MaximizedEvent = OnMaximized;
        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on minimized.

Author

Ziyad

Date

29/11/2014

Parameters

OnMinimized	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().

```
{
    if(IsValidDestroyedEvent(OnMinimized))
    {
        MinimizedEvent = OnMinimized;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on mouse button event.

Author

Ziyad

Date

29/11/2014

Parameters

OnMouseButton-Event	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().

```
{
    //we don't really need to check if the context has been created
    if(IsValidKeyEvent(OnMouseButtonEvent))
    {
        MouseButtonEvent = OnMouseButtonEvent;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on mouse move.

Author

Ziyad

Date

29/11/2014

Parameters

OnMouseMove	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().

```
{
    if(IsValidMouseMoveEvent(OnMouseMove))
    {
        MouseMoveEvent = OnMouseMove;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on mouse wheel event.

Author

Ziyad

Date

29/11/2014

Parameters

OnMouseWheel	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().

```
{
    if(IsValidMouseWheelEvent(OnMouseWheel))
    {
        MouseWheelEvent = OnMouseWheel;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on moved.

Author

Ziyad

Date

29/11/2014

Parameters

OnMoved	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().

```
{
    if(IsValidMovedEvent(OnMoved))
    {
        MovedEvent = OnMoved;
        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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

Sets on resize.

Author

Ziyad

Date

29/11/2014

Parameters

OnResize	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().

```
{
    if(IsValidMovedEvent(OnResize))
    {
        ResizeEvent = OnResize;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (ContextCreated)
    {
        Position[0] = X;
        Position[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetPosition(Position[0], Position[1]);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetPosition(X, Y);
#endif
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

Sets the resolution of the window.

Author

Ziyad

Date

29/11/2014

Parameters

Width	The new width of the window.
Height	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().

```
{
    if (ContextCreated)
    {
        if(Width > 0 && Height > 0)
        {
            Resolution[0] = Width;
            Resolution[1] = Height;

#if defined(CURRENT_OS_WINDOWS)
            Windows_SetResolution(Resolution[0], Resolution[1]);
#endif

#if defined(CURRENT_OS_LINUX)
            Linux_SetResolution(Width, Height);
#endif

            glViewport(0, 0, Resolution[0], Resolution[1]);

            return FOUNDATION_OKAY;
        }

        else
        {
            PrintErrorMessage(ERROR_INVALIDRESOLUTION);
            return FOUNDATION_ERROR;
        }
    }
        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}
```

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

Definition at line 889 of file Window.cpp.

References ContextCreated, ERROR_NOCONTEXT, FOUNDATION_ERROR, FOUNDATION_OKAY, and Print-ErrorMessage().

Referenced by WindowManager::SetWindowStyle().

```
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetStyle(WindowType);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetStyle(WindowType);
#endif

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}
```

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

Sets swap interval(V-sync).

Author

Ziyad

Date

29/11/2014

Parameters

SwapSetting	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().

```
{
    if(ContextCreated)
    {
    CurrentSwapInterval = SwapSetting;
#if defined(CURRENT_OS_WINDOWS)
    Windows_VerticalSync(SwapSetting);
#endif

#if defined(CURRENT_OS_LINUX)
    Linux_VerticalSync(SwapSetting);
#endif

    return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

Sets title bar.

Author

Ziyad

Date

29/11/2014

Parameters

NewTitle	The new title bar of the window.

Definition at line 844 of file Window.cpp.

References ContextCreated, ERROR_INVALIDTITLEBAR, ERROR_NOCONTEXT, FOUNDATION_ERROR, FOUNDATION_OKAY, and PrintErrorMessage().

Referenced by WindowManager::SetWindowTitleBar().

```
{
    if (ContextCreated)
    {
        if(NewTitle != nullptr)
        {
#if defined(CURRENT_OS_LINUX)
            Linux_SetTitleBar(NewTitle);
#endif

#if defined(CURRENT_OS_WINDOWS)
            Windows_SetTitleBar(NewTitle);
#endif
            return FOUNDATION_OKAY;
        }

        else
        {
            PrintErrorMessage(ERROR_INVALIDTITLEBAR);
            return FOUNDATION_ERROR;
        }
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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 Print-ErrorMessage().

Referenced by ∼FWindow().

```
{
    if(ContextCreated)
    {

#if defined (CURRENT_OS_WINDOWS)
        Windows_Shutdown();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Shutdown();
#endif
        ContextCreated = GL_FALSE;
        return FOUNDATION_OKAY;
    }

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}
```

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().

```
{
    if(ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        SwapBuffers(DeviceContextHandle);
#endif

#if defined(CURRENT_OS_LINUX)
        glXSwapBuffers(WindowManager::GetDisplay(), WindowHandle);
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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(), GetIs-Minimized(), 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 FWindoa 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_On-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 OnMouse-Move)

  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().

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

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.

```
{
    if (!GetInstance()->Windows.empty())
    {
#if defined(CURRENT_OS_WINDOWS)
        for each(auto CurrentWindow in GetInstance()->Windows)
        {
            delete CurrentWindow;
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            delete CurrentWindow;
        }
#endif
        GetInstance()->Windows.clear();
    }
}
```

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

in,out	*NewWindow*	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().

```
{
    if (GetInstance()->IsInitialized())
    {
        if (NewWindow != nullptr)
        {
            GetInstance()->Windows.push_back(NewWindow);
            NewWindow->ID = GetInstance()->Windows.size() - 1;
            NewWindow->Initialize();
            return GetInstance();
        }
        PrintErrorMessage(ERROR_INVALIDWINDOW);
        return nullptr;
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}
```

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().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->DisableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->DisableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}
```

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

Does the window exist.

Author

Ziyad

Date

30/11/2014

Parameters

WindowName	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().

```
{
    if (GetInstance()->IsInitialized())
    {
        if (IsValidString(WindowName))
        {
#if defined(CURRENT_OS_WINDOWS)
            for each(auto iter in GetInstance()->Windows)
            {
                if (iter->Name == WindowName)
                {
                    return GL_TRUE;
                }
            }
#endif

#if defined(CURRENT_OS_LINUX)
            for (auto iter : GetInstance()->Windows)
            {
                if (iter->Name == WindowName)
                {
                    return GL_TRUE;
                }
            }
#endif
        }
        PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
        return GL_FALSE;
    }
    return GL_FALSE;
}
```

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

Does the window exist.

Author

Ziyad

Date

30/11/2014

Parameters

WindowIndex	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.

```
{
    if (GetInstance()->IsInitialized())
    {
        if (WindowIndex <= (GetInstance()->Windows.size() - 1))
        {
            return FOUNDATION_OKAY;
        }

        PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
        return FOUNDATION_ERROR;
    }
    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->EnableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;

}
```

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().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->EnableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}
```

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

Focus window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
ShouldBe-Focused	Whether the window should be in event focus.

Definition at line 1518 of file WindowManager.cpp.

References DoesExist(), FWindow::Focus(), FOUNDATION_ERROR, and GetWindowByName().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->Focus(ShouldBeFocused);
    }

    return FOUNDATION_ERROR;
}
```

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

Focus window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
ShouldBe-Focused	Whether the window should be in event focus.

Definition at line 1540 of file WindowManager.cpp.

References DoesExist(), FWindow::Focus(), FOUNDATION_ERROR, and GetWindowByIndex().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->Focus(ShouldBeFocused);
    }

    return FOUNDATION_ERROR;
}
```

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().

```
{
    if(!WindowManager::Instance)
    {
        WindowManager::Instance = new WindowManager();
        return WindowManager::Instance;
    }

    else
    {
        return WindowManager::Instance;
    }
}
```

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

Gets mouse position in screen.

Author

Ziyad

Date

29/11/2014

Parameters

in,out	*X*	The X coordinate of the mouse relative to screen position.
in,out	*Y*	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.

```
{
    if (GetInstance()->IsInitialized())
    {
        X = GetInstance()->ScreenMousePosition[0];
        Y = GetInstance()->ScreenMousePosition[1];
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;

}
```

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.

```
{
    if (GetInstance()->IsInitialized())
    {
        return GetInstance()->ScreenMousePosition;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}
```

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

	WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

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

Gets mouse position in window.

Author

Ziyad

Date

29/11/2014

Parameters

	WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.15 GLuint ∗ WindowManager::GetMousePositionInWindow (const char ∗ *WindowName*) [static]

Gets mouse position in window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetMousePosition();
    }

    return nullptr;
}
```

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

Gets mouse position in window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetMousePosition();
    }
    PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
    return nullptr;
}
```

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(), PrintError-Message(), and Windows.

Referenced by main().

```
{
    if(GetInstance()->IsInitialized())
    {
        return GetInstance()->Windows.size();
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}
```

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 Screen-Resolution[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.

```
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)
        RECT l_Screen;
        HWND m_Desktop = GetDesktopWindow();
        GetWindowRect(m_Desktop, &l_Screen);

        GetInstance()->ScreenResolution[0] = l_Screen.right;
        GetInstance()->ScreenResolution[1] = l_Screen.bottom;
        return GetInstance()->ScreenResolution;

#endif

#if defined(CURRENT_OS_LINUX)
        GetInstance()->ScreenResolution[0] = WidthOfScreen(
    XDefaultScreenOfDisplay(GetInstance()->m_Display));
        GetInstance()->ScreenResolution[1] = HeightOfScreen(
    XDefaultScreenOfDisplay(GetInstance()->m_Display));

        return GetInstance()->ScreenResolution;
#endif
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;

}
```

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

Gets screen resolution.

Author

Ziyad

Date

29/11/2014

Parameters

in,out	*Width*	The width.
in,out	*Height*	The height.

Definition at line 500 of file WindowManager.cpp.

References ERROR_NOTINITIALIZED, FOUNDATION_ERROR, FOUNDATION_OKAY, GetInstance(), IsInitialized(), PrintErrorMessage(), and ScreenResolution.

```
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)

        RECT l_Screen;
        HWND m_Desktop = GetDesktopWindow();
        GetWindowRect(m_Desktop, &l_Screen);
        Width = l_Screen.right;
        Height = l_Screen.bottom;
#endif

#if defined(CURRENT_OS_LINUX)

        Width = WidthOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
        Height = HeightOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));


        GetInstance()->ScreenResolution[0] = Width;
        GetInstance()->ScreenResolution[1] = Height;
#endif

        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}
```

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

Gets window by index.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if (DoesExist(WindowIndex))
    {
#if defined(CURRENT_OS_WINDOWS)
        for each (auto CurrentWindow in GetInstance()->Windows)
        {
            if (CurrentWindow->ID == WindowIndex)
            {
                return CurrentWindow;
            }
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            if(CurrentWindow->ID == WindowIndex)
            {
                return CurrentWindow;
            }
        }
#endif
        PrintErrorMessage(ERROR_WINDOWNOTFOUND);
        return nullptr;
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	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(), GetWindowIsFullScreen(), GetWindowIsInFocus(), GetWindowIsMaximized(), GetWindowIsMinimized(), 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().

```
{
    if (DoesExist(WindowName))
    {
#if defined(CURRENT_OS_WINDOWS)
        for each(auto CurrentWindow in GetInstance()->Windows)
        {
            if (CurrentWindow->Name == WindowName)
            {
                return CurrentWindow;
            }
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            if (CurrentWindow->Name == WindowName)
            {
                return CurrentWindow;
            }
        }
#endif
        PrintErrorMessage(ERROR_WINDOWNOTFOUND);
        return nullptr;
    }
    return nullptr;
}
```

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

Gets window index.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.

Returns

The window index.

Definition at line 1406 of file WindowManager.cpp.

References DoesExist(), GetWindowByName(), and FWindow::ID.

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->ID;
    }

    return 0;
}
```

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

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsFullScreen();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets whether the window is full screen.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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.

```
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->GetIsFullScreen();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is in focus.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetInFocus();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is in focus.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetInFocus();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is maximized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsMaximized();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is maximized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetIsMaximized();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is minimized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsMinimized();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window is minimized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetIsMinimized();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets window name.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetWindowName();
    }

    return nullptr;
}
```

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

	WindowName	Name of the window.
in,out	*X*	The X coordinate of the window relative to screen coordinates.
in,out	*Y*	The Y coordinate of the window relative to screen coordinates.

Definition at line 693 of file WindowManager.cpp.

References DoesExist(), FWindow::GetPosition(), and GetWindowByName().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetPosition(X, Y);
    }

    return GL_FALSE;
}
```

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

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

Definition at line 716 of file WindowManager.cpp.

References DoesExist(), FWindow::GetPosition(), and GetWindowByIndex().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetPosition(X, Y);
    }

    return GL_FALSE;
}
```

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

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetPosition();
    }

    return nullptr;
}
```

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

Gets window position relative to screen coordinates.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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.

```
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->GetPosition();
    }

    return nullptr;
}
```

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

	WindowName	Name of the window.
in,out	*Width*	The width.
in,out	*Height*	The height.

Definition at line 542 of file WindowManager.cpp.

References DoesExist(), ERROR_NOTINITIALIZED, FOUNDATION_ERROR, FOUNDATION_OKAY, GetInstance(), GetWindowByName(), IsInitialized(), and PrintErrorMessage().

```
{
    if (GetInstance()->IsInitialized())
    {
        if (DoesExist(WindowName))
        {
            if (GetWindowByName(WindowName)->GetResolution(Width, Height))
            {
                return FOUNDATION_OKAY;
            }
            return FOUNDATION_ERROR;
        }
        return FOUNDATION_ERROR;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}
```

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

Gets window resolution.

Author

Ziyad

Date

29/11/2014

Parameters

	WindowIndex	Zero-based index of the window.
in,out	*Width*	The width.
in,out	*Height*	The height.

Definition at line 574 of file WindowManager.cpp.

References DoesExist(), ERROR_NOTINITIALIZED, FOUNDATION_ERROR, FOUNDATION_OKAY, FWindow::GetResolution(), GetWindowByIndex(), and PrintErrorMessage().

```
{
    if (DoesExist(WindowIndex))
    {
        GetWindowByIndex(WindowIndex)->GetResolution(Width, Height);
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}
```

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

Gets window resolution as an array.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetResolution();
    }

    return nullptr;
}
```

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

Gets window resolution.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetResolution();
    }

    return nullptr;
}
```

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

Gets whether the window should close.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetShouldClose();
    }

    return FOUNDATION_ERROR;
}
```

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

Gets whether the window should close.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetShouldClose();
    }

    return FOUNDATION_ERROR;
}
```

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().

```
{
    GetInstance()->Initialized = GL_FALSE;
#if defined(CURRENT_OS_LINUX)
    return Linux_Initialize();
#endif

#if defined(CURRENT_OS_WINDOWS)
    return Windows_Initialize();
#endif
}
```

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().

```
{
    return GetInstance()->Initialized;
}
```

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

Maximize window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
ShouldBe-Maximized	Whether the window should be maximized.

Definition at line 1338 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByName().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeMaximized);
    }

    return FOUNDATION_ERROR;
}
```

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

Maximize window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
ShouldBe-Maximized	Whether the window should be maximized.

Definition at line 1360 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByIndex().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMaximized);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
ShouldBe-Minimized	whether the window should be minimized.

Definition at line 1248 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByName().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeMinimized);
    }

    return FOUNDATION_ERROR;
}
```

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

Minimize window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
ShouldBe-Minimized	Whether the window should be minimized.

Definition at line 1270 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByIndex().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMinimized);
    }

    return FOUNDATION_ERROR;
}
```

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 PrintError-Message().

Referenced by main().

```
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)
        return GetInstance()->Windows_PollForEvents();
#endif

#if defined (CURRENT_OS_LINUX)
        return GetInstance()->Linux_PollForEvents();
#endif
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.

Definition at line 1561 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::Restore().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->Restore();
    }
    return FOUNDATION_ERROR;
    //implement window focusing
}
```

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

Restore window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.

Definition at line 1582 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::Restore().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->Restore();
    }

    return FOUNDATION_ERROR;
    //implement window focusing
}
```

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

WindowName	Name of the window.
ShouldBe-Fullscreen	whether the window should be in fullscreen mode.

Definition at line 1158 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByName().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeFullscreen);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
ShouldBe-Fullscreen	whether the window should be in fullscreen mode.

Definition at line 1180 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::FullScreen(), and GetWindowByIndex().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeFullscreen);
    }

    return FOUNDATION_ERROR;
}
```

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.

```
{
    GetInstance()->ScreenMousePosition[0] = X;
    GetInstance()->ScreenMousePosition[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
    return Windows_SetMousePositionInScreen(X, Y);
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_SetMousePositionInScreen(X, Y);
#endif
}
```

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

WindowName	Name of the window.
X	The new X coordinate of the mouse position relative to window coordinates.
Y	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
X	The new X coordinate of the mouse position relative to window coordinates.
Y	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().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
OnDestroyed	The on destroyed.

Definition at line 1842 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnDestroyed().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnDestroyed(OnDestroyed);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on destroyed.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnDestroyed	The on destroyed.

Definition at line 1864 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnDestroyed().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnDestroyed(OnDestroyed);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on focus.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
OnFocus	The on focus.

Definition at line 1990 of file WindowManager.cpp.

References FWindow::FocusEvent, FOUNDATION_ERROR, FOUNDATION_OKAY, GetWindowByName(), and IsValidString().

```
{
    if(IsValidString(WindowName))
    {
        GetWindowByName(WindowName)->FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on focus.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnFocus	The on focus.

Definition at line 2013 of file WindowManager.cpp.

References DoesExist(), FWindow::FocusEvent, FOUNDATION_ERROR, FOUNDATION_OKAY, and GetWindowByIndex().

```
{
    if(DoesExist(WindowIndex))
    {
        GetWindowByIndex(WindowIndex)->FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
OnKey	The on key event.

Definition at line 1710 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnKeyEvent().

Referenced by main().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnKeyEvent(OnKey);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on key event.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnKey	The on key event.

Definition at line 1732 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnKeyEvent().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnKeyEvent(OnKey);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on maximized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
OnMaximized	The on maximized.

Definition at line 1886 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMaximized().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMaximized(OnMaximized);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on maximized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnMaximized	The on maximized.

Definition at line 1908 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMaximized().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMaximized(OnMaximized);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on minimized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
OnMinimized	The on minimized.

Definition at line 1930 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMinimized().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMinimized(OnMinimized);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on minimized.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnMinimized	The on minimized.

Definition at line 1952 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMinimized().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMinimized(OnMinimized);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
OnMouseButton	The on mouse button event.

Definition at line 1754 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMouseButton-Event().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseButtonEvent(
  OnMouseButton);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
OnMouseButton	The on mouse button event.

Definition at line 1776 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMouseButton-Event().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->
  SetOnMouseButtonEvent(OnMouseButton);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
OnMouseMove	The on mouse move.

Definition at line 2124 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMouseMove().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseMove(OnMouseMove);
    }

    return FOUNDATION_ERROR;
}
```

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

Sets window on mouse move.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnMouseMove	The on mouse move.

Definition at line 2146 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMouseMove().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMouseMove(OnMouseMove);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
OnMouseWheel	The on mouse wheel event.

Definition at line 1798 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMouseWheel-Event().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseWheelEvent(
  OnMouseWheel);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
OnMouseWheel	The on mouse wheel event.

Definition at line 1820 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMouseWheel-Event().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->
  SetOnMouseWheelEvent(OnMouseWheel);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.75 GLboolean WindowManager::SetWindowOnMoved (const char ∗ *WindowName*, OnMovedEvent *OnMoved*) [static]

Sets window on moved.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
OnMoved	The on moved.

Definition at line 2036 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnMoved().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMoved(OnMoved);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.76 GLboolean WindowManager::SetWindowOnMoved (GLuint *WindowIndex*, OnMovedEvent *OnMoved*) [static]

Sets window on moved.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnMoved	The on moved.

Definition at line 2058 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnMoved().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMoved(OnMoved);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.77 GLboolean WindowManager::SetWindowOnResize (const char ∗ *WindowName,* OnResizeEvent *OnResize*) [static]

Sets window on resize.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.
OnResize	The on resize.

Definition at line 2080 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetOnResize().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnResize(OnResize);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.78 GLboolean WindowManager::SetWindowOnResize (GLuint *WindowIndex,* OnResizeEvent *OnResize*) [static]

Sets window on resize.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
OnResize	The on resize.

Definition at line 2102 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetOnResize().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnResize(OnResize);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
X	The new X coordinate of the window relative to screen coordinates.
Y	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().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetPosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
X	The new X coordinate of the window relative to screen coordinates.
Y	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.

```
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->SetPosition(X, Y);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
Width	The width.
Height	The height.

Definition at line 647 of file WindowManager.cpp.

References DoesExist(), GetWindowByName(), and FWindow::SetResolution().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetResolution(Width, Height);
    }

    return GL_FALSE;
}
```

3.2.3.82 GLboolean WindowManager::SetWindowResolution (GLuint *WindowIndex,* GLuint *Width,* GLuint *Height*) [static]

Sets window resolution.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
Width	The width.
Height	The height.

Definition at line 670 of file WindowManager.cpp.

References DoesExist(), GetWindowByIndex(), and FWindow::SetResolution().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetResolution(Width, Height);
    }

    return GL_FALSE;
}
```

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().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetStyle(WindowStyle);
    }

    return FOUNDATION_ERROR;
}
```

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().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetStyle(WindowStyle);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
a_SyncSetting	The synchronize setting.

Definition at line 1605 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SetSwapInterval().

```
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetSwapInterval(a_SyncSetting);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.86 GLboolean WindowManager::SetWindowSwapInterval (GLuint *WindowIndex,* GLint *a_SyncSetting*) [static]

Sets window swap interval.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
a_SyncSetting	The synchronize setting.

Definition at line 1627 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SetSwapInterval().

```
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetSwapInterval(a_SyncSetting)
;
    }

    return FOUNDATION_ERROR;
}
```

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

WindowName	Name of the window.
NewTitle	The new title bar text.

Definition at line 1428 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), IsValidString(), and FWindow::SetTitle-Bar().

```
{
    if(DoesExist(WindowName) && IsValidString(NewTitle))
    {
        return GetWindowByName(WindowName)->SetTitleBar(NewTitle);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.88 GLboolean WindowManager::SetWindowTitleBar (GLuint *WindowIndex,* const char ∗ *NewTitle*) [static]

Sets window title bar.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.
NewTitle	The new title bar text.

Definition at line 1450 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), IsValidString(), and FWindow::SetTitle-Bar().

```
{
    if(DoesExist(WindowIndex) && IsValidString(NewTitle))
    {
        return GetWindowByIndex(WindowIndex)->SetTitleBar(NewTitle);
    }

    return FOUNDATION_ERROR;
}
```

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().

```
{
#if defined(CURRENT_OS_WINDOWS)
    for each(auto CurrentWindow in GetInstance()->Windows)
    {
        delete CurrentWindow;
    }

    GetInstance()->Windows.clear();

#endif

#if defined(CURRENT_OS_LINUX)
    for (auto CurrentWindow : GetInstance()->Windows)
    {
        delete CurrentWindow;
    }
    
    GetInstance()->Windows.clear();
    
    XCloseDisplay(GetInstance()->m_Display);
#endif

    delete Instance;
}
```

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

WindowName	Name of the window.
Key	The key.

Returns

The state of the key.

Definition at line 978 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::GetKeyState(), and GetWindowByName().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetKeyState(Key);
    }

    return FOUNDATION_ERROR;
}
```

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

WindowIndex	Zero-based index of the window.
Key	The key.

Returns

The state of the key.

Definition at line 1002 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, FWindow::GetKeyState(), and GetWindowByIndex().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetKeyState(Key);
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.92 GLboolean WindowManager::WindowSwapBuffers (const char ∗ *WindowName*) [static]

Swap DrawBuffers for that window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowName	Name of the window.

Definition at line 1069 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByName(), and FWindow::SwapDrawBuffers().

```
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SwapDrawBuffers();
    }

    return FOUNDATION_ERROR;
}
```

3.2.3.93 GLboolean WindowManager::WindowSwapBuffers (GLuint *WindowIndex*) [static]

Swap DrawBuffers for that window.

Author

Ziyad

Date

29/11/2014

Parameters

WindowIndex	Zero-based index of the window.

Definition at line 1090 of file WindowManager.cpp.

References DoesExist(), FOUNDATION_ERROR, GetWindowByIndex(), and FWindow::SwapDrawBuffers().

```
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SwapDrawBuffers();
    }

    return FOUNDATION_ERROR;
}
```

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(), GetWindowIsFullScreen(), 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

KeySym	The key symbol.
KeyState	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");
```

```
    }
}

/**********************************************************************************************/
int main()
{
    WindowManager::Initialize();
    WindowManager::AddWindow(new FWindow("Example"));//->AddWindow(new
 FWindow("Example2"));
    WindowManager::SetWindowOnKeyEvent("Example", &
OnWindowKeyPressed);
    WindowManager::SetWindowStyle("Example",
WINDOWSTYLE_BARE);
    WindowManager::EnableWindowDecorator("Example",
DECORATOR_MINIMIZEBUTTON);
    while (!WindowManager::GetWindowShouldClose("Example"))
    {
        WindowManager::PollForEvents();

        for (GLuint i = 0; i < WindowManager::GetNumWindows(); i++)
        {
            WindowManager::GetWindowByIndex(i)->
MakeCurrentContext();
            glClearColor(0.25f, 0.25f, 0.25f, 0.25f);
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            WindowManager::GetWindowByIndex(i)->
SwapDrawBuffers();
        }
    }

    WindowManager::ShutDown();
    return 0;
}
```

4.3 Window.cpp File Reference

```
#include <limits.h>
#include "Window.h"
#include "WindowManager.h"
```

4.4 Window.cpp

```
/**********************************************************************************************/
#include <limits.h>
#include "Window.h"
#include "WindowManager.h"

#if defined(CURRENT_OS_LINUX)
#include <cstring>
#endif

/**********************************************************************************************/
FWindow::FWindow(const char*  WindowName,
    GLuint Width /* = 1280 */,
    GLuint Height /* = 720 */,
    GLuint ColourBits /* = 32 */,
    GLuint DepthBits /* = 8 */,
    GLuint StencilBits /* = 8 */) :
    Name(WindowName),
    ColourBits(ColourBits),
    DepthBits(DepthBits),
    StencilBits(StencilBits)
{
    Resolution[0] = Width;
    Resolution[1] = Height;
    Position[0] = 0;
    Position[1] = 0;
    ShouldClose = GL_FALSE;
    EXTSwapControlSupported = GL_FALSE;
    SGISwapControlSupported = GL_FALSE;
    MESASwapControlSupported = GL_FALSE;

    if(!IsValidString(WindowName))
    {
        PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
        exit(0);
    }

    InitializeEvents();

    CurrentState = WINDOWSTATE_NORMAL;
    ContextCreated = GL_FALSE;
    IsCurrentContext = GL_FALSE;
}

/*********************************************************************************************/
FWindow::~FWindow()
{
    Shutdown();
}

/*********************************************************************************************/
GLboolean FWindow::Shutdown()
{
    if(ContextCreated)
    {

#if defined (CURRENT_OS_WINDOWS)
        Windows_Shutdown();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Shutdown();
#endif
        ContextCreated = GL_FALSE;
        return FOUNDATION_OKAY;
    }

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean FWindow::Initialize()
{
#if defined(CURRENT_OS_WINDOWS)
    return Windows_Initialize();
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_Initialize();
#endif
}

/*********************************************************************************************/
GLboolean FWindow::GetShouldClose()
{
    return ShouldClose;
}

/*********************************************************************************************/
void FWindow::InitializeEvents()
{
    KeyEvent = nullptr;
    MouseButtonEvent = nullptr;
    MouseWheelEvent = nullptr;
    DestroyedEvent = nullptr;
    MaximizedEvent = nullptr;
    MinimizedEvent = nullptr;
//  RestoredEvent = nullptr;
    MovedEvent = nullptr;
    MouseMoveEvent = nullptr;
}

/*********************************************************************************************/
GLboolean FWindow::GetKeyState(GLuint Key)
{
    return Keys[Key];
}

/*********************************************************************************************/
GLboolean FWindow::InitializeGL()
{
#if defined(CURRENT_OS_WINDOWS)
    return Windows_InitializeGL();
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_InitializeGL();
#endif
}

/*********************************************************************************************/
```

```
GLboolean FWindow::SwapDrawBuffers()
{
    if(ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        SwapBuffers(DeviceContextHandle);
#endif

#if defined(CURRENT_OS_LINUX)
        glXSwapBuffers(WindowManager::GetDisplay(), WindowHandle);
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::SetSwapInterval(GLint SwapSetting)
{
    if(ContextCreated)
    {
    CurrentSwapInterval = SwapSetting;
#if defined(CURRENT_OS_WINDOWS)
    Windows_VerticalSync(SwapSetting);
#endif

#if defined(CURRENT_OS_LINUX)
    Linux_VerticalSync(SwapSetting);
#endif

    return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLuint FWindow::GetCurrentState()
{
    return CurrentState;
}

/**********************************************************************************************/
GLboolean FWindow::SetCurrentState(GLuint NewState)
{
    if(ContextCreated)
    {

    Restore();

    switch(NewState)
    {
        case WINDOWSTATE_MAXIMIZED:
            {
                Maximize(GL_TRUE);
                break;
            }

        case WINDOWSTATE_MINIMIZED:
            {
                Minimize(GL_TRUE);
                break;
            }

            case WINDOWSTATE_FULLSCREEN:
            {
                FullScreen(GL_FALSE);
                break;
            }

            default:
            {
                break;
            }
    }
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::GetIsFullScreen()
```

```
{
    if(ContextCreated)
    {
        return (CurrentState == WINDOWSTATE_FULLSCREEN);
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::FullScreen(GLboolean ShouldBeFullscreen)
{
    if(ContextCreated)
    {
        if (ShouldBeFullscreen)
        {
            CurrentState = WINDOWSTATE_FULLSCREEN;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_LINUX)
        Linux_FullScreen(ShouldBeFullscreen);
#endif

#if defined(CURRENT_OS_WINDOWS)
        Windows_FullScreen();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_OKAY;
}

/**********************************************************************************************/
GLboolean FWindow::GetIsMinimized()
{
    return (CurrentState == WINDOWSTATE_MINIMIZED);
}

/**********************************************************************************************/
GLboolean FWindow::Minimize(GLboolean NewState)
{
    if(ContextCreated)
    {
        if(NewState)
        {
            CurrentState = WINDOWSTATE_MINIMIZED;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_WINDOWS)
        Windows_Minimize();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Minimize(NewState);
#endif

        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::GetIsMaximized()
{
    return (CurrentState == WINDOWSTATE_MAXIMIZED) ;
}

/**********************************************************************************************/
GLboolean FWindow::Maximize(GLboolean NewState)
{
    if(ContextCreated)
    {
        if(NewState)
        {
            CurrentState = WINDOWSTATE_MAXIMIZED;
        }

        else
        {
            CurrentState = WINDOWSTATE_NORMAL;
        }

#if defined(CURRENT_OS_WINDOWS)
         Windows_Maximize();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Maximize(NewState);
#endif
         return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::Restore()
{
    if (ContextCreated)
    {
        switch (CurrentState)
        {
        case WINDOWSTATE_MAXIMIZED:
        {
            Maximize(GL_FALSE);
            break;
        }

        case WINDOWSTATE_FULLSCREEN:
        {
            FullScreen(GL_FALSE);
            break;
        }
        }

        CurrentState = WINDOWSTATE_NORMAL;
#if defined(CURRENT_OS_WINDOWS)
        Windows_Restore();
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_Restore();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::GetResolution(GLuint& Width, GLuint& Height)
{
    if (ContextCreated)
    {
        Width = Resolution[0];
        Height = Resolution[1];
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLuint* FWindow::GetResolution()
{
    return Resolution;
}

/**********************************************************************************************/
GLboolean FWindow::SetResolution(GLuint Width, GLuint Height)
{
    if (ContextCreated)
    {
        if(Width > 0 && Height > 0)
        {
            Resolution[0] = Width;
            Resolution[1] = Height;

#if defined(CURRENT_OS_WINDOWS)
            Windows_SetResolution(Resolution[0], Resolution[1]);
#endif

#if defined(CURRENT_OS_LINUX)
            Linux_SetResolution(Width, Height);
#endif

            glViewport(0, 0, Resolution[0], Resolution[1]);

            return FOUNDATION_OKAY;
        }

        else
        {
            PrintErrorMessage(ERROR_INVALIDRESOLUTION);
            return FOUNDATION_ERROR;
        }
    }
        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::GetMousePosition(GLuint& X, GLuint& Y)
{
    if (ContextCreated)
    {
        X = MousePosition[0];
        Y = MousePosition[1];
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLuint* FWindow::GetMousePosition()
{
    if (ContextCreated)
    {
        return MousePosition;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}

/**********************************************************************************************/
GLboolean FWindow::SetMousePosition(GLuint X, GLuint Y)
{
    if (ContextCreated)
    {
        MousePosition[0] = X;
        MousePosition[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetMousePosition(X, Y);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetMousePosition(X, Y);
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
GLboolean FWindow::GetPosition(GLuint& X, GLuint& Y)
{
    if (ContextCreated)
    {
        X = Position[0];
        Y = Position[1];

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

```

```
/*********************************************************************************************/
GLuint* FWindow::GetPosition()
{
    return Position;
}

/*********************************************************************************************/
GLboolean FWindow::SetPosition(GLuint X, GLuint Y)
{
    if (ContextCreated)
    {
        Position[0] = X;
        Position[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetPosition(Position[0], Position[1]);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetPosition(X, Y);
#endif
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
const char* FWindow::GetWindowName()
{
    if (ContextCreated)
    {
        return Name;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}

/*********************************************************************************************/
GLboolean FWindow::SetTitleBar(const char* NewTitle)
{
    if (ContextCreated)
    {
        if(NewTitle != nullptr)
        {
#if defined(CURRENT_OS_LINUX)
            Linux_SetTitleBar(NewTitle);
#endif

#if defined(CURRENT_OS_WINDOWS)
            Windows_SetTitleBar(NewTitle);
#endif
            return FOUNDATION_OKAY;
        }

        else
        {
            PrintErrorMessage(ERROR_INVALIDTITLEBAR);
            return FOUNDATION_ERROR;
        }
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

GLboolean FWindow::SetIcon(const char* Icon, GLuint Width, GLuint Height)
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetIcon(Icon, Width, Height);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetIcon(Icon, Width, Height);
#endif

        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}

GLboolean FWindow::SetStyle(GLuint WindowType)
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_SetStyle(WindowType);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_SetStyle(WindowType);
#endif

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}

/**************************************************************************************************/
GLboolean FWindow::MakeCurrentContext()
{
    if(ContextCreated)
    {
        IsCurrentContext = true;
#if defined(CURRENT_OS_WINDOWS)
        wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);
#endif

#if defined(CURRENT_OS_LINUX)
        glXMakeCurrent(WindowManager::GetDisplay(), WindowHandle, Context);
#endif
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/**************************************************************************************************/
GLboolean FWindow::GetIsCurrentContext()
{
    if(ContextCreated)
    {
        return IsCurrentContext;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return GL_FALSE;
}

/**************************************************************************************************/
GLboolean FWindow::GetContextHasBeenCreated()
{
    return ContextCreated;
}

/**************************************************************************************************/
void FWindow::InitGLExtensions()
{
#if defined(CURRENT_OS_WINDOWS)
    Windows_InitGLExtensions();
#endif

#if defined(CURRENT_OS_LINUX)
    Linux_InitGLExtensions();
#endif
}

/**************************************************************************************************/
GLboolean FWindow::PrintOpenGLVersion()
{
    if(ContextCreated)
    {
        printf("%s\n", glGetString(GL_VERSION));
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

const char* FWindow::GetOpenGLVersion()
{
    if(ContextCreated)
    {
        return (const char*)glGetString(GL_VERSION);
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return nullptr;
}
```

```

GLboolean FWindow::PrintOpenGLExtensions()
{
    if(ContextCreated)
    {
        printf("%s \n", (const char*)glGetString(GL_EXTENSIONS));
        return FOUNDATION_OKAY;
    }

        PrintErrorMessage(ERROR_NOCONTEXT);
        return FOUNDATION_ERROR;
}

/*************************************************************************************************/
const char* FWindow::GetOpenGLExtensions()
{
    if(ContextCreated)
    {
        return (const char*)glGetString(GL_EXTENSIONS);
    }

    else
    {
        PrintErrorMessage(ERROR_NOCONTEXT);
        return nullptr;
    }
}

/*************************************************************************************************/
GLboolean FWindow::GetInFocus()
{
    return InFocus;
}

/*************************************************************************************************/
GLboolean FWindow::Focus(GLboolean ShouldBeInFocus)
{
    if (ContextCreated)
    {
        InFocus = ShouldBeInFocus;

#if defined(CURRENT_OS_LINUX)
        Linux_Focus(ShouldBeInFocus);
#endif

#if defined(CURRENT_OS_WINDOWS)
        Windows_Focus();
#endif

        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean FWindow::SetOnKeyEvent(OnKeyEvent OnKey)
{
    if (IsValidKeyEvent(OnKey))
    {
        KeyEvent = OnKey;
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean FWindow::SetOnMouseButtonEvent(
      OnMouseButtonEvent OnMouseButtonEvent)
{
    //we don't really need to check if the context has been created
    if(IsValidKeyEvent(OnMouseButtonEvent))
    {
        MouseButtonEvent = OnMouseButtonEvent;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean FWindow::SetOnMouseWheelEvent(
      OnMouseWheelEvent OnMouseWheel)
{
    if(IsValidMouseWheelEvent(OnMouseWheel))
    {
        MouseWheelEvent = OnMouseWheel;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
GLboolean FWindow::SetOnDestroyed(OnDestroyedEvent OnDestroyed)
{
    if(IsValidDestroyedEvent(OnDestroyed))
    {
        DestroyedEvent = OnDestroyed;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
GLboolean FWindow::SetOnMaximized(OnMaximizedEvent OnMaximized)
{
    if(IsValidDestroyedEvent(OnMaximized))
    {
        MaximizedEvent = OnMaximized;
        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
GLboolean FWindow::SetOnMinimized(OnMinimizedEvent OnMinimized)
{
    if(IsValidDestroyedEvent(OnMinimized))
    {
        MinimizedEvent = OnMinimized;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/*void FWindow::SetOnRestored(OnRestoredEvent OnRestored)
{
    if (IsValid(OnRestored))
    {
        RestoredEvent = OnRestored;
    }
}*/

/***************************************************************************************************/
GLboolean FWindow::SetOnFocus(OnFocusEvent OnFocus)
{
    if(IsValidFocusEvent(OnFocus))
    {
        FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
GLboolean FWindow::SetOnMoved(OnMovedEvent OnMoved)
{
    if(IsValidMovedEvent(OnMoved))
    {
        MovedEvent = OnMoved;
        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
GLboolean FWindow::SetOnResize(OnResizeEvent OnResize)
{
    if(IsValidMovedEvent(OnResize))
    {
        ResizeEvent = OnResize;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

/*****************************************************************************************************/
GLboolean FWindow::SetOnMouseMove(OnMouseMoveEvent OnMouseMove)
{
    if(IsValidMouseMoveEvent(OnMouseMove))
    {
        MouseMoveEvent = OnMouseMove;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_INVALIDEVENT);
    return FOUNDATION_ERROR;
}

GLboolean FWindow::EnableDecorator(GLbitfield Decorator)
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_EnableDecorator(Decorator);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_EnableDecorator(Decorator);
#endif

        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}

GLboolean FWindow::DisableDecorator(GLbitfield Decorator)
{
    if (ContextCreated)
    {
#if defined(CURRENT_OS_WINDOWS)
        Windows_DisableDecorator(Decorator);
#endif

#if defined(CURRENT_OS_LINUX)
        Linux_DisableDecorator(Decorator);
#endif
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOCONTEXT);
    return FOUNDATION_ERROR;
}
```

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

```
#ifndef WINDOW_H
#define WINDOW_H
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <fcntl.h>
#include "WindowAPI_Defs.h"

#if defined(CURRENT_OS_WINDOWS)
#include <io.h>

LRESULT CALLBACK FWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
//this automatically loads the OpenGL library if you are using Visual studio
#pragma comment (lib, "opengl32.lib")
//this makes sure that the entry point of your program is main(). not Winmain
#pragma comment(linker, "/subsystem:windows /ENTRY:mainCRTStartup")
#endif

class WindowManager; // just a forward declaration for the window manager

class FWindow
{
public:
    //window constructor
    FWindow(const char*  WindowName, GLuint Width = 1280, GLuint Height = 720, GLuint ColourBits = 8,
        GLuint DepthBits = 24, GLuint StencilBits = 8);

    //window deconstruction
    ~FWindow();

    //Initializes the window depending on OS
    GLboolean Initialize();

    //shut down respective OpenGL context
    GLboolean Shutdown();

    //return the size/resolution of the window
    GLboolean GetResolution(GLuint& Width, GLuint& Height);
    //return the size/resolution of the window
    GLuint* GetResolution();
    //set the size/Resolution of the window
    GLboolean SetResolution(GLuint Width, GLuint Height);

    //return the position of the mouse cursor relative to the window co-ordinates
    GLboolean GetMousePosition(GLuint& X, GLuint& Y);
    //return the Position of the mouse cursor relative to the window co-ordinates
    GLuint* GetMousePosition();
    //set the position of the mouse cursor relative the the window co-ordinates
    GLboolean SetMousePosition(GLuint X, GLuint Y);

    //return the Position of the window relative to the screen co-ordinates
    GLboolean GetPosition(GLuint& X, GLuint& Y);
    //return the Position of the window relative to the screen co-ordinates
    GLuint* GetPosition();
    //Set the Position of the window relative to the screen co-ordinates
    GLboolean SetPosition(GLuint X, GLuint Y);

    //return the current state of the window
    GLuint GetCurrentState();
    //set the current state of the window
    GLboolean SetCurrentState(GLuint NewState);

    //get the state of a key(Down/Up) by index
    GLboolean GetKeyState(GLuint Key);

    //whether or not the window should be closing
    GLboolean GetShouldClose();

    //make the window swap draw buffers
    GLboolean SwapDrawBuffers();

    //toggle full screen mode depending on NewState. (true = Full screen, false = normal)
    GLboolean FullScreen(GLboolean NewState);
    //return if the window is in full screen mode
    GLboolean GetIsFullScreen();

    //toggle minimization depending on NewState. (true = minimized, false = normal)
    GLboolean Minimize(GLboolean NewState);
    //return if the window is Minimized
    GLboolean GetIsMinimized();

    // set and get for maximizing a window
    GLboolean Maximize(GLboolean NewState);
    GLboolean GetIsMaximized();

    //restore the window to its natural state
    GLboolean Restore();

    //creates on OpenGL Context
    GLboolean InitializeGL();

    //get and set for window name
    const char* GetWindowName();
    GLboolean SetTitleBar(const char* NewText);

    //set the style for the window
    GLboolean SetStyle(GLuint WindowType);

    //set the window icon
    GLboolean SetIcon(const char* Icon, GLuint Width, GLuint Height);

    //make the window the current OpenGL context to be drawn
    GLboolean MakeCurrentContext();

    //returns Whether the current window is the current OpenGL context to be drawn
    GLboolean GetIsCurrentContext();

    //returns whether the OpenGL context for this window has been created
    GLboolean GetContextHasBeenCreated();

    //whether the window is in focus
    GLboolean GetInFocus();
    GLboolean Focus(GLboolean NewState);

    //enable vertical sync if supported
    //a swap setting of -1 turns on adaptive V-sync on supported systems
    GLboolean SetSwapInterval(GLint SwapSetting);

    //set the on key event callback for this window
    GLboolean SetOnKeyEvent(OnKeyEvent OnKey);
    //set the on mouse button event callback for this window
    GLboolean SetOnMouseButtonEvent(OnMouseButtonEvent OnMouseButton
);
    //set the on mouse wheel event callback for this window
    GLboolean SetOnMouseWheelEvent(OnMouseWheelEvent OnMouseWheel);
    //set the window on destroyed event callback for this window
    GLboolean SetOnDestroyed(OnDestroyedEvent OnDestroyed);
    //set the window on maximizes event callback for this window
    GLboolean SetOnMaximized(OnMaximizedEvent OnMaximized);
    //set the window on minimized event callback for this window
    GLboolean SetOnMinimized(OnMinimizedEvent OnMinimized);
    //set the window on restored event callback for this window
    //void SetOnRestored(OnRestoredEvent OnRestored);
    //set the window on focus event callback for this window
    GLboolean SetOnFocus(OnFocusEvent OnFocus);
    //set the window on moved event callback for this window
    GLboolean SetOnMoved(OnMovedEvent OnMoved);
    //set the window on resize event callback for this window
    GLboolean SetOnResize(OnResizeEvent OnResize);
    //set the window on Mouse move callback event for this window
    GLboolean SetOnMouseMove(OnMouseMoveEvent OnMouseMove);

    //print the current OpenGL version
    GLboolean PrintOpenGLVersion();
    //return the current OpenGL version as a string
    const char* GetOpenGLVersion();
    //print all supported extensions
    GLboolean PrintOpenGLExtensions();
    //return all the supported extensions
    const char* GetOpenGLExtensions();

    //enable window decorator
    GLboolean EnableDecorator(GLbitfield Decorator);
    //disable window decorator
    GLboolean DisableDecorator(GLbitfield Decorator);

    friend class WindowManager; // lets window use private variables of WindowManager

private:

    const char*  Name;
    GLuint ID;
    GLint ColourBits;
    GLint DepthBits;
    GLint StencilBits;
    GLboolean Keys[KEY_LAST];
    GLboolean MouseButton[MOUSE_LAST];
    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;
//set all the Events to null
    void InitializeEvents();
    //Initializes OpenGL extensions
    void InitGLExtensions();

    OnKeyEvent KeyEvent;
    OnMouseButtonEvent MouseButtonEvent;
    OnMouseWheelEvent MouseWheelEvent;
    OnDestroyedEvent DestroyedEvent;
    OnMaximizedEvent MaximizedEvent;
    OnMinimizedEvent MinimizedEvent;
    //OnRestoredEvent RestoredEvent; /**< this is the callback to be used when the window has been restored
   in a non-programmatic fashion*/
    OnFocusEvent FocusEvent;
    OnMovedEvent MovedEvent;
    OnResizeEvent ResizeEvent;
    OnMouseMoveEvent MouseMoveEvent;
    GLboolean EXTSwapControlSupported;
    GLboolean SGISwapControlSupported;
    GLboolean MESASwapControlSupported;
    //this section is for the windows side of the Window API
#if defined(CURRENT_OS_WINDOWS)

private:

    //tells windows to create a generic window. need to implement window styles sometime later
    GLboolean Windows_Initialize(UINT a_Style = CS_OWNDC | CS_HREDRAW | CS_VREDRAW,
        GLint a_ClearScreenExtra = 0, GLint FWindowExtra = 0,
        HINSTANCE a_Instance = GetModuleHandle(0),
        HICON a_Icon = LoadIcon(0, IDI_APPLICATION),
        HCURSOR a_Cursor = LoadCursor(0, IDC_ARROW),
        HBRUSH a_Brush = (HBRUSH)BLACK_BRUSH);
    //uses the Win32 system to set the resolution/size of the window
    void Windows_SetResolution(GLuint Width, GLuint Height);
    //uses the win32 system to set the position of the window relative to the screen
    void Windows_SetPosition(GLuint X, GLuint Y);
    //uses the win32 system to set the position of the mouse cursor relative to the window
    void Windows_SetMousePosition(GLuint X, GLuint& Y);
    /*uses the win32 system to have the window completely fill the screen and be
    drawn above the toolbar. changing the screen resolution to match has been disabled
    due to event handling issues*/
    void Windows_FullScreen();
    //uses the win32 system to minimize/hide the window. minimized windows don't receive events
    void Windows_Minimize();
    //uses the win32 system to maximize the window.
    void Windows_Maximize();
    //uses the win32 system to restore the window
    void Windows_Restore();
    //uses the win32 system to set the Title Bar text
    void Windows_SetTitleBar(const char* NewTitle);
    //uses the Win32 system to set the window icon
    void Windows_SetIcon(const char* Icon, GLuint Width, GLuint Height);
    //uses the win32 system to put the window into event focus
    void Windows_Focus();
    //initialize OpenGL for this window
    GLboolean Windows_InitializeGL();
    //cleanly shutdown this window(window would still need to be deleted of course)
    void Windows_Shutdown();
    //turns of vertical sync using the EXT extension
    void Windows_VerticalSync(GLint EnableSync);
    //enables given window decoration via Win32
    void Windows_EnableDecorator(GLbitfield Decorator);
    //disables given window decoration via Win32
    void Windows_DisableDecorator(GLbitfield Decorator);

    void Windows_SetStyle(GLuint WindowType);
    //get the handle of the window. to be used internally only
    HWND GetWindowHandle();

    //initialize the pixel format
    void InitializePixelFormat();

    //initialize NEEDED OpenGL extensions for the windows platform
    void Windows_InitGLExtensions();

    HDC DeviceContextHandle;
    HGLRC GLRenderingContextHandle;
    HPALETTE PaletteHandle;
    PIXELFORMATDESCRIPTOR PixelFormatDescriptor;
    WNDCLASS WindowClass;
    HWND WindowHandle;
    HINSTANCE InstanceHandle;
    PFNWGLSWAPINTERVALEXTPROC SwapIntervalEXT;
    PFNWGLSWAPBUFFERSMSCOMLPROC SwapIntervalMSCOM; // what the holy fuck is MSCOM?
    PFNWGLGETEXTENSIONSSTRINGEXTPROC GetExtensionsStringEXT;
#endif

#if defined(CURRENT_OS_LINUX)
    //uses the X11 system to initialize the window
    GLboolean Linux_Initialize();
    //uses the X11 system to set the size/resolution of the window
    void Linux_SetResolution(GLuint Width, GLuint Height);
    //uses the X11 system to set the window position relative to screen co-ordinates
    void Linux_SetPosition(GLuint X, GLuint Y);
    //uses the X11 system to set the mouse position relative to the window co-ordinates
    void Linux_SetMousePosition(GLuint X, GLuint Y);
    //uses the X11 system to toggle full screen mode
    void Linux_FullScreen(GLboolean NewState);
    //uses the X11 system to toggle minimization
    void Linux_Minimize(GLboolean NewState);
    //uses the X11 system to toggle maximization
    void Linux_Maximize(GLboolean NewState);
    //uses the X11 system to restore the window
    void Linux_Restore();
    //uses the X11 system to toggle the window's event focus state
    void Linux_Focus(GLboolean NewState);
    //uses the X11 system to set the title bar of the window
    void Linux_SetTitleBar(const char* NewName);
    //uses the X11 system to set the icon of the window
    void Linux_SetIcon(const char* Icon, GLuint Width, GLuint Height);
    //uses the X11 system to initialize create an OpenGL context for the window
    GLboolean Linux_InitializeGL();
    //uses OpenGL extensions for Linux to toggle Vertical syncing
    void Linux_VerticalSync(GLint EnableSync);
    //shut down the window. closes all connections to the X11 system
    void Linux_Shutdown();
    //enables given window decoration via Win32
    void Linux_EnableDecorator(GLbitfield Decorator);
    //disables given window decoration via Win32
    void Linux_DisableDecorator(GLbitfield Decorator);
    //set the style of the window by enabling/disabling certain decorators
    void Linux_SetStyle(GLuint WindowStyle);

    //initialize the window manager Atomics needed for the X11 extended window manager
    void InitializeAtomics();
    //initialize the NEEDED OpenGL extensions that are supported on Linux
    void Linux_InitGLExtensions();

    //get the Handle To the Window
    Window GetWindowHandle();

    GLXFBConfig GetBestFrameBufferConfig();

    Window WindowHandle;
    GLXContext Context;
    XVisualInfo* VisualInfo;
    GLint* Attributes;
    XSetWindowAttributes SetAttributes;
    GLbitfield Decorators;
    //these are the callbacks for the GLX swap interval extension.
    PFNGLXSWAPINTERVALMESAPROC SwapIntervalMESA;
    PFNGLXSWAPINTERVALEXTPROC SwapIntervalEXT;
    PFNGLXSWAPINTERVALSGIPROC SwapIntervalSGI;
    /*these atomics are needed to change window states via the extended window manager
    I might move them to window manager considering these are essentially constants
    */
    Atom AtomState;     //_NET_WM_STATE
    Atom AtomHidden;    // _NET_WM_STATE_HIDDEN
    Atom AtomFullScreen;  //NET_WM_STATE_FULLSCREEN
    Atom AtomMaxHorz;  // _NET_WM_STATE_MAXIMIZED_HORZ
    Atom AtomMaxVert;  // _NET_WM_STATE_MAXIMIZED_VERT
    Atom AtomClose;     // _NET_WM_CLOSE_WINDOW
    Atom AtomActive;    //_NET_ACTIVE_WINDOW
    Atom AtomDemandsAttention;  //_NET_WM_STATE_DEMANDS_ATTENTION
    Atom AtomFocused;  //_NET_WM_STATE_FOCUSED
    Atom AtomCardinal;  //_NET_WM_CARDINAL
    Atom AtomIcon;  //_NET_WM_ICON
    Atom AtomHints;  //_NET_WM_HINTS

    Atom AtomWindowType;
    Atom AtomWindowTypeDesktop;  //_NET_WM_WINDOW_TYPE_SPLASH
    Atom AtomWindowTypeSplash;
    Atom AtomWindowTypeNormal;
    Atom AtomAllowedActions;
    Atom AtomActionResize;
    Atom AtomActionMinimize;
    Atom AtomActionShade;
    Atom AtomActionMaximizeHorz;
    Atom AtomActionMaximizeVert;
    Atom AtomActionClose;
    Atom AtomDesktopGeometry;
#endif
};

#endif
```

4.7 Window_Linux.cpp File Reference

```
#include "Window.h"
#include "WindowManager.h"
```

4.8 Window_Linux.cpp

```
/*****************************************************************************************/
#include "Window.h"
#include "WindowManager.h"
#if defined(CURRENT_OS_LINUX)
#include <cstring>
/*****************************************************************************************/
GLboolean FWindow::Linux_Initialize()
{
    Attributes = new GLint[12]{GLX_DOUBLEBUFFER, GLX_DEPTH_SIZE, DepthBits, GLX_STENCIL_SIZE,
StencilBits,
       GLX_RED_SIZE, ColourBits, GLX_GREEN_SIZE, ColourBits, GLX_BLUE_SIZE,
ColourBits, None};

    Decorators = 1;
    CurrentWindowStyle |= LINUX_DECORATOR_CLOSE |
LINUX_DECORATOR_MAXIMIZE | LINUX_DECORATOR_MINIMIZE |
LINUX_DECORATOR_MOVE;

    if (!WindowManager::GetDisplay())
    {
        PrintErrorMessage(ERROR_LINUX_CANNOTCONNECTXSERVER
);
        exit(0);
    }

    VisualInfo = glXGetVisualFromFBConfig(WindowManager::GetDisplay(), GetBestFrameBufferConfig());

    //VisualInfo = glXChooseVisual(WindowManager::GetDisplay(), 0, Attributes);

    if (!VisualInfo)
    {
        PrintErrorMessage(ERROR_LINUX_INVALIDVISUALINFO);
        exit(0);
    }

    SetAttributes.colormap = XCreateColormap(WindowManager::GetDisplay(),
        DefaultRootWindow(WindowManager::GetDisplay()),
        VisualInfo->visual, AllocNone);

    SetAttributes.event_mask = ExposureMask | KeyPressMask
        | KeyReleaseMask | MotionNotify | ButtonPressMask | ButtonReleaseMask
        | FocusIn | FocusOut | Button1MotionMask | Button2MotionMask | Button3MotionMask |
        Button4MotionMask | Button5MotionMask | PointerMotionMask | FocusChangeMask
        | VisibilityChangeMask | PropertyChangeMask | SubstructureNotifyMask;

    WindowHandle = XCreateWindow(WindowManager::GetInstance()->m_Display,
        XDefaultRootWindow(WindowManager::GetInstance()->m_Display), 0, 0,
        Resolution[0], Resolution[1],
        0, VisualInfo->depth, InputOutput,
        VisualInfo->visual, CWColormap | CWEventMask,
        &SetAttributes);

    if(!WindowHandle)
    {
        PrintErrorMessage(ERROR_LINUX_CANNOTCREATEWINDOW);
        exit(0);
    }

    XMapWindow(WindowManager::GetDisplay(), WindowHandle);
    XStoreName(WindowManager::GetDisplay(), WindowHandle,
        Name);

    InitializeAtomics();

    XSetWMProtocols(WindowManager::GetDisplay(), WindowHandle, &AtomClose, GL_TRUE);

    return Linux_InitializeGL();
}

/**********************************************************************************************/
void FWindow::Linux_Shutdown()
{
    if(CurrentState == WINDOWSTATE_FULLSCREEN)
    {
        Restore();
    }

    glXDestroyContext(WindowManager::GetDisplay(), Context);
    XUnmapWindow(WindowManager::GetDisplay(), WindowHandle);
    XDestroyWindow(WindowManager::GetDisplay(), WindowHandle);
    WindowHandle = 0;
    Context = 0;
}

/**********************************************************************************************/
void FWindow::Linux_SetResolution(GLuint Width, GLuint Height)
{
    Resolution[0] = Width;
    Resolution[1] = Height;
    XResizeWindow(WindowManager::GetDisplay(),
        WindowHandle, Resolution[0], Resolution[1]);
}

/**********************************************************************************************/
void FWindow::Linux_SetPosition(GLuint X, GLuint Y)
{
    XWindowChanges l_WindowChanges;

    l_WindowChanges.x = X;
    l_WindowChanges.y = Y;

    XConfigureWindow(
            WindowManager::GetDisplay(),
            WindowHandle, CWX | CWY, &l_WindowChanges);
}

/**********************************************************************************************/
void FWindow::Linux_SetMousePosition(GLuint X, GLuint Y)
{
    XWarpPointer(
            WindowManager::GetInstance()->m_Display,
            WindowHandle, WindowHandle,
            Position[0], Position[1],
            Resolution[0], Resolution[1],
            X, Y);
}

/**********************************************************************************************/
void FWindow::Linux_FullScreen(GLboolean ShouldBeFullscreen)
{
    XEvent l_Event;
    memset(&l_Event, 0, sizeof(l_Event));

    l_Event.xany.type = ClientMessage;
    l_Event.xclient.message_type = AtomState;
    l_Event.xclient.format = 32;
    l_Event.xclient.window = WindowHandle;
    l_Event.xclient.data.l[0] = ShouldBeFullscreen;
    l_Event.xclient.data.l[1] = AtomFullScreen;

    XSendEvent(WindowManager::GetDisplay(),
            XDefaultRootWindow(WindowManager::GetDisplay()),
            0, SubstructureNotifyMask, &l_Event);
}

/**********************************************************************************************/
void FWindow::Linux_Minimize(GLboolean ShouldBeMinimized)
{
    if(ShouldBeMinimized)
    {
        XIconifyWindow(WindowManager::GetDisplay(),
                WindowHandle, 0);
    }

    else
    {
        XMapWindow(WindowManager::GetDisplay(), WindowHandle);
    }
}

/**********************************************************************************************/
```

```
void FWindow::Linux_Maximize(GLboolean ShouldBeMaximized)
{
    XEvent l_Event;
    memset(&l_Event, 0, sizeof(l_Event));

    l_Event.xany.type = ClientMessage;
    l_Event.xclient.message_type = AtomState;
    l_Event.xclient.format = 32;
    l_Event.xclient.window = WindowHandle;
    l_Event.xclient.data.l[0] = ShouldBeMaximized;
    l_Event.xclient.data.l[1] = AtomMaxVert;
    l_Event.xclient.data.l[2] = AtomMaxHorz;

    XSendEvent(WindowManager::GetDisplay(),
            XDefaultRootWindow(WindowManager::GetDisplay()),
            0, SubstructureNotifyMask, &l_Event);
}

/***********************************************************************************************/
void FWindow::Linux_Restore()
{
    XMapWindow(WindowManager::GetDisplay(), WindowHandle);
}

/***********************************************************************************************/
void FWindow::Linux_SetTitleBar(const char* NewTitle)
{
    XStoreName(WindowManager::GetDisplay(),
            WindowHandle, NewTitle);
}

void FWindow::Linux_SetIcon(const char* Icon, GLuint Width, GLuint Height)
{

    unsigned char* Image32 = (unsigned char*)malloc(Width * Height * 4);

    //XImage* Image = XCreateImage(WindowManager::GetDisplay(), CopyFromParent, DepthBits, ZPixmap, 0,
 Image32,
            //Width, Height, ColourBits, 0);


    FILE* l_File = fopen(Icon, "r");

    if(l_File == 0)
    {
        printf("file not found \n");
    }

    fclose(l_File);

    Pixmap pix = XCreatePixmap(WindowManager::GetDisplay(), XDefaultRootWindow(WindowManager::GetDisplay())
,
            Width, Height, ColourBits);

    XGCValues Values;
    GC gc = XCreateGC(WindowManager::GetDisplay(), pix, 0, &Values);

    //XPutImage(WindowManager::GetDisplay(), pix, gc, Image, Width, Height, 0, 0, Width, Height);


    GLuint width, height, X_Hot, Y_Hot = 0;

    //XReadBitmapFile(Icon, &width, &height, &Data, &X_Hot, &Y_Hot);

    //printf("%i %i %i %i \n", width, height, X_Hot, Y_Hot);
    //printf("%s\n", Data);

    //XChangeProperty(WindowManager::GetDisplay(), WindowHandle,
        //AtomIcon, AtomCardinal, 32, PropModeReplace, (unsigned char*)Image, sizeof(Image));

    system("ls");
    //XEvent Event
    //memset(&Event, 0, sizeof(Event));

    //l_Event.xany.type = ClientMessage;
    //l_Event.xclient.message_type = AtomState;
}

/***********************************************************************************************/
void FWindow::Linux_Focus(GLboolean ShouldBeInFocus)
{
    if(InFocus)
    {
        XMapWindow(WindowManager::GetDisplay(), WindowHandle);
    }

    else
    {
        XUnmapWindow(WindowManager::GetDisplay(), WindowHandle);
    }
}

/***************************************************************************************************/
void FWindow::Linux_VerticalSync(GLint EnableSync)
{
    if(EXTSwapControlSupported)
    {
        SwapIntervalEXT(WindowManager::GetDisplay(), WindowHandle, EnableSync);
    }

    if(MESASwapControlSupported)
    {
        SwapIntervalMESA(EnableSync);
    }

    if(SGISwapControlSupported)
    {
        if(EnableSync < 0)
        {
            EnableSync = 0;
        }
        SwapIntervalSGI(EnableSync);
    }
}

/***************************************************************************************************/
GLboolean FWindow::Linux_InitializeGL()
{
    if(!Context)
    {
        Context = glXCreateContext(
            WindowManager::GetDisplay(),
            VisualInfo, 0, GL_TRUE);

        if(Context)
        {

            glXMakeCurrent(WindowManager::GetDisplay(),
            WindowHandle, Context);

            XWindowAttributes l_Attributes;

            XGetWindowAttributes(WindowManager::GetDisplay(),
            WindowHandle, &l_Attributes);
            Position[0] = l_Attributes.x;
            Position[1] = l_Attributes.y;

            const char* ExtensionsAvailable = 0;

            ExtensionsAvailable = glXQueryExtensionsString(WindowManager::GetDisplay(), 0);

            if(!ExtensionsAvailable)
            {
                PrintWarningMessage(WARNING_NOGLEXTENSIONS);
            }

            else
            {
                InitGLExtensions();
            }
            ContextCreated = GL_TRUE;
            return FOUNDATION_OKAY;
        }
    }

    else
    {
        PrintErrorMessage(ERROR_EXISTINGCONTEXT);
        return FOUNDATION_ERROR;
    }

    return FOUNDATION_ERROR;
}

/***************************************************************************************************/
void FWindow::InitializeAtomics()
{
    AtomState = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE", GL_FALSE);
    AtomFullScreen = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_FULLSCREEN", GL_FALSE);
    AtomMaxHorz = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_MAXIMIZED_HORZ", GL_FALSE);
    AtomMaxVert = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_MAXIMIZED_VERT", GL_FALSE);
    AtomClose = XInternAtom(WindowManager::GetDisplay(), "WM_DELETE_WINDOW", GL_FALSE);
    AtomHidden = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_HIDDEN", GL_FALSE);
    AtomActive = XInternAtom(WindowManager::GetDisplay(), "_NET_ACTIVE_WINDOW", GL_FALSE);
    AtomDemandsAttention = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_DEMANDS_ATTENTION",
GL_FALSE);
    AtomFocused = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_STATE_FOCUSED", GL_FALSE);
    AtomCardinal = XInternAtom(WindowManager::GetDisplay(), "CARDINAL", GL_FALSE);
    AtomIcon = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_ICON", GL_FALSE);
    AtomHints = XInternAtom(WindowManager::GetDisplay(), "_MOTIF_WM_HINTS", GL_TRUE);

    AtomWindowType = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE", GL_FALSE);
    AtomWindowTypeDesktop = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_UTILITY",
GL_FALSE);
    AtomWindowTypeSplash = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_SPLASH", GL_FALSE)
;
    AtomWindowTypeNormal = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_WINDOW_TYPE_NORMAL", GL_FALSE)
;

    AtomAllowedActions = XInternAtom(WindowManager::GetDisplay(), "_NET_WM_ALLOWED_ACTIONS", GL_FALSE);
    AtomActionResize = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_RESIZE", GL_FALSE);
    AtomActionMinimize = XInternAtom(WindowManager::GetDisplay(), "_WM_ACTION_MINIMIZE", GL_FALSE);
    AtomActionShade = XInternAtom(WindowManager::GetDisplay(), "WM_ACTION_SHADE", GL_FALSE);
    AtomActionMaximizeHorz = XInternAtom(WindowManager::GetDisplay(), "_WM_ACTION_MAXIMIZE_HORZ", GL_FALSE)
;
    AtomActionMaximizeVert = XInternAtom(WindowManager::GetDisplay(), "_WM_ACTION_MAXIMIZE_VERT", GL_FALSE)
;
    AtomActionClose = XInternAtom(WindowManager::GetDisplay(), "_WM_ACTION_CLOSE", GL_FALSE);

    AtomDesktopGeometry = XInternAtom(WindowManager::GetDisplay(), "_NET_DESKTOP_GEOMETRY", GL_FALSE);
}

/**********************************************************************************************/
void FWindow::Linux_InitGLExtensions()
{
    SwapIntervalEXT = nullptr;
    SwapIntervalSGI = nullptr;
    SwapIntervalMESA = nullptr;

    SwapIntervalMESA = (PFNGLXSWAPINTERVALMESAPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalMESA")
;
    SwapIntervalEXT = (PFNGLXSWAPINTERVALEXTPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalEXT");
    SwapIntervalSGI = (PFNGLXSWAPINTERVALSGIPROC)glXGetProcAddress((const GLubyte*)"glXSwapIntervalSGI");

    if(SwapIntervalMESA)
    {
        //printf("MESA swap interval supported\n");
        MESASwapControlSupported = GL_TRUE;
    }

    if(SwapIntervalEXT)
    {
        //printf("EXT swap interval supported \n");
        EXTSwapControlSupported = GL_TRUE;
    }

    if(SwapIntervalSGI)
    {
        //printf("SGI swap interval supported \n");
        SGISwapControlSupported = GL_TRUE;
    }
}

/**********************************************************************************************/
Window FWindow::GetWindowHandle()
{
    return WindowHandle;
}

void FWindow::Linux_EnableDecorator(GLbitfield Decorator)
{
        if(Decorator & DECORATOR_CLOSEBUTTON)
        {
            CurrentWindowStyle |= LINUX_DECORATOR_CLOSE;
            Decorators = 1;
        }

        if(Decorator & DECORATOR_MINIMIZEBUTTON)
        {
            CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
            Decorators = 1;
        }

        if(Decorator & DECORATOR_MAXIMIZEBUTTON)
        {
            CurrentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
            Decorators = 1;
        }

        if(Decorator & DECORATOR_ICON)
        {
            //Linux (at least cinammon) doesnt have icons in the window. only in the taskbar icon
        }

        //just need to set it to 1 to enable all decorators that include title bar
        if(Decorator & DECORATOR_TITLEBAR)
        {
            Decorators = 1;
        }

        if(Decorator & DECORATOR_BORDER)
        {
            Decorators = 1;
        }

        if(Decorator & DECORATOR_SIZEABLEBORDER)
        {
            Decorators = 1;
        }

        long hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR, CurrentWindowStyle, Decorators, 0, 0};

        XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
               PropModeReplace,(unsigned char*) hints, 5);

        XMapWindow(WindowManager::GetDisplay(), WindowHandle);
}

void FWindow::Linux_DisableDecorator(GLbitfield Decorator)
{
        if(Decorator & DECORATOR_CLOSEBUTTON)
        {
            //I hate doing this but it is neccessary to keep functionality going.
            GLboolean MinimizeEnabled, MaximizeEnabled;

            if(Decorator & DECORATOR_MAXIMIZEBUTTON)
            {
                MaximizeEnabled = GL_TRUE;
            }

            if(Decorator & DECORATOR_MINIMIZEBUTTON)
            {
                MinimizeEnabled = GL_TRUE;
            }

            CurrentWindowStyle &= ~LINUX_DECORATOR_CLOSE;

            if(MaximizeEnabled)
            {
                CurrentWindowStyle |= LINUX_DECORATOR_MAXIMIZE;
            }

            if(MinimizeEnabled)
            {
                CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
            }

            Decorators = 1;
        }

        if(Decorator & DECORATOR_MINIMIZEBUTTON)
        {
            CurrentWindowStyle &= ~LINUX_DECORATOR_MINIMIZE;
            Decorators = 1;
        }

        if(Decorator & DECORATOR_MAXIMIZEBUTTON)
        {
            GLboolean MinimizeEnabled;

            if(Decorator & DECORATOR_MINIMIZEBUTTON)
            {
                MinimizeEnabled = GL_TRUE;
            }

            CurrentWindowStyle &= ~LINUX_DECORATOR_MAXIMIZE;

            if(MinimizeEnabled)
            {
                CurrentWindowStyle |= LINUX_DECORATOR_MINIMIZE;
            }

            Decorators = 1;
        }

        if(Decorator & DECORATOR_ICON)
        {
            //Linux (at least cinammon) doesnt have icons in the window. only in the taskbar icon
        }

        //just need to set it to 1 to enable all decorators that include title bar
        if(Decorator & DECORATOR_TITLEBAR)
        {
            Decorators = LINUX_DECORATOR_BORDER;
        }

        if(Decorator & DECORATOR_BORDER)
        {
            Decorators = 0;
        }

        if(Decorator & DECORATOR_SIZEABLEBORDER)
        {
            Decorators = 0;
        }

        long hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR, CurrentWindowStyle, Decorators, 0, 0};

        XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32,
               PropModeReplace,(unsigned char*) hints, 5);

        XMapWindow(WindowManager::GetDisplay(), WindowHandle);
}

void FWindow::Linux_SetStyle(GLuint WindowStyle)
{
    switch(WindowStyle)
    {
        case WINDOWSTYLE_DEFAULT:
            {
                Decorators = (1L << 2);
                CurrentWindowStyle = LINUX_DECORATOR_MOVE | LINUX_DECORATOR_CLOSE |
                    LINUX_DECORATOR_MAXIMIZE | LINUX_DECORATOR_MINIMIZE;
                long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR, CurrentWindowStyle, Decorators, 0, 0};

                XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32, PropModeReplace,
                        (unsigned char*)Hints, 5);

                XMapWindow(WindowManager::GetDisplay(), WindowHandle);


                break;
            }

        case WINDOWSTYLE_BARE:
            {
                Decorators = (1L << 2);
                CurrentWindowStyle = (1L << 2);
                long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR, CurrentWindowStyle, Decorators, 0, 0};

                XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32, PropModeReplace,
                        (unsigned char*)Hints, 5);

                XMapWindow(WindowManager::GetDisplay(), WindowHandle);


                break;
            }

        case WINDOWSTYLE_POPUP:
            {
                Decorators = 0;
                CurrentWindowStyle = (1L << 2);
                long Hints[5] = {LINUX_FUNCTION | LINUX_DECORATOR, CurrentWindowStyle, Decorators, 0, 0};

                XChangeProperty(WindowManager::GetDisplay(), WindowHandle, AtomHints, XA_ATOM, 32, PropModeReplace,
                        (unsigned char*)Hints, 5);

                XMapWindow(WindowManager::GetDisplay(), WindowHandle);

                break;
            }

            default:
            {
                PrintErrorMessage(ERROR_INVALIDWINDOWSTYLE);
                break;
            }
    }
}

GLXFBConfig FWindow::GetBestFrameBufferConfig()
{
    const GLint VisualAttributes[]=
    {
        GLX_X_RENDERABLE, GL_TRUE,
        GLX_DRAWABLE_TYPE, GLX_WINDOW_BIT,
        GLX_X_VISUAL_TYPE, GLX_TRUE_COLOR,
        GLX_RED_SIZE, ColourBits,
        GLX_GREEN_SIZE, ColourBits,
        GLX_BLUE_SIZE, ColourBits,
        GLX_ALPHA_SIZE, ColourBits,
        GLX_DEPTH_SIZE, DepthBits,
        GLX_STENCIL_SIZE, StencilBits,
        GLX_DOUBLEBUFFER, GL_TRUE,
        None
    };

    GLint FrameBufferCount;
    GLuint BestBufferConfig, BestNumSamples = 0;
    GLXFBConfig* Configs = glXChooseFBConfig(WindowManager::GetDisplay(), 0, VisualAttributes, &
      FrameBufferCount);

    for(GLuint CurrentConfig = 0; CurrentConfig < FrameBufferCount; CurrentConfig++)
    {
        XVisualInfo* VisInfo = glXGetVisualFromFBConfig(WindowManager::GetDisplay(), Configs[CurrentConfig]
      );

        if(VisInfo)
        {
            //printf("%i %i %i\n", VisInfo->depth, VisInfo->bits_per_rgb, VisInfo->colormap_size);
            GLint Samples, SampleBuffer;
            glXGetFBConfigAttrib(WindowManager::GetDisplay(), Configs[CurrentConfig], GLX_SAMPLE_BUFFERS, &
      SampleBuffer);
            glXGetFBConfigAttrib(WindowManager::GetDisplay(), Configs[CurrentConfig], GLX_SAMPLES, &Samples
      );

            if(SampleBuffer && Samples > -1)
            {
                BestBufferConfig = CurrentConfig;
                BestNumSamples = Samples;
            }
        }

        XFree(VisInfo);
    }

    GLXFBConfig BestConfig = Configs[BestBufferConfig];

    XFree(Configs);

    return BestConfig;
}
#endif
```

4.9 Window_Windows.cpp File Reference

```
#include "Window.h"
#include "WindowManager.h"
```

4.10 Window_Windows.cpp

```
/*********************************************************************************************/
#include "Window.h"
#include "WindowManager.h"

#if defined(CURRENT_OS_WINDOWS)

/*********************************************************************************************/
HWND FWindow::GetWindowHandle()
{
    return WindowHandle;
```

```
}

/*********************************************************************************************/
GLboolean FWindow::Windows_InitializeGL()
{

        DeviceContextHandle = GetDC(WindowHandle);
        InitializePixelFormat();
        GLRenderingContextHandle = wglCreateContext(DeviceContextHandle);
        wglMakeCurrent(DeviceContextHandle, GLRenderingContextHandle);

        ContextCreated = (GLRenderingContextHandle != nullptr);

        if (ContextCreated)
        {
#ifndef CONTEXT_CREATED
#define CONTEXT_CREATED
#endif
            Windows_InitGLExtensions();
            return FOUNDATION_OKAY;
        }

    /*********************************************************************************************/
    PrintErrorMessage(ERROR_INVALIDCONTEXT);
    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
void FWindow::InitializePixelFormat()
{
    PixelFormatDescriptor = {
        sizeof(PIXELFORMATDESCRIPTOR),  /* size */
        1,                              /* version */
        PFD_SUPPORT_OPENGL |
        PFD_DRAW_TO_WINDOW |
        PFD_DOUBLEBUFFER,               /* support double-buffering */
        PFD_TYPE_RGBA,                  /* color type */
        ColourBits, 0,                             /* preferred color depth */
        0, 0,
        0, 0,
        0, 0,
        0,                /* color bits (ignored) */                              /* no alpha buffer */
                          /* alpha bits (ignored) */
        0,                              /* no accumulation buffer */
        0, 0, 0, 0,                     /* accum bits (ignored) */
        DepthBits,                    /* depth buffer */
        StencilBits,                  /* no stencil buffer */
        0,                              /* no auxiliary buffers */
        PFD_MAIN_PLANE,                 /* main layer */
        0,                              /* reserved */
        0, 0, 0,                        /* no layer, visible, damage masks */
    };

    int l_PixelFormat = ChoosePixelFormat(DeviceContextHandle, &PixelFormatDescriptor);

    if (l_PixelFormat)
    {
        SetPixelFormat(DeviceContextHandle, l_PixelFormat, &PixelFormatDescriptor);
        return;
    }
    return;
}

GLboolean FWindow::Windows_Initialize(
    UINT a_Style /* = CS_OWNDC | CS_HREDRAW | CS_DROPSHADOW */,
    int a_ClearScreenExtra /* = 0 */,
    int WindowExtra /* = 0 */,
    HINSTANCE a_Instance /* = GetModuleHandle(0) */,
    HICON a_Icon /* = LoadIcon(0 , IDI_APPLICATION)*/,
    HCURSOR a_Cursor /* = LoadCursor(0 , IDC_ARROW)*/,
    HBRUSH a_Brush /* = (HBRUSH)BLACK_BRUSH */)
{
    InstanceHandle = a_Instance;
    WindowClass.style = a_Style;
    WindowClass.lpfnWndProc = WindowManager::StaticWindowProcedure;
    WindowClass.cbClsExtra = 0;
    WindowClass.cbWndExtra = 0;
    WindowClass.hInstance = InstanceHandle;
    WindowClass.hIcon = a_Icon;
    WindowClass.hCursor = a_Cursor;
    WindowClass.hbrBackground = a_Brush;
    WindowClass.lpszMenuName = Name;
    WindowClass.lpszClassName = Name;
    RegisterClass(&WindowClass);

    CurrentWindowStyle = WS_OVERLAPPEDWINDOW;

    WindowHandle =
        CreateWindow(Name, Name, CurrentWindowStyle, 0,
        0, Resolution[0],
        Resolution[1],
        0, 0, 0, 0);

    if (WindowHandle)
    {
        ShowWindow(WindowHandle, GL_TRUE);
        UpdateWindow(WindowHandle);
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_WINDOWS_CANNOTCREATEWINDOW);
    return FOUNDATION_ERROR;
}

void FWindow::Windows_Shutdown()
{
    if (GLRenderingContextHandle)
    {
        wglMakeCurrent(nullptr, nullptr);
        wglDeleteContext(GLRenderingContextHandle);
    }

    if (PaletteHandle)
    {
        DeleteObject(PaletteHandle);
    }
    ReleaseDC(WindowHandle, DeviceContextHandle);
    UnregisterClass(Name, InstanceHandle);

    FreeModule(InstanceHandle);

    DeviceContextHandle = nullptr;
    WindowHandle = nullptr;
    GLRenderingContextHandle = nullptr;


    //exit here or the loop will just keep running
    //exit(FOUNDATION_OKAY);
}

void FWindow::Windows_FullScreen()
{

        SetWindowLongPtr(WindowHandle, GWL_STYLE,
            WS_SYSMENU | WS_POPUP | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_VISIBLE);

        MoveWindow(WindowHandle, 0, 0, WindowManager::GetScreenResolution()[0],
            WindowManager::GetScreenResolution()[1], GL_TRUE);

        //
/*

        DEVMODE l_ScreenSettings;
        memset(&l_ScreenSettings, 0, sizeof(l_ScreenSettings));
        l_ScreenSettings.dmSize = sizeof(l_ScreenSettings);
        l_ScreenSettings.dmPelsWidth = F_WM::GetScreenResolution()[0];
        l_ScreenSettings.dmPelsHeight = F_WM::GetScreenResolution()[1];
        l_ScreenSettings.dmBitsPerPel = m_ColourBits;
        l_ScreenSettings.dmFields = DM_PELSWIDTH | DM_PELSHEIGHT | DM_BITSPERPEL;

        if (ChangeDisplaySettings(&l_ScreenSettings, CDS_FULLSCREEN) != DISP_CHANGE_SUCCESSFUL)
        {
            printf("could not successfully change to full screen mode \n");
        }*/


        /*RECT l_Rect;
        l_Rect.left = 0;
        l_Rect.top = 0;
        l_Rect.right = m_Resolution[0];
        l_Rect.bottom = m_Resolution[1];

        DEVMODE l_ScreenSettings;

        l_ScreenSettings.dmSize = sizeof(l_ScreenSettings);
        l_ScreenSettings.dmPelsWidth = m_Resolution[0];
        l_ScreenSettings.dmPelsHeight = m_Resolution[1];
        l_ScreenSettings.dmBitsPerPel = 32;
        l_ScreenSettings.dmFields = DM_BITSPERPEL | DM_PELSWIDTH | DM_PELSHEIGHT;

        if (ChangeDisplaySettings(&l_ScreenSettings, CDS_RESET) != DISP_CHANGE_SUCCESSFUL)
        {
            printf("could not successfully change back to regular mode. dear god what have i done? \n");
        }

        SetWindowLongPtr(m_WindowHandle, GWL_STYLE, WS_OVERLAPPEDWINDOW | WS_VISIBLE);
        AdjustWindowRect(&l_Rect, WS_OVERLAPPEDWINDOW, GL_FALSE);
        MoveWindow(m_WindowHandle, m_Position[0], m_Position[1], l_Rect.right, l_Rect.bottom, GL_TRUE);*/
}

void FWindow::Windows_Minimize()
{
    if (CurrentState == WINDOWSTATE_MINIMIZED)
    {
        ShowWindow(WindowHandle, SW_MINIMIZE);
    }

    else
    {
        ShowWindow(WindowHandle, SW_RESTORE);
    }
}

void FWindow::Windows_Maximize()
{
    if (CurrentState == WINDOWSTATE_MAXIMIZED)
    {
        ShowWindow(WindowHandle, SW_MAXIMIZE);
    }

    else
    {
        ShowWindow(WindowHandle, SW_RESTORE);
    }
}

void FWindow::Windows_Restore()
{
    ShowWindow(WindowHandle, SW_RESTORE);
}

void FWindow::Windows_Focus()
{
    if (InFocus)
    {
        SetFocus(WindowHandle);
    }

    else
    {
        SetFocus(nullptr);
    }
}

void FWindow::Windows_SetMousePosition(GLuint X, GLuint& Y)
{
    POINT l_MousePoint;
    l_MousePoint.x = X;
    l_MousePoint.y = Y;
    ScreenToClient(WindowHandle, &l_MousePoint);
    SetCursorPos(l_MousePoint.x, l_MousePoint.y);
}

void FWindow::Windows_SetTitleBar(const char* NewTitle)
{
    SetWindowText(WindowHandle, Name);
}

void FWindow::Windows_SetIcon(const char* Icon, GLuint Width, GLuint Height)
{
    HANDLE icon = LoadImage(InstanceHandle, Icon,
        IMAGE_ICON, Width, Height, LR_LOADFROMFILE);
    SendMessage(WindowHandle, (UINT)WM_SETICON, ICON_BIG, (LPARAM)icon);
}

void FWindow::Windows_SetPosition(GLuint X, GLuint Y)
{
    RECT rect = { X, Y, X, Y };
    AdjustWindowRect(&rect, GWL_STYLE | WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        GL_FALSE);

    SetWindowPos(WindowHandle, HWND_TOP, X, Y,
        Resolution[0], Resolution[1], SWP_SHOWWINDOW | SWP_NOSIZE);
}

void FWindow::Windows_SetResolution(GLuint Width, GLuint Height)
{
    SetWindowPos(WindowHandle, HWND_TOP, Position[0], Position[1],
        Resolution[0], Resolution[1], SWP_SHOWWINDOW | SWP_NOMOVE);
}
```

```

void FWindow::Windows_VerticalSync(GLint EnableSync)
{
    if (EXTSwapControlSupported)
    {
        SwapIntervalEXT(EnableSync);
    }
}

void FWindow::Windows_InitGLExtensions()
{
    SwapIntervalEXT = nullptr;
    GetExtensionsStringEXT = nullptr;

    GetExtensionsStringEXT = (PFNWGLGETEXTENSIONSSTRINGEXTPROC)
        wglGetProcAddress("wglGetExtensionsStringEXT");

    SwapIntervalEXT = (PFNWGLSWAPINTERVALEXTPROC)
        wglGetProcAddress("wglSwapIntervalEXT");

    if (SwapIntervalEXT)
    {
        EXTSwapControlSupported = GL_TRUE;
    }
}

void FWindow::Windows_EnableDecorator(GLbitfield Decorator)
{
    CurrentWindowStyle = WS_VISIBLE | WS_CLIPSIBLINGS;

    if (Decorator & DECORATOR_BORDER)
    {
        CurrentWindowStyle |= WS_BORDER;
    }

    if (Decorator & DECORATOR_TITLEBAR)
    {
        CurrentWindowStyle |= WS_CAPTION;
    }

    if (Decorator & DECORATOR_ICON)
    {
        CurrentWindowStyle |= WS_ICONIC;
    }

    if (Decorator & DECORATOR_CLOSEBUTTON)
    {
        CurrentWindowStyle |= WS_SYSMENU;
    }

    if (Decorator & DECORATOR_MINIMIZEBUTTON)
    {
        CurrentWindowStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
    }

    if (Decorator & DECORATOR_MAXIMIZEBUTTON)
    {
        CurrentWindowStyle |= WS_MAXIMIZEBOX | WS_SYSMENU;
    }

    if (Decorator & DECORATOR_VERTICALSCROLLBAR)
    {
        CurrentWindowStyle |= WS_VSCROLL;
    }

    if (Decorator & DECORATOR_HORIZONTALSCROLLBAR)
    {
        CurrentWindowStyle |= WS_HSCROLL;
    }

    if (Decorator & DECORATOR_SIZEABLEBORDER)
    {
        CurrentWindowStyle |= WS_SIZEBOX;
    }

    SetWindowLongPtr(WindowHandle, GWL_STYLE,
        CurrentWindowStyle);
}

void FWindow::Windows_DisableDecorator(GLbitfield Decorator)
{
    if (Decorator & DECORATOR_BORDER)
    {
        CurrentWindowStyle &= ~WS_BORDER;
    }

    if (Decorator & DECORATOR_TITLEBAR)
    {
        CurrentWindowStyle &= ~WS_MAXIMIZEBOX;
    }

    if (Decorator & DECORATOR_ICON)
    {
        CurrentWindowStyle &= ~WS_ICONIC;
    }

    if (Decorator & DECORATOR_CLOSEBUTTON)
    {
        CurrentWindowStyle &= ~WS_SYSMENU;
    }

    if (Decorator & DECORATOR_MINIMIZEBUTTON)
    {
        CurrentWindowStyle &= ~WS_MINIMIZEBOX;
    }

    if (Decorator & DECORATOR_MAXIMIZEBUTTON)
    {
        CurrentWindowStyle &= ~WS_MAXIMIZEBOX;
    }

    if (Decorator & DECORATOR_VERTICALSCROLLBAR)
    {
        CurrentWindowStyle &= ~WS_VSCROLL;
    }

    if (Decorator & DECORATOR_HORIZONTALSCROLLBAR)
    {
        CurrentWindowStyle &= ~WS_HSCROLL;
    }

    if (Decorator & DECORATOR_SIZEABLEBORDER)
    {
        CurrentWindowStyle &= ~WS_SIZEBOX;
    }

    SetWindowLongPtr(WindowHandle, GWL_STYLE,
        CurrentWindowStyle | WS_VISIBLE);
}

void FWindow::Windows_SetStyle(GLuint WindowType)
{
    switch (WindowType)
    {
    case WINDOWSTYLE_DEFAULT:
    {
        EnableDecorator(DECORATOR_TITLEBAR | DECORATOR_BORDER |
            DECORATOR_CLOSEBUTTON | DECORATOR_MINIMIZEBUTTON | DECORATOR_MAXIMIZEBUTTON);
        break;
    }

    case WINDOWSTYLE_POPUP:
    {
        EnableDecorator(0);
        break;
    }

    case WINDOWSTYLE_BARE:
    {
        EnableDecorator(DECORATOR_TITLEBAR | DECORATOR_BORDER);
        break;
    }

    default:
    {
        PrintErrorMessage(ERROR_INVALIDWINDOWSTYLE);
        break;
    }
    }
}

#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_NOGLEXTENSIONS 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::Get-Resolution(), FWindow::GetWindowName(), FWindow::MakeCurrentContext(), FWindow::Maximize(), Print-ErrorMessage(), 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 PrintError-Message().

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 fist 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_NOGLEXTENSIONS 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::Set-CurrentState().

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::Set-CurrentState().

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 F-Window::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().

```
{
    return (OnMaximized != nullptr);
}
```

4.11.3.2 static GLboolean IsValidFocusEvent (**OnFocusEvent** ***OnFocus***) [inline], [static]

Definition at line 201 of file WindowAPI_Defs.h.

Referenced by FWindow::SetOnFocus().

```
{
    return (OnFocus != nullptr);
}
```

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().

```
{
    return (OnKeyPressed != nullptr);
}
```

4.11.3.4 static GLboolean IsValidMouseMoveEvent (**OnMouseMoveEvent** ***OnMouseMove***) [inline], [static]

Definition at line 211 of file WindowAPI_Defs.h.

Referenced by FWindow::SetOnMouseMove().

```
{
    return (OnMouseMove != nullptr);
}
```

4.11.3.5 static GLboolean IsValidMouseWheelEvent (OnMouseWheelEvent *MouseWheelEvent*) [inline], [static]

Definition at line 191 of file WindowAPI_Defs.h.

Referenced by FWindow::SetOnMouseWheelEvent().

```
{
    return (MouseWheelEvent != nullptr);
}
```

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().

```
{
    return (OnMoved != nullptr);
}
```

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().

```
{
    return (String != nullptr);
}
```

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::GetIsCurrentContext(), 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().

```
{
    switch(ErrorNumber)
    {
        case ERROR_NOCONTEXT:
        {
            printf("Error: An OpenGL context must first be created(initialize the window) \n");
            break;
        }

        case ERROR_INVALIDWINDOWNAME:
        {
            printf("Error: invald window name \n");
            break;
        }

        case ERROR_INVALIDWINDOWINDEX:
        {
            printf("Error: invalid window index \n");
            break;
        }

        case ERROR_INVALIDWINDOWSTATE:
        {
            printf("Error: invalid window state \n");
            break;
        }

        case ERROR_INVALIDRESOLUTION:
        {
            printf("Error: invalid resolution \n");
            break;
        }

        case ERROR_INVALIDCONTEXT:
        {
            printf("Error: Failed to create OpenGL context \n");
            break;
        }

        case ERROR_EXISTINGCONTEXT:
        {
            printf("Error: context already created \n");
            break;
        }

        case ERROR_NOTINITIALIZED:
        {
            printf("Error: Window manager not initialized \n");
            break;
        }

        case ERROR_ALREADYINITIALIZED:
        {
            printf("Error: window has already been initialized \n");
            break;
        }

        case ERROR_INVALIDTITLEBAR:
        {
            printf("Error: invalid title bar name (cannot be null or nullptr) \n");
            break;
        }

        case ERROR_INVALIDEVENT:
        {
            printf("Error: invalid event callback given \n");
            break;
        }

        case ERROR_WINDOWNOTFOUND:
        {
            printf("Error: window was not found \n");
            break;
        }

        case ERROR_INVALIDWINDOWSTYLE:
        {
            printf("Error: invalid window style given \n");
            break;
        }

        case ERROR_INVALIDWINDOW:
        {
            printf("Error: invalid window given \n");
            break;
        }

        case ERROR_FUNCTIONNOTIMPLEMENTED:
        {
            printf("Error: I'm sorry but this function has not been implemented yet :( \n");
            break;
        }

        case ERROR_LINUX_CANNOTCONNECTXSERVER:
        {
            printf("Error: cannot connect to X server \n");
            break;
        }

        case ERROR_LINUX_INVALIDVISUALINFO:
        {
            printf("Error: Invalid visual information given \n");
            break;
        }

        case ERROR_LINUX_CANNOTCREATEWINDOW:
        {
            printf("Error: failed to create window \n");
            break;
        }

        case ERROR_LINUX_FUNCTIONNOTIMPLEMENTED:
        {
            printf("Error: function not implemented on linux platform yet. sorry :( \n");
            break;
        }

        case ERROR_WINDOWS_CANNOTCREATEWINDOW:
        {
            printf("Error: failed to create window \n");
            break;
        }

        case ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED:
        {
            printf("Error: function not implemented on Windows platform yet. sorry ;( \n");
            break;
        }

        default:
        {
            printf("Error: unspecified Error \n");
            break;
        }
    }
}
```

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.

```
{
    switch(WarningNumber)
    {
        case WARNING_NOGLEXTENSIONS:
            {
                printf("Warning: no OpenGL extensions available \n");
                break;
            }

        case WARNING_NOTCURRENTCONTEXT:
            {
                printf("Warning: window not the current OpenGL context being rendered to \n");
                break;
            }

        default:
            {
                printf("Warning: unspecified warning \n");
                break;
            }
    }
}
```

4.12 WindowAPI_Defs.h

```
/*********************************************************************************************/
#ifndef WINDOWAPI_DEFS_H
#define WINDOWAPI_DEFS_H

#include <stdio.h>
#include <stdlib.h>
#include <list>

#if defined(_MSC_VER) || defined(_WIN32) || defined(_WIN64)
#define CURRENT_OS_WINDOWS
#include <windows.h>
#include <gl/GL.h>
#include "../dependencies/wglext.h"
#endif

#if defined(__linux__) || defined(__GNUG__) || defined(__GNUC__) || defined(__clang__)
#define CURRENT_OS_LINUX
#include <GL/glx.h>
#include "../dependencies/glxext.h"
#include <GL/glext.h>
#include <GL/glx.h>
#include <GL/glu.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>
#include <X11/Xatom.h>
#include <string>
#endif

#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_NOGLEXTENSIONS 1

#define LINUX_FUNCTION 1
#define LINUX_DECORATOR 2



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 (*OnRestoredEvent)(); //only really works on windows, Linux doesn't even have an atomic for
 it. might need to remove
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);
//return wether the given string is valid
static inline GLboolean IsValidString(const char* String)
{
    return (String != nullptr);
```

```
}

//return whether the given event is valid
static inline GLboolean IsValidKeyEvent(OnKeyEvent OnKeyPressed)
{
    return (OnKeyPressed != nullptr);
}
//return whether the given event is valid
static inline GLboolean IsValidMouseWheelEvent(
OnMouseWheelEvent MouseWheelEvent)
{
    return (MouseWheelEvent != nullptr);
}
//return whether the given event is valid
static inline GLboolean IsValidDestroyedEvent(
OnMaximizedEvent OnMaximized)
{
    return (OnMaximized != nullptr);
}
//return whether the given event is valid
static inline GLboolean IsValidFocusEvent(OnFocusEvent OnFocus)
{
    return (OnFocus != nullptr);
}
//return whether the given event is valid
static inline GLboolean IsValidMovedEvent(OnMovedEvent OnMoved)
{
    return (OnMoved != nullptr);
}
//return whether the given event is valid
static inline GLboolean IsValidMouseMoveEvent(
OnMouseMoveEvent OnMouseMove)
{
    return (OnMouseMove != nullptr);
}

//print the warning message assosciated with the given warning number
static inline void PrintWarningMessage(GLuint WarningNumber)
{
    switch(WarningNumber)
    {
        case WARNING_NOGLEXTENSIONS:
            {
                printf("Warning: no OpenGL extensions available \n");
                break;
            }

        case WARNING_NOTCURRENTCONTEXT:
            {
                printf("Warning: window not the current OpenGL context being rendered to \n");
                break;
            }

        default:
            {
                printf("Warning: unspecified warning \n");
                break;
            }
    }
}

//print out the error associated with the given error number
static void PrintErrorMessage(GLuint ErrorNumber)
{
    switch(ErrorNumber)
    {
        case ERROR_NOCONTEXT:
        {
            printf("Error: An OpenGL context must first be created(initialize the window) \n");
            break;
        }

        case ERROR_INVALIDWINDOWNAME:
        {
            printf("Error: invald window name \n");
            break;
        }

        case ERROR_INVALIDWINDOWINDEX:
        {
            printf("Error: invalid window index \n");
            break;
        }

        case ERROR_INVALIDWINDOWSTATE:
        {
            printf("Error: invalid window state \n");
```

```
                break;
            }

            case ERROR_INVALIDRESOLUTION:
            {
                printf("Error: invalid resolution \n");
                break;
            }

            case ERROR_INVALIDCONTEXT:
            {
                printf("Error: Failed to create OpenGL context \n");
                break;
            }

            case ERROR_EXISTINGCONTEXT:
            {
                printf("Error: context already created \n");
                break;
            }

            case ERROR_NOTINITIALIZED:
            {
                printf("Error: Window manager not initialized \n");
                break;
            }

            case ERROR_ALREADYINITIALIZED:
            {
                printf("Error: window has already been initialized \n");
                break;
            }

            case ERROR_INVALIDTITLEBAR:
            {
                printf("Error: invalid title bar name (cannot be null or nullptr) \n");
                break;
            }

            case ERROR_INVALIDEVENT:
            {
                printf("Error: invalid event callback given \n");
                break;
            }

            case ERROR_WINDOWNOTFOUND:
            {
                printf("Error: window was not found \n");
                break;
            }

            case ERROR_INVALIDWINDOWSTYLE:
            {
                printf("Error: invalid window style given \n");
                break;
            }

            case ERROR_INVALIDWINDOW:
            {
                printf("Error: invalid window given \n");
                break;
            }

            case ERROR_FUNCTIONNOTIMPLEMENTED:
            {
                printf("Error: I'm sorry but this function has not been implemented yet :( \n");
                break;
            }

            case ERROR_LINUX_CANNOTCONNECTXSERVER:
            {
                printf("Error: cannot connect to X server \n");
                break;
            }

            case ERROR_LINUX_INVALIDVISUALINFO:
            {
                printf("Error: Invalid visual information given \n");
                break;
            }

            case ERROR_LINUX_CANNOTCREATEWINDOW:
            {
                printf("Error: failed to create window \n");
                break;
            }

```

```
        case ERROR_LINUX_FUNCTIONNOTIMPLEMENTED:
        {
            printf("Error: function not implemented on linux platform yet. sorry :( \n");
            break;
        }

        case ERROR_WINDOWS_CANNOTCREATEWINDOW:
        {
            printf("Error: failed to create window \n");
            break;
        }

        case ERROR_WINDOWS_FUNCTIONNOTIMPLEMENTED:
        {
            printf("Error: function not implemented on Windows platform yet. sorry ;( \n");
            break;
        }

        default:
        {
            printf("Error: unspecified Error \n");
            break;
        }
    }
}

#endif
```

4.13 WindowManager.cpp File Reference

```
#include "WindowManager.h"
```

4.14 WindowManager.cpp

```
/*********************************************************************************************/
#include "WindowManager.h"

/*********************************************************************************************/
WindowManager::WindowManager()
{
    //GetInstance()->Initialized = GL_FALSE;
}

/*********************************************************************************************/
GLboolean WindowManager::Initialize()
{
    GetInstance()->Initialized = GL_FALSE;
#if defined(CURRENT_OS_LINUX)
    return Linux_Initialize();
#endif

#if defined(CURRENT_OS_WINDOWS)
    return Windows_Initialize();
#endif
}

GLboolean WindowManager::IsInitialized()
{
    return GetInstance()->Initialized;
}

/*********************************************************************************************/
WindowManager::~WindowManager()
{
    if (!GetInstance()->Windows.empty())
    {
#if defined(CURRENT_OS_WINDOWS)
        for each(auto CurrentWindow in GetInstance()->Windows)
        {
            delete CurrentWindow;
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            delete CurrentWindow;
        }
#endif
        GetInstance()->Windows.clear();
    }
}

/**********************************************************************************************/
FWindow* WindowManager::GetWindowByName(const char* WindowName)
{
    if (DoesExist(WindowName))
    {
#if defined(CURRENT_OS_WINDOWS)
        for each(auto CurrentWindow in GetInstance()->Windows)
        {
            if (CurrentWindow->Name == WindowName)
            {
                return CurrentWindow;
            }
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            if (CurrentWindow->Name == WindowName)
            {
                return CurrentWindow;
            }
        }
#endif
        PrintErrorMessage(ERROR_WINDOWNOTFOUND);
        return nullptr;
    }
    return nullptr;
}

/**********************************************************************************************/
FWindow* WindowManager::GetWindowByIndex(GLuint WindowIndex)
{
    if (DoesExist(WindowIndex))
    {
#if defined(CURRENT_OS_WINDOWS)
        for each (auto CurrentWindow in GetInstance()->Windows)
        {
            if (CurrentWindow->ID == WindowIndex)
            {
                return CurrentWindow;
            }
        }
#endif

#if defined(CURRENT_OS_LINUX)
        for (auto CurrentWindow : GetInstance()->Windows)
        {
            if(CurrentWindow->ID == WindowIndex)
            {
                return CurrentWindow;
            }
        }
#endif
        PrintErrorMessage(ERROR_WINDOWNOTFOUND);
        return nullptr;
    }

    return FOUNDATION_ERROR;
}

/**********************************************************************************************/
WindowManager* WindowManager::AddWindow(
FWindow* NewWindow)
{
    if (GetInstance()->IsInitialized())
    {
        if (NewWindow != nullptr)
        {
            GetInstance()->Windows.push_back(NewWindow);
            NewWindow->ID = GetInstance()->Windows.size() - 1;
            NewWindow->Initialize();
            return GetInstance();
        }
        PrintErrorMessage(ERROR_INVALIDWINDOW);
        return nullptr;
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}

```

```
/*************************************************************************************************/
WindowManager* WindowManager::GetInstance()
{
    if(!WindowManager::Instance)
    {
        WindowManager::Instance = new WindowManager();
        return WindowManager::Instance;
    }

    else
    {
        return WindowManager::Instance;
    }
}

/*************************************************************************************************/
GLboolean WindowManager::DoesExist(const char* WindowName)
{
    if (GetInstance()->IsInitialized())
    {
        if (IsValidString(WindowName))
        {
#if defined(CURRENT_OS_WINDOWS)
            for each(auto iter in GetInstance()->Windows)
            {
                if (iter->Name == WindowName)
                {
                    return GL_TRUE;
                }
            }
#endif

#if defined(CURRENT_OS_LINUX)
            for (auto iter : GetInstance()->Windows)
            {
                if (iter->Name == WindowName)
                {
                    return GL_TRUE;
                }
            }
#endif
        }
        PrintErrorMessage(ERROR_INVALIDWINDOWNAME);
        return GL_FALSE;
    }
    return GL_FALSE;
}

/*************************************************************************************************/
GLboolean WindowManager::DoesExist(GLuint WindowIndex)
{
    if (GetInstance()->IsInitialized())
    {
        if (WindowIndex <= (GetInstance()->Windows.size() - 1))
        {
            return FOUNDATION_OKAY;
        }

        PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
        return FOUNDATION_ERROR;
    }
    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLuint WindowManager::GetNumWindows()
{
    if(GetInstance()->IsInitialized())
    {
        return GetInstance()->Windows.size();
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
void WindowManager::ShutDown()
{
#if defined(CURRENT_OS_WINDOWS)
    for each(auto CurrentWindow in GetInstance()->Windows)
    {
        delete CurrentWindow;
    }

    GetInstance()->Windows.clear();

```

```
#endif

#if defined(CURRENT_OS_LINUX)
    for (auto CurrentWindow : GetInstance()->Windows)
    {
        delete CurrentWindow;
    }

    GetInstance()->Windows.clear();

    XCloseDisplay(GetInstance()->m_Display);
#endif

    delete Instance;
}

/*************************************************************************************************/
GLboolean WindowManager::GetMousePositionInScreen(GLuint& X, GLuint&
 Y)
{
    if (GetInstance()->IsInitialized())
    {
        X = GetInstance()->ScreenMousePosition[0];
        Y = GetInstance()->ScreenMousePosition[1];
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;

}

/*************************************************************************************************/
GLuint* WindowManager::GetMousePositionInScreen()
{
    if (GetInstance()->IsInitialized())
    {
        return GetInstance()->ScreenMousePosition;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}

/*************************************************************************************************/
GLboolean WindowManager::SetMousePositionInScreen(GLuint X, GLuint Y
)
{
    GetInstance()->ScreenMousePosition[0] = X;
    GetInstance()->ScreenMousePosition[1] = Y;
#if defined(CURRENT_OS_WINDOWS)
    return Windows_SetMousePositionInScreen(X, Y);
#endif

#if defined(CURRENT_OS_LINUX)
    return Linux_SetMousePositionInScreen(X, Y);
#endif
}

/*************************************************************************************************/
GLuint* WindowManager::GetScreenResolution()
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)
        RECT l_Screen;
        HWND m_Desktop = GetDesktopWindow();
        GetWindowRect(m_Desktop, &l_Screen);

        GetInstance()->ScreenResolution[0] = l_Screen.right;
        GetInstance()->ScreenResolution[1] = l_Screen.bottom;
        return GetInstance()->ScreenResolution;

#endif

#if defined(CURRENT_OS_LINUX)
        GetInstance()->ScreenResolution[0] = WidthOfScreen(
XDefaultScreenOfDisplay(GetInstance()->m_Display));
        GetInstance()->ScreenResolution[1] = HeightOfScreen(
XDefaultScreenOfDisplay(GetInstance()->m_Display));

        return GetInstance()->ScreenResolution;
#endif
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;

```

```
}

/*****************************************************************************************/
GLboolean WindowManager::PollForEvents()
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)
        return GetInstance()->Windows_PollForEvents();
#endif

#if defined (CURRENT_OS_LINUX)
        return GetInstance()->Linux_PollForEvents();
#endif
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::GetScreenResolution(GLuint& Width, GLuint&
Height)
{
    if (GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)

        RECT l_Screen;
        HWND m_Desktop = GetDesktopWindow();
        GetWindowRect(m_Desktop, &l_Screen);
        Width = l_Screen.right;
        Height = l_Screen.bottom;
#endif

#if defined(CURRENT_OS_LINUX)

        Width = WidthOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));
        Height = HeightOfScreen(XDefaultScreenOfDisplay(GetInstance()->m_Display));


        GetInstance()->ScreenResolution[0] = Width;
        GetInstance()->ScreenResolution[1] = Height;
#endif

        return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::GetWindowResolution(const char* WindowName,
GLuint& Width, GLuint& Height)
{
    if (GetInstance()->IsInitialized())
    {
        if (DoesExist(WindowName))
        {
            if (GetWindowByName(WindowName)->GetResolution(Width, Height))
            {
                return FOUNDATION_OKAY;
            }
            return FOUNDATION_ERROR;
        }
        return FOUNDATION_ERROR;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::GetWindowResolution(GLuint WindowIndex, GLuint&
Width, GLuint& Height)
{
    if (DoesExist(WindowIndex))
    {
        GetWindowByIndex(WindowIndex)->GetResolution(Width, Height);
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
```

```
GLuint* WindowManager::GetWindowResolution(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetResolution();
    }

    return nullptr;
}

/**********************************************************************************************/
GLuint* WindowManager::GetWindowResolution(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetResolution();
    }

    return nullptr;
}

/**********************************************************************************************/
GLboolean WindowManager::SetWindowResolution(const char* WindowName,
GLuint Width, GLuint Height)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetResolution(Width, Height);
    }

    return GL_FALSE;
}

/**********************************************************************************************/
GLboolean WindowManager::SetWindowResolution(GLuint WindowIndex, GLuint
Width, GLuint Height)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetResolution(Width, Height);
    }

    return GL_FALSE;
}

/**********************************************************************************************/
GLboolean WindowManager::GetWindowPosition(const char* WindowName, GLuint&
X, GLuint& Y)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetPosition(X, Y);
    }

    return GL_FALSE;
}

/**********************************************************************************************/
GLboolean WindowManager::GetWindowPosition(GLuint WindowIndex, GLuint& X,
GLuint& Y)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetPosition(X, Y);
    }

    return GL_FALSE;
}

/**********************************************************************************************/
GLuint* WindowManager::GetWindowPosition(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetPosition();
    }

    return nullptr;
}

/**********************************************************************************************/
GLuint* WindowManager::GetWindowPosition(GLuint WindowIndex)
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->GetPosition();
    }

    return nullptr;
}

/**************************************************************************************/
GLboolean WindowManager::SetWindowPosition(const char* WindowName, GLuint X
, GLuint Y)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetPosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

/**************************************************************************************/
GLboolean WindowManager::SetWindowPosition(GLuint WindowIndex, GLuint X,
GLuint Y)
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->SetPosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

/**************************************************************************************/
GLboolean WindowManager::GetMousePositionInWindow(const char*
WindowName, GLuint& X, GLuint& Y)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

/**************************************************************************************/
GLboolean WindowManager::GetMousePositionInWindow(GLuint WindowIndex
, GLuint& X, GLuint& Y)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

/**************************************************************************************/
GLuint* WindowManager::GetMousePositionInWindow(const char*
WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetMousePosition();
    }

    return nullptr;
}

/**************************************************************************************/
GLuint* WindowManager::GetMousePositionInWindow(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetMousePosition();
    }
    PrintErrorMessage(ERROR_INVALIDWINDOWINDEX);
    return nullptr;
}

/**************************************************************************************/
GLboolean WindowManager::SetMousePositionInWindow(const char*
WindowName, GLuint X, GLuint Y)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

```

```
/*************************************************************************************************/
GLboolean WindowManager::SetMousePositionInWindow(GLuint WindowIndex
, GLuint X, GLuint Y)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetMousePosition(X, Y);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::WindowGetKey(const char* WindowName, GLuint Key)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetKeyState(Key);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::WindowGetKey(GLuint WindowIndex, GLuint Key)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetKeyState(Key);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::GetWindowShouldClose(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetShouldClose();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::GetWindowShouldClose(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetShouldClose();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::WindowSwapBuffers(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SwapDrawBuffers();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::WindowSwapBuffers(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SwapDrawBuffers();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::GetWindowIsFullScreen(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsFullScreen();
    }

    return FOUNDATION_ERROR;
```

```
}

/*******************************************************************************************/
GLboolean WindowManager::GetWindowIsFullScreen(GLuint WindowIndex)
{
    if(WindowIndex <= GetInstance()->Windows.size() -1)
    {
        return GetWindowByIndex(WindowIndex)->GetIsFullScreen();
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::SetFullScreen(const char* WindowName, GLboolean ShouldBeFullscreen)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeFullscreen);
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::SetFullScreen(GLuint WindowIndex, GLboolean ShouldBeFullscreen)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeFullscreen);
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::GetWindowIsMinimized(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsMinimized();
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::GetWindowIsMinimized(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetIsMinimized();
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::MinimizeWindow(const char* WindowName, GLboolean ShouldBeMinimized)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeMinimized);
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::MinimizeWindow(GLuint WindowIndex, GLboolean ShouldBeMinimized)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMinimized);
    }

    return FOUNDATION_ERROR;
}

/*******************************************************************************************/
GLboolean WindowManager::GetWindowIsMaximized(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetIsMaximized();
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::GetWindowIsMaximized(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetIsMaximized();
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::MaximizeWindow(const char* WindowName, GLboolean ShouldBeMaximized)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->FullScreen(ShouldBeMaximized);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::MaximizeWindow(GLuint WindowIndex, GLboolean ShouldBeMaximized)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->FullScreen(ShouldBeMaximized);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
const char* WindowManager::GetWindowName(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetWindowName();
    }

    return nullptr;
}

/*****************************************************************************************/
GLuint WindowManager::GetWindowIndex(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->ID;
    }

    return 0;
}

/*****************************************************************************************/
GLboolean WindowManager::SetWindowTitleBar(const char* WindowName, const char* NewTitle)
{
    if(DoesExist(WindowName) && IsValidString(NewTitle))
    {
        return GetWindowByName(WindowName)->SetTitleBar(NewTitle);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************/
GLboolean WindowManager::SetWindowTitleBar(GLuint WindowIndex, const char* NewTitle)
{
    if(DoesExist(WindowIndex) && IsValidString(NewTitle))
    {
        return GetWindowByIndex(WindowIndex)->SetTitleBar(NewTitle);
    }

    return FOUNDATION_ERROR;
}
```

```

/*************************************************************************************************/
GLboolean WindowManager::GetWindowIsInFocus(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->GetInFocus();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::GetWindowIsInFocus(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->GetInFocus();
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::FocusWindow(const char* WindowName, GLboolean
ShouldBeFocused)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->Focus(ShouldBeFocused);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::FocusWindow(GLuint WindowIndex, GLboolean
ShouldBeFocused)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->Focus(ShouldBeFocused);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::RestoreWindow(const char* WindowName)
{
    if(DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->Restore();
    }
    return FOUNDATION_ERROR;
    //implement window focusing
}

/*************************************************************************************************/
GLboolean WindowManager::RestoreWindow(GLuint WindowIndex)
{
    if(DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->Restore();
    }

    return FOUNDATION_ERROR;
    //implement window focusing
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowSwapInterval(const char* WindowName,
GLint a_SyncSetting)
{
    if (DoesExist(WindowName))
    {
         return GetWindowByName(WindowName)->SetSwapInterval(a_SyncSetting);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowSwapInterval(GLuint WindowIndex,
GLint a_SyncSetting)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetSwapInterval(a_SyncSetting);
    }

    return FOUNDATION_ERROR;
}

GLboolean WindowManager::SetWindowStyle(const char* WindowName, GLuint WindowStyle)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetStyle(WindowStyle);
    }

    return FOUNDATION_ERROR;
}

GLboolean WindowManager::SetWindowStyle(GLuint WindowIndex, GLuint WindowStyle)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetStyle(WindowStyle);
    }

    return FOUNDATION_ERROR;
}

GLboolean WindowManager::EnableWindowDecorator(const char* WindowName, GLbitfield Decorators)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->EnableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;

}

GLboolean WindowManager::EnableWindowDecorator(GLuint WindowIndex, GLbitfield Decorators)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->EnableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}

GLboolean WindowManager::DisableWindowDecorator(const char* WindowName, GLbitfield Decorators)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->DisableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}

GLboolean WindowManager::DisableWindowDecorator(GLuint WindowIndex, GLbitfield Decorators)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->DisableDecorator(Decorators);
    }

    return FOUNDATION_ERROR;
}

/****************************************************************************************************/
GLboolean WindowManager::SetWindowOnKeyEvent(const char* WindowName, OnKeyEvent OnKey)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnKeyEvent(OnKey);
    }

    return FOUNDATION_ERROR;
}

```

```
/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnKeyEvent(GLuint WindowIndex,
OnKeyEvent OnKey)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnKeyEvent(OnKey);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseButtonEvent(const char*
WindowName, OnMouseButtonEvent OnMouseButton)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseButtonEvent(
OnMouseButton);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseButtonEvent(GLuint
WindowIndex, OnMouseButtonEvent OnMouseButton)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->
SetOnMouseButtonEvent(OnMouseButton);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseWheelEvent(const char*
WindowName, OnMouseWheelEvent OnMouseWheel)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseWheelEvent(
OnMouseWheel);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseWheelEvent(GLuint
WindowIndex, OnMouseWheelEvent OnMouseWheel)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->
SetOnMouseWheelEvent(OnMouseWheel);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnDestroyed(const char* WindowName,
OnDestroyedEvent OnDestroyed)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnDestroyed(OnDestroyed);
    }

    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
GLboolean WindowManager::SetWindowOnDestroyed(GLuint WindowIndex,
OnDestroyedEvent OnDestroyed)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnDestroyed(OnDestroyed);
    }

    return FOUNDATION_ERROR;
}
```

```

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnMaximized(const char* WindowName,
OnMaximizedEvent OnMaximized)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMaximized(OnMaximized);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnMaximized(GLuint WindowIndex,
OnMaximizedEvent OnMaximized)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMaximized(OnMaximized);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnMinimized(const char* WindowName,
OnMinimizedEvent OnMinimized)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMinimized(OnMinimized);
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnMinimized(GLuint WindowIndex,
OnMinimizedEvent OnMinimized)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMinimized(OnMinimized);
    }

    return FOUNDATION_ERROR;
}

/*void WindowManager::SetWindowOnRestored(const char* WindowName, OnRestoredEvent OnRestored)
{
    if (DoesExist(WindowName))
    {
        GetWindowByName(WindowName)->SetOnRestored(OnRestored);
    }
}

void WindowManager::SetWindowOnRestored(GLuint WindowIndex, OnRestoredEvent OnRestored)
{
    if (DoesExist(WindowIndex))
    {
        GetWindowByIndex(WindowIndex)->SetOnRestored(OnRestored);
    }
}*/

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnFocus(const char* WindowName,
OnFocusEvent OnFocus)
{
    if(IsValidString(WindowName))
    {
        GetWindowByName(WindowName)->FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }

    return FOUNDATION_ERROR;
}

/*****************************************************************************************************/
GLboolean WindowManager::SetWindowOnFocus(GLuint WindowIndex,
OnFocusEvent OnFocus)
{
    if(DoesExist(WindowIndex))
    {
        GetWindowByIndex(WindowIndex)->FocusEvent = OnFocus;
        return FOUNDATION_OKAY;
    }
    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnMoved(const char* WindowName,
OnMovedEvent OnMoved)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMoved(OnMoved);
    }

    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnMoved(GLuint WindowIndex,
OnMovedEvent OnMoved)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMoved(OnMoved);
    }

    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnResize(const char* WindowName,
OnResizeEvent OnResize)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnResize(OnResize);
    }

    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnResize(GLuint WindowIndex,
OnResizeEvent OnResize)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnResize(OnResize);
    }

    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseMove(const char* WindowName,
OnMouseMoveEvent OnMouseMove)
{
    if (DoesExist(WindowName))
    {
        return GetWindowByName(WindowName)->SetOnMouseMove(OnMouseMove);
    }

    return FOUNDATION_ERROR;
}

/*********************************************************************************************/
GLboolean WindowManager::SetWindowOnMouseMove(GLuint WindowIndex,
OnMouseMoveEvent OnMouseMove)
{
    if (DoesExist(WindowIndex))
    {
        return GetWindowByIndex(WindowIndex)->SetOnMouseMove(OnMouseMove);
    }

    return FOUNDATION_ERROR;
}

WindowManager* WindowManager::Instance = 0;
```

4.15 WindowManager.h File Reference

```
#include "WindowAPI_Defs.h"
#include "Window.h"
```

Classes

- class WindowManager

4.16 WindowManager.h

```
#ifndef WINDOW_MANAGER_H
#define WINDOW_MANAGER_H


#include "WindowAPI_Defs.h"
#include "Window.h"

class FWindow;

class WindowManager
{
    friend FWindow;
    public:

    WindowManager();
    ~WindowManager();
        static void ShutDown();

        static FWindow* GetWindowByName(const char* WindowName);
        static FWindow* GetWindowByIndex(GLuint WindowIndex);

        static WindowManager* AddWindow(FWindow* NewWindow);

        //return the total amount of windows the manager has
        static GLuint GetNumWindows();

        //gets and sets for the mouse position in the screen
        static GLboolean GetMousePositionInScreen(GLuint& X, GLuint& Y);
        static GLuint* GetMousePositionInScreen();
        static GLboolean SetMousePositionInScreen(GLuint X, GLuint Y);

        // get the screen resolution for the screen that is being drawn to
        static GLuint* GetScreenResolution();
        static GLboolean GetScreenResolution(GLuint& Width, GLuint& Height);

        //these are another way to set and get window variables
        //apart from the functions that are available to the user
        //via each window

        //sets and gets for window resolution
        static GLboolean GetWindowResolution(const char* WindowName, GLuint& Width, GLuint& Height);
        static GLboolean GetWindowResolution(GLuint WindowIndex, GLuint& Width, GLuint& Height);
        static GLuint* GetWindowResolution(const char* WindowName);
        static GLuint* GetWindowResolution(GLuint WindowIndex);
        static GLboolean SetWindowResolution(const char* WindowName, GLuint Width, GLuint Height);
        static GLboolean SetWindowResolution(GLuint WindowIndex, GLuint Width, GLuint Height);

        //sets and gets for window position
        static GLboolean GetWindowPosition(const char* WindowName, GLuint& X, GLuint& Y);
        static GLboolean GetWindowPosition(GLuint WindowIndex, GLuint& X, GLuint& Y);
        static GLuint* GetWindowPosition(const char* WindowName);
        static GLuint* GetWindowPosition(GLuint WindowIndex);
        static GLboolean SetWindowPosition(const char* WindowName, GLuint X, GLuint Y);
        static GLboolean SetWindowPosition(GLuint WindowIndex, GLuint X, GLuint Y);

        //sets and gets for the mouse position in window
        static GLboolean GetMousePositionInWindow(const char* WindowName, GLuint& X, GLuint& Y);
        static GLboolean GetMousePositionInWindow(GLuint WindowIndex, GLuint& X, GLuint& Y);
        static GLuint* GetMousePositionInWindow(const char* WindowName);
        static GLuint* GetMousePositionInWindow(GLuint WindowIndex);
        static GLboolean SetMousePositionInWindow(const char* WindowName, GLuint X, GLuint Y);
        static GLboolean SetMousePositionInWindow(GLuint WindowIndex, GLuint X, GLuint Y);

        //gets for window keys
        static GLboolean WindowGetKey(const char* WindowName, GLuint Key);
        static GLboolean WindowGetKey(GLuint WindowIndex, GLuint Key);

        //gets for window should close
        static GLboolean GetWindowShouldClose(const char* WindowName);
        static GLboolean GetWindowShouldClose(GLuint WindowIndex);

        //swap buffers
        static GLboolean WindowSwapBuffers(const char* WindowName);
        static GLboolean WindowSwapBuffers(GLuint WindowIndex);

        //sets and gets for fullscreen
        static GLboolean SetFullScreen(const char* WindowName, GLboolean NewState);
        static GLboolean SetFullScreen(GLuint WindowIndex, GLboolean NewState);
        static GLboolean GetWindowIsFullScreen(const char* WindowName);
        static GLboolean GetWindowIsFullScreen(GLuint WindowIndex);

        //gets and sets for minimized
        static GLboolean GetWindowIsMinimized(const char* WindowName);
        static GLboolean GetWindowIsMinimized(GLuint WindowIndex);
        static GLboolean MinimizeWindow(const char* WindowName, GLboolean NewState);
        static GLboolean MinimizeWindow(GLuint WindowIndex, GLboolean NewState);

        //gets and sets for maximised state
        static GLboolean GetWindowIsMaximized(const char* WindowName);
        static GLboolean GetWindowIsMaximized(GLuint WindowIndex);
        static GLboolean MaximizeWindow(const char* WindowName, GLboolean NewState);
        static GLboolean MaximizeWindow(GLuint WindowIndex, GLboolean NewState);

        //gets and sets for window name and index
        static const char* GetWindowName(GLuint WindowIndex);
        static GLuint GetWindowIndex(const char*  WindowName);

        static GLboolean SetWindowTitleBar(const char* WindowName, const char* NewName);
        static GLboolean SetWindowTitleBar(GLuint WindowIndex, const char* NewName);

        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);

        //gets and sets window is in focus(Linux only?)
        static GLboolean GetWindowIsInFocus(const char* WindowName);
        static GLboolean GetWindowIsInFocus(GLuint WindowIndex);
        static GLboolean FocusWindow(const char* WindowName, GLboolean NewState);
        static GLboolean FocusWindow(GLuint WindowIndex, GLboolean NewState);

        //gets and sets for restoring the window
        static GLboolean RestoreWindow(const char* WindowName);
        static GLboolean RestoreWindow(GLuint WindowIndex);

        //get window obscurity. I feel like this is completely useless
        //static GLboolean GetWindowIsObscured(const char* WindowName);
        //static GLboolean GetWindowIsObscured(GLuint WindowIndex);

        //enable vertical sync on selected window
        static GLboolean SetWindowSwapInterval(const char* WindowName, GLint
EnableSync);
        static GLboolean SetWindowSwapInterval(GLuint WindowIndex, GLint EnableSync);

        //initialize the window manager
        static GLboolean Initialize();
        static GLboolean IsInitialized();

        //ask the window to poll for window events
        static GLboolean PollForEvents();

        //remove a window from the manager
        static GLboolean RemoveWindow(FWindow* WindowToBeRemoved);

        //set the styleof the given window
        static GLboolean SetWindowStyle(const char* WindowName, GLuint WindowStyle);
        static GLboolean SetWindowStyle(GLuint WindowIndex, GLuint WindowStyle);

        //enable the given decorators of the given window
        static GLboolean EnableWindowDecorator(const char* WindowName, GLbitfield
Decorators);
        static GLboolean EnableWindowDecorator(GLuint WindowIndex, GLbitfield
Decorators);

        //disable the given decorators of the given window
        static GLboolean DisableWindowDecorator(const char* WindowName, GLbitfield
Decorators);
        static GLboolean DisableWindowDecorator(GLuint WindowIndex, GLbitfield
Decorators);

        //set callbacks for the selected window
        static GLboolean SetWindowOnKeyEvent(const char* WindowName, OnKeyEvent OnKey);
        static GLboolean SetWindowOnKeyEvent(GLuint WindowIndex, OnKeyEvent OnKey);

        static GLboolean SetWindowOnMouseButtonEvent(const char* WindowName, OnMouseButtonEvent a_OnMouseButtonEvent);
        static GLboolean SetWindowOnMouseButtonEvent(GLuint WindowIndex, OnMouseButtonEvent a_OnMouseButtonEvent);

        static GLboolean SetWindowOnMouseWheelEvent(const char* WindowName, OnMouseWheelEvent OnMouseWheelEvent);
        static GLboolean SetWindowOnMouseWheelEvent(GLuint WindowIndex, OnMouseWheelEvent OnMouseWheelEvent);

        static GLboolean SetWindowOnDestroyed(const char* WindowName, OnDestroyedEvent OnDestroyed);
        static GLboolean SetWindowOnDestroyed(GLuint WindowIndex, OnDestroyedEvent OnDestroyed);

        static GLboolean SetWindowOnMaximized(const char* WindowName, OnMaximizedEvent OnMaximized);
        static GLboolean SetWindowOnMaximized(GLuint WindowIndex, OnMaximizedEvent OnMaximized);

        static GLboolean SetWindowOnMinimized(const char* WindowName, OnMinimizedEvent a_OnMiniimzed);
        static GLboolean SetWindowOnMinimized(GLuint WindowIndex, OnMinimizedEvent a_OnMiniimzed);

    //  static void SetWindowOnRestored(const char* WindowName, OnRestoredEvent OnRestored);
        //static void SetWindowOnRestored(GLuint WindowIndex, OnRestoredEvent OnRestored);

        static GLboolean SetWindowOnFocus(const char* WindowName, OnFocusEvent OnFocus);
        static GLboolean SetWindowOnFocus(GLuint WindowIndex, OnFocusEvent OnFocus);

        static GLboolean SetWindowOnMoved(const char* WindowName, OnMovedEvent OnMoved);
        static GLboolean SetWindowOnMoved(GLuint WindowIndex, OnMovedEvent OnMoved);

        static GLboolean SetWindowOnResize(const char* WindowName, OnResizeEvent OnResize);
        static GLboolean SetWindowOnResize(GLuint WindowIndex, OnResizeEvent OnResize);

        static GLboolean SetWindowOnMouseMove(const char* WindowName, OnMouseMoveEvent OnMouseMove);
        static GLboolean SetWindowOnMouseMove(GLuint WindowIndex, OnMouseMoveEvent OnMouseMove);

    private:

        //make sure the window exists in the window manager
        static GLboolean DoesExist(const char* WindowName);
        static GLboolean DoesExist(GLuint WindowIndex);

        //get a static reference to the window manager
        static WindowManager* GetInstance();

        std::list<FWindow*> Windows;
        static WindowManager* Instance;
        GLuint ScreenResolution[2];
        GLuint ScreenMousePosition[2];
        GLboolean Initialized;
#if defined(CURRENT_OS_WINDOWS)
        LRESULT CALLBACK WindowProcedure(HWND WindowHandle, GLuint Message, WPARAM WordParam, LPARAM LongParam);

        static LRESULT CALLBACK StaticWindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam, LPARAM LongParam);

        static FWindow* GetWindowByHandle(HWND WindowHandle);

        static GLboolean Windows_PollForEvents();
        static GLboolean Windows_Initialize();
        static GLboolean Windows_Shutdown();
        static GLboolean Windows_SetMousePositionInScreen(GLuint X, GLuint Y);

        static void CreateTerminal();
        static GLuint Windows_TranslateKey(WPARAM WordParam, LPARAM LongParam);

        HDC DeviceContextHandle;
        MSG Message;
#endif
```

```
#if defined(CURRENT_OS_LINUX)
        static FWindow* GetWindowByHandle(Window WindowHandle);
        static FWindow* GetWindowByEvent(XEvent Event);

        static GLboolean Linux_Initialize();
        static void Linux_Shutdown();

        static GLboolean Linux_PollForEvents();
        static GLboolean Linux_SetMousePositionInScreen(GLuint X, GLuint Y);
        static Display* GetDisplay();

        static GLuint Linux_TranslateKey(GLuint KeySym);
        static const char* Linux_GetEventType(XEvent Event);

        Display* m_Display;
        XEvent m_Event;
#endif
};
#endif
```

4.17 WindowManager_Linux.cpp File Reference

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

4.18 WindowManager_Linux.cpp

```
/**********************************************************************************************/
#include "WindowManager.h"

#include <limits.h>
#if defined(CURRENT_OS_LINUX)

/**********************************************************************************************/
FWindow* WindowManager::GetWindowByHandle(Window WindowHandle)
{
    if(GetInstance()->IsInitialized())
    {
    for (auto Iter : GetInstance()->Windows)
    {
        if (Iter->GetWindowHandle() == WindowHandle)
        {
            return Iter;
        }
    }

    return nullptr;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}

/**********************************************************************************************/
GLboolean WindowManager::Linux_Initialize()
{
    GetInstance()->m_Display = XOpenDisplay(0);

    if(!GetInstance()->m_Display)
    {
        PrintErrorMessage(ERROR_LINUX_CANNOTCONNECTXSERVER
);
        return FOUNDATION_ERROR;
    }

    GetInstance()->ScreenResolution[0] = WidthOfScreen(XScreenOfDisplay(
GetInstance()->m_Display,
                DefaultScreen(GetInstance()->m_Display)));

    GetInstance()->ScreenResolution[1] = HeightOfScreen(XScreenOfDisplay(
GetInstance()->m_Display,
                DefaultScreen(GetInstance()->m_Display)));

    GetInstance()->Initialized = GL_TRUE;

    return FOUNDATION_OKAY;
```

```
}

/*************************************************************************************************/
GLboolean WindowManager::Linux_SetMousePositionInScreen(GLuint X, GLuint Y)
{
    if(GetInstance()->IsInitialized())
    {
        XWarpPointer(GetInstance()->m_Display, None,
            XDefaultRootWindow(GetInstance()->m_Display), 0, 0,
            GetScreenResolution()[0],
            GetScreenResolution()[1],
            X, Y);
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return FOUNDATION_ERROR;
}

/*************************************************************************************************/
void WindowManager::Linux_Shutdown()
{
    XCloseDisplay(GetInstance()->m_Display);
}

/*************************************************************************************************/
FWindow* WindowManager::GetWindowByEvent(XEvent Event)
{
    if(GetInstance()->IsInitialized())
    {
    switch(Event.type)
    {
        case Expose:
            {
                return GetWindowByHandle(Event.xexpose.window);
            }

        case DestroyNotify:
            {
                return GetWindowByHandle(Event.xdestroywindow.window);
            }

        case CreateNotify:
            {
                return GetWindowByHandle(Event.xcreatewindow.window);
            }

        case KeyPress:
            {
                return GetWindowByHandle(Event.xkey.window);
            }

        case KeyRelease:
            {
                return GetWindowByHandle(Event.xkey.window);
            }

        case ButtonPress:
            {
                return GetWindowByHandle(Event.xbutton.window);
            }

        case ButtonRelease:
            {
                return GetWindowByHandle(Event.xbutton.window);
            }

        case MotionNotify:
            {
                return GetWindowByHandle(Event.xmotion.window);
            }

        case FocusIn:
            {
                return GetWindowByHandle(Event.xfocus.window);
            }

        case FocusOut:
            {
                return GetWindowByHandle(Event.xfocus.window);
            }

        case ResizeRequest:
            {
                return GetWindowByHandle(Event.xresizerequest.window);
            }

        case ConfigureNotify:
            {
                return GetWindowByHandle(Event.xconfigure.window);
            }

        case PropertyNotify:
            {
                return GetWindowByHandle(Event.xproperty.window);
            }

        case GravityNotify:
            {
                return GetWindowByHandle(Event.xgravity.window);
            }

        case ClientMessage:
            {
                return GetWindowByHandle(Event.xclient.window);
            }

        case VisibilityNotify:
            {
                return GetWindowByHandle(Event.xvisibility.window);
            }

    default:
        {
            return nullptr;
        }
    }
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
    return nullptr;
}

/***********************************************************************************************/
Display* WindowManager::GetDisplay()
{
    return GetInstance()->m_Display;
}

/***********************************************************************************************/
GLboolean WindowManager::Linux_PollForEvents()
{
    if(GetInstance()->IsInitialized())
    {
    XNextEvent(GetInstance()->m_Display, &GetInstance()->m_Event);

    XEvent l_Event = GetInstance()->m_Event;
    FWindow* l_Window = GetWindowByEvent(l_Event);

    switch (l_Event.type)
    {
        case Expose:
        {
            break;
        }

        case DestroyNotify:
        {

            if(IsValidDestroyedEvent(l_Window->
DestroyedEvent))
            {
                l_Window->DestroyedEvent();
            }

            l_Window->Shutdown();
            break;
        }

        /*case CreateNotify:
        {
            printf("FWindow was created\n");
            l_Window->InitializeGL();

            if(Foundation_Tools::IsValid(l_Window->m_OnCreated))
            {
                l_Window->m_OnCreated();
            }

            break;
        }*/

        case KeyPress:
        {
            GLuint l_FunctionKeysym = XKeycodeToKeysym(
                    GetInstance()->m_Display, l_Event.xkey.keycode, 1);

            if(l_FunctionKeysym <= 255)
            {
                l_Window->Keys[l_FunctionKeysym] = KEYSTATE_DOWN;
                if(IsValidKeyEvent(l_Window->KeyEvent))
                {
                    l_Window->KeyEvent(l_FunctionKeysym, KEYSTATE_DOWN);
                }
            }

            else
            {
                l_Window->Keys[
                    Linux_TranslateKey(l_FunctionKeysym)] = KEYSTATE_DOWN;

                if(IsValidKeyEvent(l_Window->KeyEvent))
                {
                    l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
KEYSTATE_DOWN);
                }
            }

            break;
        }

        case KeyRelease:
        {
            GLboolean l_IsRetriggered = GL_FALSE;
            if(XEventsQueued(GetInstance()->m_Display, QueuedAfterReading))
            {
                XEvent l_NextEvent;
                XPeekEvent(GetInstance()->m_Display, &l_NextEvent);

                if(l_NextEvent.type == KeyPress &&
                        l_NextEvent.xkey.time == l_Event.xkey.time &&
                        l_NextEvent.xkey.keycode == l_Event.xkey.keycode)
                {
                    XNextEvent(GetInstance()->m_Display, &l_Event);
                    l_IsRetriggered = GL_TRUE;
                }
            }

            if(!l_IsRetriggered)
            {
                GLuint l_FunctionKeysym = XKeycodeToKeysym(GetInstance()->m_Display,
                       l_Event.xkey.keycode, 1);

                if(l_FunctionKeysym <= 255)
                {
                    l_Window->Keys[l_FunctionKeysym] = KEYSTATE_UP;

                    if(IsValidKeyEvent(l_Window->KeyEvent))
                    {
                        l_Window->KeyEvent(l_FunctionKeysym, KEYSTATE_UP);
                    }
                }

                else
                {
                    l_Window->Keys[
                    Linux_TranslateKey(l_FunctionKeysym)] = KEYSTATE_UP;

                    if(IsValidKeyEvent(l_Window->KeyEvent))
                    {
                        l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
KEYSTATE_UP);
                    }
                }

                if(IsValidKeyEvent(l_Window->KeyEvent))
                {
                    l_Window->KeyEvent(Linux_TranslateKey(l_FunctionKeysym),
KEYSTATE_UP);
                }
            }

            break;
        }

        case ButtonPress:
        {
            switch(l_Event.xbutton.button)
            {
                case 1:
                    {
                        l_Window->MouseButton[MOUSE_LEFTBUTTON] =
MOUSE_BUTTONDOWN;

                        if(IsValidKeyEvent(l_Window->MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(MOUSE_LEFTBUTTON, MOUSE_BUTTONDOWN);
                        }
                        break;
                    }

                case 2:
                    {
                        l_Window->MouseButton[MOUSE_MIDDLEBUTTON] = MOUSE_BUTTONDOWN;

                        if(IsValidKeyEvent(l_Window->MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(MOUSE_MIDDLEBUTTON, MOUSE_BUTTONDOWN);
                        }
                        break;
                    }

                case 3:
                    {
                        l_Window->MouseButton[MOUSE_RIGHTBUTTON] = MOUSE_BUTTONDOWN;

                        if(IsValidKeyEvent(l_Window->MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(MOUSE_RIGHTBUTTON, MOUSE_BUTTONDOWN);
                        }
                        break;
                    }

                case 4:
                    {
                        l_Window->MouseButton[MOUSE_SCROLL_UP] = MOUSE_BUTTONDOWN;

                        if(IsValidMouseWheelEvent(l_Window->MouseWheelEvent))
                        {
                            l_Window->MouseWheelEvent(MOUSE_SCROLL_UP);
                        }
                        break;
                    }

                case 5:
                    {
                        l_Window->MouseButton[MOUSE_SCROLL_DOWN] = MOUSE_BUTTONDOWN;

                        if(IsValidMouseWheelEvent(l_Window->MouseWheelEvent))
                        {
                            l_Window->MouseWheelEvent(MOUSE_SCROLL_DOWN);
                        }
                        break;
                    }

                default:
                    {
                        break;
                    }
            }

            break;
        }

        case ButtonRelease:
        {
            switch(l_Event.xbutton.button)
            {
                case 1:
                    {
                        l_Window->MouseButton[MOUSE_LEFTBUTTON] = MOUSE_BUTTONUP;

                        if(IsValidKeyEvent(l_Window->MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(
MOUSE_LEFTBUTTON, MOUSE_BUTTONUP);
                        }
                        break;
                    }

                case 2:
                    {
                        l_Window->MouseButton[MOUSE_MIDDLEBUTTON] =
MOUSE_BUTTONUP;

                        if(IsValidKeyEvent(l_Window->
MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(
MOUSE_MIDDLEBUTTON, MOUSE_BUTTONUP);
                        }
                        break;
                    }

                case 3:
                    {
                        l_Window->MouseButton[MOUSE_RIGHTBUTTON] =
MOUSE_BUTTONUP;

                        if(IsValidKeyEvent(l_Window->
MouseButtonEvent))
                        {
                            l_Window->MouseButtonEvent(
MOUSE_RIGHTBUTTON, MOUSE_BUTTONUP);
                        }
                        break;
                    }

                case 4:
                    {
                        l_Window->MouseButton[MOUSE_SCROLL_UP] =
MOUSE_BUTTONDOWN;
                        break;
                    }

                case 5:
                    {
                        l_Window->MouseButton[MOUSE_SCROLL_DOWN] =
MOUSE_BUTTONDOWN;
                        break;
                    }

                default:
                    {
                        break;
                    }
            }
            break;
        }

        //when the mouse/pointer device is moved
        case MotionNotify:
        {
            //set the windows mouse position to match the event
            l_Window->MousePosition[0] =
                l_Event.xmotion.x;

            l_Window->MousePosition[1] =
                l_Event.xmotion.y;

            GetInstance()->ScreenMousePosition[0] = l_Event.xmotion.x_root;
            GetInstance()->ScreenMousePosition[1] = l_Event.xmotion.y_root;

            if(IsValidMouseMoveEvent(l_Window->
MouseMoveEvent))
            {
                l_Window->MouseMoveEvent(l_Event.xmotion.x,
                        l_Event.xmotion.y, l_Event.xmotion.x_root,
                        l_Event.xmotion.y_root);
            }
            break;
        }

        //when the window goes out of focus
        case FocusOut:
        {
            l_Window->InFocus = GL_FALSE;
            if(IsValidFocusEvent(l_Window->FocusEvent))
            {
                l_Window->FocusEvent(
                    l_Window->InFocus);
            }
            break;
        }

        //when the window is back in focus (use to restore?)
        case FocusIn:
        {
            l_Window->InFocus = GL_TRUE;

            if(IsValidFocusEvent(l_Window->FocusEvent))
            {
                l_Window->FocusEvent(l_Window->InFocus);
            }
            break;
        }

        //when a request to resize the window is made either by
        //dragging out the window or programmatically
        case ResizeRequest:
        {
            glViewport(0, 0,
                    l_Window->GetResolution()[0],
                    l_Window->GetResolution()[1]);

            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();

            break;
        }

        //when a request to configure the window is made
        case ConfigureNotify:
        {
            glViewport(0, 0, l_Event.xconfigure.width,
                        l_Event.xconfigure.height);

            //check if window was resized
            if((GLuint)l_Event.xconfigure.width != l_Window->Resolution[0]
                    || (GLuint)l_Event.xconfigure.height != l_Window->Resolution[1])
            {
                if(IsValidMovedEvent(l_Window->ResizeEvent))
                {
                    l_Window->ResizeEvent(l_Event.xconfigure.width, l_Event.xconfigure.height);
                }

                l_Window->Resolution[0] = l_Event.xconfigure.width;
                l_Window->Resolution[1] = l_Event.xconfigure.height;
            }

            //check if window was moved
            if((GLuint)l_Event.xconfigure.x != l_Window->Position[0]
                    || (GLuint)l_Event.xconfigure.y != l_Window->Position[1])
            {
                if(IsValidMovedEvent(l_Window->MovedEvent))
                {
                    l_Window->MovedEvent(l_Event.xconfigure.x, l_Event.xconfigure.y);
                }

                l_Window->Position[0] = l_Event.xconfigure.x;
                l_Window->Position[1] = l_Event.xconfigure.y;
            }
            break;
        }

        case PropertyNotify:
        {
            //this is needed in order to read from the windows WM_STATE Atomic
            //to determine if the property notify event was caused by a client
            //iconify event(minimizing the window) or a maximise event

            Atom l_Type;
            GLint l_Format;
            ulong l_NumItems, l_BytesAfter;
            unsigned char* l_Properties = nullptr;

            XGetWindowProperty(WindowManager::GetDisplay(), l_Event.xproperty.window,
                        l_Window->AtomState,
                        0, LONG_MAX, GL_FALSE, AnyPropertyType,
                        &l_Type, &l_Format, &l_NumItems, &l_BytesAfter,
                        &l_Properties);

                if(l_Properties && (l_Format == 32))
                {
                    for(GLuint l_CurrentItem = 0; l_CurrentItem < l_NumItems; l_CurrentItem++)
                    {
                        long l_Property = ((long*)(l_Properties))[l_CurrentItem];
                        if(l_Property == l_Window->AtomHidden)
                        {
                            if(IsValidDestroyedEvent(l_Window->
      MinimizedEvent))
                            {
                                l_Window->MinimizedEvent();
                            }
                        }

                        if(l_Property == l_Window->AtomMaxVert ||
                                l_Property == l_Window->AtomMaxVert)
                        {
                            if(IsValidDestroyedEvent(l_Window->
      MaximizedEvent))
                            {
                                l_Window->MaximizedEvent();
                            }
                        }

                        if(l_Property == l_Window->AtomDemandsAttention)
                        {
                            //printf("window demands attention \n");
                        }
                    }
                }

            break;
        }

        case GravityNotify:
        {
            //this is only supposed to pop up when the parent of this window(if any) has something happen
            //to it so that this window can react to said event as well.
            break;
        }

        case ClientMessage:
        {
            const char* l_AtomName = XGetAtomName(WindowManager::GetDisplay(), l_Event.xclient.message_type
      );
            if(IsValidString(l_AtomName))
            {
                //print the name of the Atom
                //printf("%s\n", l_AtomName);
            }

            if((Atom)l_Event.xclient.data.l[0] == l_Window->AtomClose)
            {
                //printf("window closed\n");
                l_Window->ShouldClose = GL_TRUE;
                l_Window->DestroyedEvent();
                l_Window->Shutdown();
                //XDestroyWindow(GetInstance()->m_Display, l_Event.xclient.window);
                break;
            }

            if((Atom)l_Event.xclient.data.l[1] == l_Window->AtomFullScreen)
            {
                //printf("resized window \n");
                break;
            }
            break;
        }

        /*case VisibilityNotify:
        {
            if(l_Event.xvisibility.state == VisibilityUnobscured)
            {
                //printf("window not obscured \n");
                l_Window->m_IsObscured = GL_FALSE;
            }

            else
            {
                //printf("window obscured\n");
                l_Window->m_IsObscured = GL_TRUE;
            }
        }*/

        default:
        {
            break;
        }
    }
    return FOUNDATION_OKAY;
    }
    PrintErrorMessage(ERROR_NOTINITIALIZED);
return FOUNDATION_ERROR;

}

const char* WindowManager::Linux_GetEventType(XEvent Event)
{
    switch (Event.type)
    {
        case MotionNotify:
        {
            return "Motion Notify Event\n";
        }

        case ButtonPress:
        {
            return "Button Press Event\n";
        }

        case ButtonRelease:
        {
            return "Button Release Event\n";
        }

        case ColormapNotify:
        {
            return "Color Map Notify event \n";
        }

        case EnterNotify:
        {
            return "Enter Notify Event\n";
        }

        case LeaveNotify:
        {
            return "Leave Notify Event\n";
        }

        case Expose:
        {
            return "Expose Event\n";
        }

        case GraphicsExpose:
        {
            return "Graphics expose event\n";
        }

        case NoExpose:
        {
            return "No Expose Event\n";
        }

        case FocusIn:
        {
            return "Focus In Event\n";
        }

        case FocusOut:
        {
            return "Focus Out Event\n";
        }

        case KeymapNotify:
        {
            return "Key Map Notify Event\n";
        }

        case KeyPress:
        {
            return "Key Press Event\n";
        }

        case KeyRelease:
        {
            return "Key Release Event\n";
        }

        case PropertyNotify:
        {
            return "Property Notify Event\n";
        }

        case ResizeRequest:
        {
            return "Resize Property Event\n";
        }

        case CirculateNotify:
        {
            return "Circulate Notify Event\n";
        }

        case ConfigureNotify:
        {
            return "configure Notify Event\n";
        }

        case DestroyNotify:
        {
            return "Destroy Notify Request\n";
        }

        case GravityNotify:
        {
            return "Gravity Notify Event \n";
        }

        case MapNotify:
        {
            return "Map Notify Event\n";
        }

        case ReparentNotify:
        {
            return "Reparent Notify Event\n";
        }

        case UnmapNotify:
        {
            return "Unmap notify event\n";
        }

        case MapRequest:
        {
            return "Map request event\n";
        }

        case ClientMessage:
        {
            return "Client Message Event\n";
        }

        case MappingNotify:
        {
            return "Mapping notify event\n";
        }

        case SelectionClear:
        {
            return "Selection Clear event\n";
        }

        case SelectionNotify:
        {
            return "Selection Notify Event\n";
        }

        case SelectionRequest:
        {
            return "Selection Request event\n";
        }

        case VisibilityNotify:
        {
            return "Visibility Notify Event\n";
        }

        default:
        {
            return 0;
        }
    }
}

GLuint WindowManager::Linux_TranslateKey(GLuint KeySym)
{
    switch (KeySym)
    {
        case XK_Escape:
        {
            return KEY_ESCAPE;
        }

        case XK_Home:
        {
            return KEY_HOME;
        }

        case XK_Left:
        {
            return KEY_ARROW_LEFT;
        }

        case XK_Right:
        {
            return KEY_ARROW_RIGHT;
        }

        case XK_Up:
        {
            return KEY_ARROW_UP;
        }

        case XK_Down:
        {
            return KEY_ARROW_DOWN;
        }

        case XK_Page_Up:
        {
            return KEY_PAGEUP;
        }

        case XK_Page_Down:
        {
            return KEY_PAGEDOWN;
        }

        case XK_End:
        {
            return KEY_END;
        }

        case XK_Print:
        {
            return KEY_PRINTSCREEN;
        }

        case XK_Insert:
        {
            return KEY_INSERT;
        }

        case XK_Num_Lock:
        {
            return KEY_NUMLOCK;
        }

        case XK_KP_Multiply:
        {
            return KEY_KEYPAD_MULTIPLY;
        }

        case XK_KP_Add:
        {
            return KEY_KEYPAD_ADD;
        }

        case XK_KP_Subtract:
        {
            return KEY_KEYPAD_SUBTRACT;
        }

        case XK_KP_Decimal:
        {
            return KEY_KEYPAD_PERIOD;
        }

        case XK_KP_Divide:
        {
            return KEY_KEYPAD_DIVIDE;
        }

        case XK_KP_0:
        {
            return KEY_KEYPAD_0;
        }

        case XK_KP_1:
```

```
        {
            return KEY_KEYPAD_1;
        }

        case XK_KP_2:
        {
            return KEY_KEYPAD_2;
        }

        case XK_KP_3:
        {
            return KEY_KEYPAD_3;
        }

        case XK_KP_4:
        {
            return KEY_KEYPAD_4;
        }

        case XK_KP_5:
        {
            return KEY_KEYPAD_5;
        }

        case XK_KP_6:
        {
            return KEY_KEYPAD_6;
        }

        case XK_KP_7:
        {
            return KEY_KEYPAD_7;
        }

        case XK_KP_8:
        {
            return KEY_KEYPAD_8;
        }

        case XK_KP_9:
        {
            return KEY_KEYPAD_9;
        }

        case XK_F1:
        {
            return KEY_F1;
        }

        case XK_F2:
        {
            return KEY_F2;
        }

        case XK_F3:
        {
            return KEY_F3;
        }

        case XK_F4:
        {
            return KEY_F4;
        }

        case XK_F5:
        {
            return KEY_F5;
        }

        case XK_F6:
        {
            return KEY_F6;
        }

        case XK_F7:
        {
            return KEY_F7;
        }

        case XK_F8:
        {
            return KEY_F8;
        }

        case XK_F9:
        {
            return KEY_F9;
        }

        case XK_F10:
        {
            return KEY_F10;
        }

        case XK_F11:
        {
            return KEY_F11;
        }

        case XK_F12:
        {
            return KEY_F12;
        }

        case XK_Shift_L:
        {
            return KEY_LEFTSHIFT;
        }

        case XK_Shift_R:
        {
            return KEY_RIGHTSHIFT;
        }

        case XK_Control_R:
        {
            return KEY_RIGHTCONTROL;
        }

        case XK_Control_L:
        {
            return KEY_LEFTCONTROL;
        }

        case XK_Caps_Lock:
        {
            return KEY_CAPSLOCK;
        }

        case XK_Alt_L:
        {
            return KEY_LEFTALT;
        }

        case XK_Alt_R:
        {
            return KEY_RIGHTALT;
        }

        default:
        {
            return 0;
        }
    }
}

#endif
```

4.19 WindowManager_Windows.cpp File Reference

```
#include "WindowManager.h"
```

4.20 WindowManager_Windows.cpp

```
/*********************************************************************************************/
#include "WindowManager.h"

#if defined(CURRENT_OS_WINDOWS)

/*********************************************************************************************/
GLboolean WindowManager::Windows_Initialize()
{
    CreateTerminal();
    RECT l_Desktop;

    HWND l_DesktopHandle = GetDesktopWindow();


    if (l_DesktopHandle)
    {
        GetWindowRect(l_DesktopHandle, &l_Desktop);

        GetInstance()->ScreenResolution[0] = l_Desktop.right;
        GetInstance()->ScreenResolution[1] = l_Desktop.bottom;

        GetInstance()->Initialized = GL_TRUE;
        return FOUNDATION_OKAY;
    }

    PrintErrorMessage(ERROR_WINDOWS_CANNOTINITIALIZE);
    return FOUNDATION_ERROR;
}

/**************************************************************************************************/
FWindow* WindowManager::GetWindowByHandle(HWND WindowHandle)
{
    if(GetInstance()->IsInitialized())
    {
#if defined(CURRENT_OS_WINDOWS)

    for each(auto CurrentWindow in GetInstance()->Windows)
    {
        if (CurrentWindow->WindowHandle == WindowHandle)
        {
            return CurrentWindow;
        }
    }

#endif

#if defined(CURRENT_OS_LINUX)
    for (auto CurrentWindow : GetInstance()->Windows)
    {
        if (CurrentWindow->WindowHandle == WindowHandle)
        {
            return CurrentWindow;
        }
    }

#endif

    return nullptr;
    }

    return nullptr;
}

LRESULT CALLBACK WindowManager::WindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam, LPARAM
LongParam)
{
    FWindow* l_Window = GetWindowByHandle(WindowHandle);
    switch (Message)
    {
    case WM_CREATE:
    {
        GetWindowByIndex(GetInstance()->Windows.size() - 1)->WindowHandle
 = WindowHandle;
        l_Window = GetWindowByHandle(WindowHandle);

        l_Window->InitializeGL();
        break;
    }

    case WM_DESTROY:
    {
        l_Window->ShouldClose = GL_TRUE;

        if (IsValidDestroyedEvent(l_Window->DestroyedEvent))
        {
            l_Window->DestroyedEvent();
        }

        l_Window->Shutdown();
        return 0;
    }
    case WM_MOVE:
    {
        l_Window->Position[0] = LOWORD(LongParam);
        l_Window->Position[1] = HIWORD(LongParam);

        if (IsValidMovedEvent(l_Window->MovedEvent))
        {
            l_Window->MovedEvent(l_Window->Position[0], l_Window->Position[1]);
        }

        break;
    }

    case WM_MOVING:
    {
        l_Window->Position[0] = LOWORD(LongParam);
        l_Window->Position[1] = HIWORD(LongParam);

        if (IsValidMovedEvent(l_Window->MovedEvent))
        {
            l_Window->MovedEvent(l_Window->Position[0], l_Window->Position[1]);
        }
        break;
    }

    case WM_SIZE:
    {
        l_Window->Resolution[0] = (GLuint)LOWORD(LongParam);
        l_Window->Resolution[1] = (GLuint)HIWORD(LongParam);

        switch (WordParam)
        {
            case SIZE_MAXIMIZED:
            {
                if (IsValidDestroyedEvent(l_Window->MaximizedEvent))
                {
                    l_Window->MaximizedEvent();
                }

                break;
            }

            case SIZE_MINIMIZED:
            {
                if (IsValidDestroyedEvent(l_Window->MinimizedEvent))
                {
                    l_Window->MinimizedEvent();
                }
                break;
            }

            /*case SIZE_RESTORED:
            {
                if (IsValid(l_Window->MaximizedEvent))
                {
                    l_Window->RestoredEvent();
                }
                break;
            }*/

            default:
            {
                if (IsValidMovedEvent(l_Window->ResizeEvent))
                {
                    l_Window->ResizeEvent(l_Window->Resolution[0],
                        l_Window->Resolution[1]);
                }

                break;
            }
        }
        break;
    }

    case WM_SIZING:
    {

        l_Window->Resolution[0] = (GLuint)LOWORD(LongParam);
        l_Window->Resolution[1] = (GLuint)HIWORD(LongParam);

        if (IsValidMovedEvent(l_Window->ResizeEvent))
        {
            l_Window->ResizeEvent(l_Window->Resolution[0],
                l_Window->Resolution[1]);
        }


        break;
    }

    case WM_KEYDOWN:
    {
        GLuint l_TranslatedKey = 0;
```

```

            switch (HIWORD(LongParam))
            {
                case 29:
                {
                    l_Window->Keys[KEY_LEFTCONTROL] = KEYSTATE_DOWN;
                    l_TranslatedKey = KEY_LEFTCONTROL;
                    break;
                }

                case 285:
                {
                    l_Window->Keys[KEY_RIGHTCONTROL] = KEYSTATE_DOWN;
                    l_TranslatedKey = KEY_RIGHTCONTROL;
                    break;
                }

                case 42:
                {
                    l_Window->Keys[KEY_LEFTSHIFT] = KEYSTATE_DOWN;
                    l_TranslatedKey = KEY_LEFTSHIFT;
                    break;
                }

                case 54:
                {
                    l_Window->Keys[KEY_RIGHTSHIFT] = KEYSTATE_DOWN;
                    l_TranslatedKey = KEY_RIGHTSHIFT;
                    break;
                }

                default:
                {
                    l_TranslatedKey = Windows_TranslateKey(WordParam, LongParam);
                    l_Window->Keys[l_TranslatedKey] = KEYSTATE_DOWN;
                    break;
                }
            }

            if (IsValidKeyEvent(l_Window->KeyEvent))
            {
                l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_DOWN);
            }
            break;
        }

        case WM_KEYUP:
        {
            GLuint l_TranslatedKey = 0;

            switch (HIWORD(LongParam))
            {
                case 49181:
                {
                    l_Window->Keys[KEY_LEFTCONTROL] = KEYSTATE_UP;
                    l_TranslatedKey = KEY_LEFTCONTROL;
                    break;
                }

                case 49437:
                {
                    l_Window->Keys[KEY_RIGHTCONTROL] = KEYSTATE_UP;
                    l_TranslatedKey = KEY_RIGHTCONTROL;
                    break;
                }

                case 49194:
                {
                    l_Window->Keys[KEY_LEFTSHIFT] = KEYSTATE_UP;
                    l_TranslatedKey = KEY_LEFTSHIFT;
                    break;
                }

                case 49206:
                {
                    l_Window->Keys[KEY_RIGHTSHIFT] = KEYSTATE_UP;
                    l_TranslatedKey = KEY_RIGHTSHIFT;
                    break;
                }

                default:
                {
                    l_TranslatedKey = Windows_TranslateKey(WordParam, LongParam);
                    l_Window->Keys[l_TranslatedKey] = KEYSTATE_UP;
                    break;
                }
            }

            if (IsValidKeyEvent(l_Window->KeyEvent))
            {
                l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_UP);
            }
            break;
        }

        case WM_SYSKEYDOWN:
        {
            GLuint l_TranslatedKey = 0;
            switch (HIWORD(LongParam))
            {
            case 8248:
            {
                l_Window->Keys[KEY_LEFTALT] = KEYSTATE_DOWN;
                l_TranslatedKey = KEY_LEFTALT;
                break;
            }


            case 8504:
            {
                l_Window->Keys[KEY_RIGHTALT] = KEYSTATE_DOWN;
                l_TranslatedKey = KEY_RIGHTALT;
            }

            default:
            {
                break;
            }
            }

            if (IsValidKeyEvent(l_Window->KeyEvent))
            {
                l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_DOWN);
            }

            break;
        }

        case WM_SYSKEYUP:
        {
            GLuint l_TranslatedKey = 0;
            switch (HIWORD(LongParam))
            {
            case 49208:
            {
                l_Window->Keys[KEY_LEFTALT] = KEYSTATE_UP;
                l_TranslatedKey = KEY_LEFTALT;
                break;
            }


            case 49464:
            {
                l_Window->Keys[KEY_RIGHTALT] = KEYSTATE_UP;
                l_TranslatedKey = KEY_RIGHTALT;
                break;
            }

            default:
            {
                break;
            }
            }

            if (IsValidKeyEvent(l_Window->KeyEvent))
            {
                l_Window->KeyEvent(l_TranslatedKey, KEYSTATE_UP);
            }
            break;
        }

        case WM_MOUSEMOVE:
        {
            l_Window->MousePosition[0] = (GLuint)LOWORD(LongParam);
            l_Window->MousePosition[1] = (GLuint)HIWORD(LongParam);

            POINT l_Point;
            l_Point.x = l_Window->MousePosition[0];
            l_Point.y = l_Window->MousePosition[1];

            ClientToScreen(WindowHandle, &l_Point);

            //printf("%i %i \n", l_Point.x, l_Point.y);

            if (IsValidMouseMoveEvent(l_Window->MouseMoveEvent))
            {
                l_Window->MouseMoveEvent(l_Window->MousePosition[0],
                    l_Window->MousePosition[1], l_Point.x, l_Point.y);
            }
            break;
        }

        case WM_LBUTTONDOWN:
        {
            l_Window->MouseButton[MOUSE_LEFTBUTTON] =
    MOUSE_BUTTONDOWN;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_LEFTBUTTON,
    MOUSE_BUTTONDOWN);
            }
            break;
        }

        case WM_LBUTTONUP:
        {
            l_Window->MouseButton[MOUSE_LEFTBUTTON] = MOUSE_BUTTONUP;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_LEFTBUTTON,
    MOUSE_BUTTONUP);
            }
            break;
        }

        case WM_RBUTTONDOWN:
        {
            l_Window->MouseButton[MOUSE_RIGHTBUTTON] =
    MOUSE_BUTTONDOWN;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_RIGHTBUTTON,
    MOUSE_BUTTONDOWN);
            }
            break;
        }

        case WM_RBUTTONUP:
        {
            l_Window->MouseButton[MOUSE_RIGHTBUTTON] =
    MOUSE_BUTTONUP;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_RIGHTBUTTON,
    MOUSE_BUTTONUP);
            }
            break;
        }

        case WM_MBUTTONDOWN:
        {
            l_Window->MouseButton[MOUSE_MIDDLEBUTTON] =
    MOUSE_BUTTONDOWN;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_MIDDLEBUTTON,
    MOUSE_BUTTONDOWN);
            }
            break;
        }

        case WM_MBUTTONUP:
        {
            l_Window->MouseButton[MOUSE_MIDDLEBUTTON] =
    MOUSE_BUTTONUP;

            if (IsValidKeyEvent(l_Window->MouseButtonEvent))
            {
                l_Window->MouseButtonEvent(MOUSE_MIDDLEBUTTON,
    MOUSE_BUTTONUP);
            }
            break;
        }

        case WM_MOUSEWHEEL:
        {
            if ((WordParam % WHEEL_DELTA) > 0)
            {
                if (IsValidMouseWheelEvent(l_Window->MouseWheelEvent))
                {
                    l_Window->MouseWheelEvent(MOUSE_SCROLL_DOWN);
                }
            }

            else
            {
                if (IsValidMouseWheelEvent(l_Window->MouseWheelEvent))
                {
                    l_Window->MouseWheelEvent(MOUSE_SCROLL_UP);
                }

            }
            break;
        }

        default:
        {
            return DefWindowProc(WindowHandle, Message, WordParam, LongParam);
        }
        }
        return 0;
    }

    LRESULT CALLBACK WindowManager::StaticWindowProcedure(HWND WindowHandle, UINT Message, WPARAM WordParam, LPARAM LongParam)
    {
        return WindowManager::GetInstance()->WindowProcedure(WindowHandle, Message, WordParam, LongParam);
    }

    GLboolean WindowManager::Windows_PollForEvents()
    {
        if (GetInstance()->IsInitialized())
        {
            GetMessage(&GetInstance()->Message, 0, 0, 0);
            TranslateMessage(&GetInstance()->Message);
            DispatchMessage(&GetInstance()->Message);
            return FOUNDATION_OKAY;
        }

        else
        {

            PrintErrorMessage(ERROR_NOTINITIALIZED);
            return FOUNDATION_ERROR;
        }
    }

    void WindowManager::CreateTerminal()
    {
        int hConHandle;
        long lStdHandle;
        FILE *fp;

        // allocate a console for this app
        AllocConsole();

        // redirect unbuffered STDOUT to the console
        lStdHandle = (long)GetStdHandle(STD_OUTPUT_HANDLE);
        hConHandle = _open_osfhandle(lStdHandle, _O_TEXT);
        fp = _fdopen(hConHandle, "w");
        *stdout = *fp;

        setvbuf(stdout, nullptr, _IONBF, 0);
    }

    GLboolean WindowManager::Windows_SetMousePositionInScreen(GLuint X, GLuint Y)
    {
        if (GetInstance()->IsInitialized())
        {
            POINT l_MousePoint;
            l_MousePoint.x = X;
            l_MousePoint.y = Y;
            SetCursorPos(l_MousePoint.x, l_MousePoint.y);
        }

        PrintErrorMessage(ERROR_NOTINITIALIZED);
        return FOUNDATION_ERROR;
    }

    GLuint WindowManager::Windows_TranslateKey(WPARAM WordParam, LPARAM LongParam)
    {
        switch (WordParam)
        {
            case VK_ESCAPE:
            {
                return KEY_ESCAPE;
            }

            case VK_F1:
            {
                return KEY_F1;
            }

            case VK_F2:
            {
                return KEY_F2;
            }

            case VK_F3:
            {
                return KEY_F3;
            }

            case VK_F4:
            {
                return KEY_F4;
            }

            case VK_F5:
            {
                return KEY_F5;
            }

            case VK_F6:
            {
                return KEY_F6;
            }

            case VK_F7:
            {
                return KEY_F7;
            }

            case VK_F8:
            {
                return KEY_F8;
            }

            case VK_F9:
            {
                return KEY_F9;
            }

            case VK_F10:
            {
                return KEY_F10;
            }

            case VK_F11:
            {
                return KEY_F11;
            }

            case VK_F12:
            {
                return KEY_F12;
            }

            case VK_BACK:
            {
                return KEY_BACKSPACE;
            }

            case VK_TAB:
            {
                return KEY_TAB;
            }

            case VK_CAPITAL:
            {
                return KEY_CAPSLOCK;
            }

            case VK_RETURN:
            {
                return KEY_ENTER;
            }

            case VK_PRINT:
```

```
        {
            return KEY_PRINTSCREEN;
        }

        case VK_SCROLL:
        {
            return KEY_SCROLLLOCK;
        }

        case VK_PAUSE:
        {
            return KEY_PAUSE;
        }

        case VK_INSERT:
        {
            return KEY_INSERT;
        }

        case VK_HOME:
        {
            return KEY_HOME;
        }

        case VK_DELETE:
        {
            return KEY_DELETE;
        }

        case VK_END:
        {
            return KEY_END;
        }

        case VK_PRIOR:
        {
            return KEY_PAGEUP;
        }

        case VK_NEXT:
        {
            return KEY_PAGEDOWN;
        }

        case VK_DOWN:
        {
            return KEY_ARROW_DOWN;
        }

        case VK_UP:
        {
            return KEY_ARROW_UP;
        }

        case VK_LEFT:
        {
            return KEY_ARROW_LEFT;
        }

        case VK_RIGHT:
        {
            return KEY_ARROW_RIGHT;
        }

        case VK_DIVIDE:
        {
            return KEY_KEYPAD_DIVIDE;
        }

        case VK_MULTIPLY:
        {
            return KEY_KEYPAD_MULTIPLY;
        }

        case VK_SUBTRACT:
        {
            return KEY_KEYPAD_DIVIDE;
        }

        case VK_ADD:
        {
            return KEY_KEYPAD_ADD;
        }

        case VK_DECIMAL:
        {
            return KEY_KEYPAD_PERIOD;
        }

        case VK_NUMPAD0:
        {
            return KEY_KEYPAD_0;
        }

        case VK_NUMPAD1:
        {
            return KEY_KEYPAD_1;
        }

        case VK_NUMPAD2:
        {
            return KEY_KEYPAD_2;
        }

        case VK_NUMPAD3:
        {
            return KEY_KEYPAD_3;
        }

        case VK_NUMPAD4:
        {
            return KEY_KEYPAD_4;
        }

        case VK_NUMPAD5:
        {
            return KEY_KEYPAD_5;
        }

        case VK_NUMPAD6:
        {
            return KEY_KEYPAD_6;
        }

        case VK_NUMPAD7:
        {
            return KEY_KEYPAD_7;
        }

        case VK_NUMPAD8:
        {
            return KEY_KEYPAD_8;
        }

        case VK_NUMPAD9:
        {
            return KEY_KEYPAD_9;
        }

        case VK_LWIN:
        {
            return KEY_LEFTWINDOW;
        }

        case VK_RWIN:
        {
            return KEY_RIGHTWINDOW;
        }

        default:
        {
            return WordParam;
        }
    }
}
#endif
```

Index