TinyWindow

0.49

Generated by Doxygen 1.8.11

Contents

Chapter 1

TinyWindow

a cross platform single header window management API

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 TinyWindow Namespace Reference

Data Structures

- class errorCategory_t
- struct uiVec2
- struct window_t
- class windowManager

Typedefs

- typedef std::function< void(unsigned int, keyState_t)> keyEvent_t
- typedef std::function< void(mouseButton_t, buttonState_t)> mouseButtonEvent_t
- typedef std::function< void(mouseScroll_t)> mouseWheelEvent_t
- typedef std::function< void(void)> destroyedEvent_t
- typedef std::function< void(void)> maximizedEvent_t
- typedef std::function< void(void)> minimizedEvent_t
- typedef std::function< void(bool)> focusEvent_t
- typedef std::function< void(unsigned int, unsigned int)> movedEvent_t
- typedef std::function< void(unsigned int, unsigned int)> resizeEvent_t
- typedef std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> mouseMoveEvent_t

Enumerations

- enum keyState_t { bad, keyState_t::up, keyState_t::down }
- enum key_t {
  bad = -1, bad, first = 256 + 1, F1,
  F2, F3, F4, F5,
  F6, F7, F8, F9,
  F10, F11, F12, capsLock,
  leftShift, rightShift, leftControl, rightControl,
  leftWindow, rightWindow, leftAlt, rightAlt,
  enter, printScreen, scrollLock, numLock,
  pause, insert, home, end,
  pageUp, pageDown, arrowDown, arrowUp,
  arrowLeft, arrowRight, keypadDivide, keypadMultiply,
  keypadSubtract, keypadAdd, keypadEnter, keypadPeriod,
  keypad0, keypad1, keypad2, keypad3,
  keypad4, keypad5, keypad6, keypad7,
  keypad8, keypad9, backspace, tab,
  del, escape, last = escape }

- enum buttonState_t { buttonState_t::up, buttonState_t::down }
- enum mouseButton_t { mouseButton_t::left, mouseButton_t::right, mouseButton_t::middle, mouseButton_t::last }
- enum mouseScroll_t { mouseScroll_t::down, mouseScroll_t::up }
- enum style_t { style_t::bare, style_t::normal, style_t::popup }
- enum state_t { state_t::normal, state_t::maximized, state_t::minimized, state_t::fullscreen }
- enum decorator_t {
  titleBar = 0x01, icon = 0x02, border = 0x04, minimizeButton = 0x08,
  maximizeButton = 0x010, closeButton = 0x20, sizeableBorder = 0x40 }
- enum error_t : int {
  error_t::success, error_t::invalidWindowName, error_t::invalidIconPath, error_t::invalidWindowIndex,
  error_t::invalidWindowState, error_t::invalidResolution, error_t::invalidContext, error_t::existingContext,
  error_t::notInitialized, error_t::alreadyInitialized, error_t::invalidTitlebar, error_t::invalidCallback,
  error_t::windowInvalid, error_t::invalidWindowStyle, error_t::functionNotImplemented, error_t::linuxCannotConnectXServer,
  error_t::linuxInvalidVisualinfo, error_t::linuxCannotCreateWindow, error_t::linuxFunctionNotImplemented,
  error_t::windowsCannotCreateWindows,
  error_t::windowsCannotInitialize, error_t::windowsFunctionNotImplemented }

Functions

- std::error_code make_error_code (error_t eCode)

Variables

- const int defaultWindowWidth = 1280
- const int defaultWindowHeight = 720

6.1.1 Typedef Documentation

6.1.1.1 typedef std::function<void(void)> TinyWindow::destroyedEvent_t

Definition at line 220 of file TinyWindow.h.

6.1.1.2 typedef std::function<void(bool)> TinyWindow::focusEvent_t

Definition at line 223 of file TinyWindow.h.

6.1.1.3 typedef std::function<void(unsigned int, keyState_t)> TinyWindow::keyEvent_t

Definition at line 217 of file TinyWindow.h.

6.1.1.4 typedef std::function<void(void)> TinyWindow::maximizedEvent_t

Definition at line 221 of file TinyWindow.h.

6.1.1.5 typedef std::function<void(void)> TinyWindow::minimizedEvent_t

Definition at line 222 of file TinyWindow.h.

6.1.1.6 typedef std::function<void(mouseButton_t, buttonState_t)> TinyWindow::mouseButtonEvent_t

Definition at line 218 of file TinyWindow.h.

6.1.1.7 typedef std::function<void(unsigned int, unsigned int, unsigned int, unsigned int)> TinyWindow::mouseMoveEvent_t

Definition at line 226 of file TinyWindow.h.

6.1.1.8 typedef std::function<void(mouseScroll_t)> TinyWindow::mouseWheelEvent_t

Definition at line 219 of file TinyWindow.h.

6.1.1.9 typedef std::function<void(unsigned int, unsigned int)> TinyWindow::movedEvent_t

Definition at line 224 of file TinyWindow.h.

6.1.1.10 typedef std::function<void(unsigned int, unsigned int)> TinyWindow::resizeEvent_t

Definition at line 225 of file TinyWindow.h.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum TinyWindow::buttonState_t [strong]

Enumerator

up The mouse button is currently up

down The mouse button is currently down

Definition at line 145 of file TinyWindow.h.

```
00146     {
00147         up,
00148         down
00149     };
```

6.1.2.2 enum TinyWindow::decorator_t

Enumerator

titleBar The title bar decoration of the window

icon The icon decoration of the window

border The border decoration of the window

minimizeButton The minimize button decoration of the window

maximizeButton The maximize button decoration pf the window

closeButton The close button decoration of the window

sizeableBorder The sizable border decoration of the window

Definition at line 180 of file TinyWindow.h.

```
00181     {
00182         titleBar = 0x01,
00183         icon = 0x02,
00184         border = 0x04,
00185         minimizeButton = 0x08,
00186         maximizeButton = 0x010,
00187         closeButton = 0x20,
00188         sizeableBorder = 0x40,
00189     };
```

6.1.2.3 enum TinyWindow::error_t : int [strong]

Enumerator

success If a function call was successful

invalidWindowName If an invalid window name was given

invalidIconPath If an invalid icon path was given

invalidWindowIndex If an invalid window index was given

invalidWindowState If an invalid window state was given

invalidResolution If an invalid window resolution was given

invalidContext If the OpenGL context for the window is invalid

existingContext If the window already has an OpenGL context

notInitialized If the window is being used without being initialized

alreadyInitialized If the window was already initialized

invalidTitlebar If the Title-bar text given was invalid

invalidCallback If the given event callback was invalid

windowInvalid If the window given was invalid

invalidWindowStyle If the window style gives is invalid

functionNotImplemented If the function has not yet been implemented in the current version of the API

linuxCannotConnectXServer Linux: if cannot connect to an X11 server

linuxInvalidVisualinfo Linux: if visual information given was invalid

linuxCannotCreateWindow Linux: when X11 fails to create a new window

linuxFunctionNotImplemented Linux: when the function has not yet been implemented on the Linux in the current version of the API

windowsCannotCreateWindows Windows: when Win32 cannot create a window

windowsCannotInitialize Windows: when Win32 cannot initialize

windowsFunctionNotImplemented Windows: when a function has yet to be implemented on the Windows platform in the current version of the API

Definition at line 191 of file TinyWindow.h.

```
00191                       : int
00192     {
00193         success,
00194         invalidWindowName,
00195         invalidIconPath,
00196         invalidWindowIndex,
00197         invalidWindowState,
00198         invalidResolution,
00199         invalidContext,
00200         existingContext,
00201         notInitialized,
00202         alreadyInitialized,
00203         invalidTitlebar,
00204         invalidCallback,
00205         windowInvalid,
00206         invalidWindowStyle,
00207         functionNotImplemented,
00208         linuxCannotConnectXServer,
00209         linuxInvalidVisualinfo,
00210         linuxCannotCreateWindow,
00211         linuxFunctionNotImplemented,
00212         windowsCannotCreateWindows,
00213         windowsCannotInitialize,
00214         windowsFunctionNotImplemented,
00215     };
```

6.1.2.4 enum TinyWindow::key_t

Enumerator

bad The key pressed is considered invalid

bad If get key state fails (could not name it ERROR)

first The first key that is not a char

F1 The F1 key

F2 The F2 key

F3 The F3 key

F4 The F4 key

F5 The F5 key

F6 The F6 key

F7 The F7 key

F8 The F8 key

F9 The F9 key

F10 The F10 key

F11 The F11 key

F12 The F12 key

capsLock The CapsLock key

leftShift The left Shift key

rightShift The right Shift key

leftControl The left Control key

rightControl The right Control key

leftWindow The left Window key

rightWindow The right Window key

leftAlt The left Alternate key

rightAlt The right Alternate key

enter The Enter/Return key

printScreen The PrintScreen key

scrollLock The ScrollLock key

numLock The NumLock key

pause The pause/break key

insert The insert key

home The Home key

end The End key

pageUp The PageUp key

pageDown The PageDown key

arrowDown The ArrowDown key

arrowUp The ArrowUp key

arrowLeft The ArrowLeft key

arrowRight The ArrowRight key

keypadDivide The KeyPad Divide key

keypadMultiply The Keypad Multiply key

keypadSubtract The Keypad Subtract key

keypadAdd The Keypad Add key

keypadEnter The Keypad Enter key

keypadPeriod The Keypad Period/Decimal key

keypad0 The Keypad 0 key

keypad1 The Keypad 1 key

keypad2 The Keypad 2 key

keypad3 The Keypad 3 key

keypad4 The Keypad 4 key

keypad5 The Keypad 5 key

keypad6 The Keypad 6 key

keypad7 The Keypad 7 key

keypad8 The keypad 8 key

keypad9 The Keypad 9 key

backspace The Backspace key

tab The Tab key

del The Delete key

escape The Escape key

last The last key to be supported

Definition at line 83 of file TinyWindow.h.

```
00084     {
00085         bad = -1,
00086         first = 256 + 1,
00087         F1,
00088         F2,
00089         F3,
00090         F4,
00091         F5,
00092         F6,
```

```
00093         F7,
00094         F8,
00095         F9,
00096         F10,
00097         F11,
00098         F12,
00099         capsLock,
00100         leftShift,
00101         rightShift,
00102         leftControl,
00103         rightControl,
00104         leftWindow,
00105         rightWindow,
00106         leftAlt,
00107         rightAlt,
00108         enter,
00109         printScreen,
00110         scrollLock,
00111         numLock,
00112         pause,
00113         insert,
00114         home,
00115         end,
00116         pageUp,
00117         pageDown,
00118         arrowDown,
00119         arrowUp,
00120         arrowLeft,
00121         arrowRight,
00122         keypadDivide,
00123         keypadMultiply,
00124         keypadSubtract,
00125         keypadAdd,
00126         keypadEnter,
00127         keypadPeriod,
00128         keypad0,
00129         keypad1,
00130         keypad2,
00131         keypad3,
00132         keypad4,
00133         keypad5,
00134         keypad6,
00135         keypad7,
00136         keypad8,
00137         keypad9,
00138         backspace,
00139         tab,
00140         del,
00141         escape,
00142         last = escape,
00143     };
```

6.1.2.5 enum TinyWindow::keyState_t [strong]

Enumerator

bad If get key state fails (could not name it ERROR)

up The key is currently up

down The key is currently down

Definition at line 76 of file TinyWindow.h.

```
00077     {
00078         bad,
00079         up,
00080         down,
00081     };
```

6.1.2.6 enum TinyWindow::mouseButton_t [strong]

Enumerator

left The left mouse button

right The right mouse button

middle The middle mouse button / ScrollWheel

last The last mouse button to be supported

Definition at line 151 of file TinyWindow.h.

```
00152     {
00153         left,
00154         right,
00155         middle,
00156         last,
00157     };
```

6.1.2.7 enum TinyWindow::mouseScroll_t [strong]

Enumerator

down The mouse wheel up

up The mouse wheel down

Definition at line 159 of file TinyWindow.h.

```
00160     {
00161         down,
00162         up
00163     };
```

6.1.2.8 enum TinyWindow::state_t [strong]

Enumerator

normal The window is in its default state

maximized The window is currently maximized

minimized The window is currently minimized

fullscreen The window is currently full screen

Definition at line 172 of file TinyWindow.h.

```
00173     {
00174         normal,
00175         maximized,
00176         minimized,
00177         fullscreen,
00178     };
```

6.1.2.9 enum TinyWindow::style_t [strong]

Enumerator

bare The window has no decorators but the window border and title bar

normal The default window style for the respective platform

popup The window has no decorators

Definition at line 165 of file TinyWindow.h.

```
00166     {
00167         bare,
00168         normal,
00169         popup,
00170     };
```

6.1.3 Function Documentation

6.1.3.1 std::error_code TinyWindow::make_error_code (error_t *eCode*)

Definition at line 366 of file TinyWindow.h.

References TinyWindow::errorCategory_t::get().

```
00367     {
00368         return std::error_code(static_cast<int>(eCode), errorCategory_t::get());
00369     }
```

6.1.4 Variable Documentation

6.1.4.1 const int TinyWindow::defaultWindowHeight = 720

Definition at line 42 of file TinyWindow.h.

6.1.4.2 const int TinyWindow::defaultWindowWidth = 1280

Definition at line 41 of file TinyWindow.h.

Chapter 7

Data Structure Documentation

7.1 TinyWindow::errorCategory_t Class Reference

`#include <TinyWindow.h>`

Inherits std::error_category.

Public Member Functions

- const char * name () const override throw ()
- virtual std::string message (int errorValue) const override
- errorCategory_t ()

Static Public Member Functions

- static const errorCategory_t & get ()

7.1.1 Detailed Description

Definition at line 228 of file TinyWindow.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 TinyWindow::errorCategory_t::errorCategory_t ( ) [inline]

Definition at line 356 of file TinyWindow.h.

```
00356 {};
```

7.1.3 Member Function Documentation

7.1.3.1 static const errorCategory_t& TinyWindow::errorCategory_t::get ( ) [inline], [static]

Definition at line 358 of file TinyWindow.h.

Referenced by TinyWindow::make_error_code().

```
00359         {
00360             const static errorCategory_t category;
00361             return category;
00362
00363         }
```

7.1.3.2 virtual std::string TinyWindow::errorCategory_t::message (int *errorValue*) const [inline], [override], [virtual]

return the error message associated with the given error number

Definition at line 240 of file TinyWindow.h.

References TinyWindow::alreadyInitialized, TinyWindow::existingContext, TinyWindow::functionNotImplemented, TinyWindow::invalidCallback, TinyWindow::invalidContext, TinyWindow::invalidIconPath, TinyWindow::invalidResolution, TinyWindow::invalidTitlebar, TinyWindow::invalidWindowIndex, TinyWindow::invalidWindowName, TinyWindow::invalidWindowState, TinyWindow::invalidWindowStyle, TinyWindow::linuxCannotConnectXServer, TinyWindow::linuxCannotCreateWindow, TinyWindow::linuxFunctionNotImplemented, TinyWindow::linuxInvalidVisualinfo, TinyWindow::notInitialized, TinyWindow::success, TinyWindow::windowInvalid, TinyWindow::windowsCannotCreateWindows, and TinyWindow::windowsFunctionNotImplemented.

```
00241         {
00242             error_t err = (error_t)errorValue;
00243             switch (err)
00244             {
00245             case error_t::invalidWindowName:
00246             {
00247                 return "Error: invalid window name \n";
00248             }
00249
00250             case error_t::invalidIconPath:
00251             {
00252                 return "Error: invalid icon path \n";
00253             }
00254
00255             case error_t::invalidWindowIndex:
00256             {
00257                 return "Error: invalid window index \n";
00258             }
00259
00260             case error_t::invalidWindowState:
00261             {
00262                 return "Error: invalid window state \n";
00263             }
00264
00265             case error_t::invalidResolution:
00266             {
00267                 return "Error: invalid resolution \n";
00268             }
00269
00270             case error_t::invalidContext:
00271             {
00272                 return "Error: Failed to create OpenGL context \n";
00273             }
00274
00275             case error_t::existingContext:
00276             {
00277                 return "Error: context already created \n";
00278             }
00279
```

```
00280             case error_t::notInitialized:
00281             {
00282                 return "Error: Window manager not initialized \n";
00283             }
00284 
00285             case error_t::alreadyInitialized:
00286             {
00287                 return "Error: window has already been initialized \n";
00288             }
00289 
00290             case error_t::invalidTitlebar:
00291             {
00292                 return "Error: invalid title bar name (cannot be null or nullptr) \n";
00293             }
00294 
00295             case error_t::invalidCallback:
00296             {
00297                 return "Error: invalid event callback given \n";
00298             }
00299 
00300             case error_t::windowInvalid:
00301             {
00302                 return "Error: window was not found \n";
00303             }
00304 
00305             case error_t::invalidWindowStyle:
00306             {
00307                 return "Error: invalid window style given \n";
00308             }
00309 
00310             case error_t::functionNotImplemented:
00311             {
00312                 return "Error: I'm sorry but this function has not been implemented yet :(\n";
00313             }
00314 
00315             case error_t::linuxCannotConnectXServer:
00316             {
00317                 return "Error: cannot connect to X server \n";
00318             }
00319 
00320             case error_t::linuxInvalidVisualinfo:
00321             {
00322                 return "Error: Invalid visual information given \n";
00323             }
00324 
00325             case error_t::linuxCannotCreateWindow:
00326             {
00327                 return "Error: failed to create window \n";
00328             }
00329 
00330             case error_t::linuxFunctionNotImplemented:
00331             {
00332                 return "Error: function not implemented on Linux platform yet. sorry :(\n";
00333             }
00334 
00335             case error_t::windowsCannotCreateWindows:
00336             {
00337                 return "Error: failed to create window \n";
00338             }
00339 
00340             case error_t::windowsFunctionNotImplemented:
00341             {
00342                 return "Error: function not implemented on Windows platform yet. sorry ;(\n";
00343             }
00344 
00345             case error_t::success:
00346             {
00347                 return "function call was successful \n";
00348             }
00349 
00350             default:
00351             {
00352                 return "Error: unspecified Error \n";
00353             }
00354             }
00355         }
```

7.1.3.3 const char* TinyWindow::errorCategory_t::name ( ) const throw) [inline], [override]

Definition at line 232 of file TinyWindow.h.

```
00233         {
00234             return "tinyWindow";
00235         }
```

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

- C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h

7.2 std::is_error_code_enum< TinyWindow::error_t > Struct Template Reference

`#include <TinyWindow.h>`

Inherits true_type.

7.2.1 Detailed Description

template<>
struct std::is_error_code_enum< TinyWindow::error_t >

Definition at line 374 of file TinyWindow.h.

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

- C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h

7.3 TinyWindow::uiVec2 Struct Reference

`#include <TinyWindow.h>`

Public Member Functions

- uiVec2 ()
- uiVec2 (unsigned int x, unsigned int y)

Static Public Member Functions

- static uiVec2 Zero ()

Data Fields

- union {
  unsigned int x
  unsigned int width
  };

- union {
  unsigned int y
  unsigned int height
  };

7.3.1 Detailed Description

Definition at line 44 of file TinyWindow.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 TinyWindow::uiVec2::uiVec2 ( ) [inline]

Definition at line 46 of file TinyWindow.h.

```
00047         {
00048             this->x = 0;
00049             this->y = 0;
00050         }
```

7.3.2.2 TinyWindow::uiVec2::uiVec2 (unsigned int *x*, unsigned int *y*) [inline]

Definition at line 52 of file TinyWindow.h.

Referenced by Zero().

```
00053         {
00054             this->x = x;
00055             this->y = y;
00056         }
```

7.3.3 Member Function Documentation

7.3.3.1 static uiVec2 TinyWindow::uiVec2::Zero ( ) [inline], [static]

Definition at line 70 of file TinyWindow.h.

References uiVec2().

```
00071         {
00072             return uiVec2(0, 0);
00073         }
```

7.3.4 Field Documentation

7.3.4.1 union { ... }

7.3.4.2 union { ... }

7.3.4.3 unsigned int TinyWindow::uiVec2::height

Definition at line 67 of file TinyWindow.h.

7.3.4.4 unsigned int TinyWindow::uiVec2::width

Definition at line 61 of file TinyWindow.h.

7.3.4.5 unsigned int TinyWindow::uiVec2::x

Definition at line 60 of file TinyWindow.h.

7.3.4.6 unsigned int TinyWindow::uiVec2::y

Definition at line 66 of file TinyWindow.h.

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

- C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h

7.4 TinyWindow::window_t Struct Reference

`#include <TinyWindow.h>`

Public Member Functions

- window_t (const char *name=nullptr, unsigned int iD=0, unsigned int colorBits=0, unsigned int depthBits=0, unsigned int stencilBits=0, bool shouldClose=false, state_t currentState=state_t::normal, keyEvent_t keyEvent=nullptr, mouseButtonEvent_t mouseButtonEvent=nullptr, mouseWheelEvent_t mouseWheelEvent=nullptr, destroyedEvent_t destroyedEvent=nullptr, maximizedEvent_t maximizedEvent=nullptr, minimizedEvent_t minimizedEvent=nullptr, focusEvent_t focusEvent=nullptr, movedEvent_t movedEvent=nullptr, resizeEvent_t resizeEvent=nullptr, mouseMoveEvent_t mouseMoveEvent=nullptr)

Data Fields

- const char * name
- unsigned int iD
- int colorBits
- int depthBits
- int stencilBits
- keyState_t keys [last]
- buttonState_t mouseButton [(unsigned int) mouseButton_t::last]
- TinyWindow::uiVec2 resolution
- TinyWindow::uiVec2 position
- TinyWindow::uiVec2 mousePosition
- bool shouldClose
- bool inFocus
- bool initialized
- bool contextCreated
- bool isCurrentContext
- state_t currentState

- unsigned int currentWindowStyle
- keyEvent_t keyEvent
- mouseButtonEvent_t mouseButtonEvent
- mouseWheelEvent_t mouseWheelEvent
- destroyedEvent_t destroyedEvent
- maximizedEvent_t maximizedEvent
- minimizedEvent_t minimizedEvent
- focusEvent_t focusEvent
- movedEvent_t movedEvent
- resizeEvent_t resizeEvent
- mouseMoveEvent_t mouseMoveEvent
- Window windowHandle
- GLXContext context
- XVisualInfo * visualInfo
- int * attributes
- XSetWindowAttributes setAttributes
- unsigned int decorators

7.4.1 Detailed Description

Definition at line 380 of file TinyWindow.h.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 TinyWindow::window_t::window_t (const char * *name* = `nullptr`, unsigned int *iD* = `0`, unsigned int *colorBits* = `0`, unsigned int *depthBits* = `0`, unsigned int *stencilBits* = `0`, bool *shouldClose* = `false`, state_t *currentState* = state_t::normal, keyEvent_t *keyEvent* = `nullptr`, mouseButtonEvent_t *mouseButtonEvent* = `nullptr`, mouseWheelEvent_t *mouseWheelEvent* = `nullptr`, destroyedEvent_t *destroyedEvent* = `nullptr`, maximizedEvent_t *maximizedEvent* = `nullptr`, minimizedEvent_t *minimizedEvent* = `nullptr`, focusEvent_t *focusEvent* = `nullptr`, movedEvent_t *movedEvent* = `nullptr`, resizeEvent_t *resizeEvent* = `nullptr`, mouseMoveEvent_t *mouseMoveEvent* = `nullptr`) [inline]

Definition at line 435 of file TinyWindow.h.

References colorBits, contextCreated, currentState, currentWindowStyle, depthBits, iD, initialized, name, TinyWindow::normal, shouldClose, and stencilBits.

```
00448     {
00449         this->name = name;
00450         this->iD = iD;
00451         this->colorBits = colorBits;
00452         this->depthBits = depthBits;
00453         this->stencilBits = stencilBits;
00454         this->shouldClose = shouldClose;
00455         this->currentState = currentState;
00456 
00457         this->keyEvent = keyEvent;
00458         this->mouseButtonEvent = mouseButtonEvent;
00459         this->mouseWheelEvent = mouseWheelEvent;
00460         this->destroyedEvent = destroyedEvent;
00461         this->maximizedEvent = maximizedEvent;
00462         this->minimizedEvent = minimizedEvent;
00463         this->focusEvent = focusEvent;
00464         this->movedEvent = movedEvent;
00465         this->resizeEvent = resizeEvent;
00466         this->mouseMoveEvent = mouseMoveEvent;
00467 
00468         initialized = false;
00469         contextCreated = false;
00470         currentWindowStyle = (unsigned int)style_t::normal;
00471 
00472 #if defined(__linux__)
00473         context = 0;
00474 #endif
00475     }
```

7.4.3 Field Documentation

7.4.3.1 int* TinyWindow::window_t::attributes

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

Definition at line 429 of file TinyWindow.h.

7.4.3.2 int TinyWindow::window_t::colorBits

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

Definition at line 385 of file TinyWindow.h.

Referenced by window_t().

7.4.3.3 GLXContext TinyWindow::window_t::context

The handle to the GLX rendering context

Definition at line 427 of file TinyWindow.h.

7.4.3.4 bool TinyWindow::window_t::contextCreated

Whether the OpenGL context has been successfully created

Definition at line 397 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::Platform_InitializeGL(), and window_t().

7.4.3.5 state_t TinyWindow::window_t::currentState

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

Definition at line 400 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::MaximizeWindow(), TinyWindow::windowManager::Minimize↩Window(), TinyWindow::windowManager::SetFullScreen(), and window_t().

7.4.3.6 unsigned int TinyWindow::window_t::currentWindowStyle

The current style of the window

Definition at line 401 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::DisableWindowDecorators(), TinyWindow::windowManager::↩EnableWindowDecorators(), and window_t().

7.4.3.7 unsigned int TinyWindow::window_t::decorators

Enabled window decorators

Definition at line 431 of file TinyWindow.h.

Referenced by main().

7.4.3.8 int TinyWindow::window_t::depthBits

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

Definition at line 386 of file TinyWindow.h.

Referenced by window_t().

7.4.3.9 destroyedEvent_t TinyWindow::window_t::destroyedEvent

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

Definition at line 406 of file TinyWindow.h.

7.4.3.10 focusEvent_t TinyWindow::window_t::focusEvent

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

Definition at line 409 of file TinyWindow.h.

7.4.3.11 unsigned int TinyWindow::window_t::iD

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

Definition at line 384 of file TinyWindow.h.

Referenced by window_t().

7.4.3.12 bool TinyWindow::window_t::inFocus

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

Definition at line 394 of file TinyWindow.h.

7.4.3.13 bool TinyWindow::window_t::initialized

Whether the window has been successfully initialized

Definition at line 396 of file TinyWindow.h.

Referenced by window_t().

7.4.3.14 bool TinyWindow::window_t::isCurrentContext

Whether the window is the current window being drawn to

Definition at line 398 of file TinyWindow.h.

7.4.3.15 keyEvent_t TinyWindow::window_t::keyEvent

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

Definition at line 403 of file TinyWindow.h.

7.4.3.16 keyState_t TinyWindow::window_t::keys[last]

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

Definition at line 388 of file TinyWindow.h.

7.4.3.17 maximizedEvent_t TinyWindow::window_t::maximizedEvent

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

Definition at line 407 of file TinyWindow.h.

7.4.3.18 minimizedEvent_t TinyWindow::window_t::minimizedEvent

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

Definition at line 408 of file TinyWindow.h.

7.4.3.19 buttonState_t TinyWindow::window_t::mouseButton[(unsigned int) mouseButton_t::last]

Record of mouse buttons that are either presses or released

Definition at line 389 of file TinyWindow.h.

7.4.3.20 mouseButtonEvent_t TinyWindow::window_t::mouseButtonEvent

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

Definition at line 404 of file TinyWindow.h.

7.4.3.21 mouseMoveEvent_t TinyWindow::window_t::mouseMoveEvent

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

Definition at line 412 of file TinyWindow.h.

7.4.3.22 TinyWindow::uiVec2 TinyWindow::window_t::mousePosition

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

Definition at line 392 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::SetMousePositionInWindow().

7.4.3.23 mouseWheelEvent_t TinyWindow::window_t::mouseWheelEvent

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

Definition at line 405 of file TinyWindow.h.

7.4.3.24 movedEvent_t TinyWindow::window_t::movedEvent

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

Definition at line 410 of file TinyWindow.h.

7.4.3.25 const char* TinyWindow::window_t::name

Name of the window

Definition at line 383 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::ShutdownWindow(), and window_t().

7.4.3.26 TinyWindow::uiVec2 TinyWindow::window_t::position

Position of the Window relative to the screen co-ordinates

Definition at line 391 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::Platform_SetWindowResolution(), and TinyWindow::windowManager::SetWindowPosition().

7.4.3.27 resizeEvent_t TinyWindow::window_t::resizeEvent

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

Definition at line 411 of file TinyWindow.h.

7.4.3.28 TinyWindow::uiVec2 TinyWindow::window_t::resolution

Resolution/Size of the window stored in an array

Definition at line 390 of file TinyWindow.h.

Referenced by TinyWindow::windowManager::Platform_SetWindowPosition(), TinyWindow::windowManager::↩ Platform_SetWindowResolution(), and TinyWindow::windowManager::SetWindowResolution().

7.4.3.29 XSetWindowAttributes TinyWindow::window_t::setAttributes

The attributes to be set for the window

Definition at line 430 of file TinyWindow.h.

7.4.3.30 bool TinyWindow::window_t::shouldClose

Whether the Window should be closing

Definition at line 393 of file TinyWindow.h.

Referenced by main(), and window_t().

7.4.3.31 int TinyWindow::window_t::stencilBits

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

Definition at line 387 of file TinyWindow.h.

Referenced by window_t().

7.4.3.32 XVisualInfo* TinyWindow::window_t::visualInfo

The handle to the Visual Information. similar purpose to PixelformatDesriptor

Definition at line 428 of file TinyWindow.h.

7.4.3.33 Window TinyWindow::window_t::windowHandle

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

Definition at line 426 of file TinyWindow.h.

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

- C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h

7.5 TinyWindow::windowManager Class Reference

`#include <TinyWindow.h>`

Public Member Functions

- windowManager (void)
- ~windowManager (void)
- void ShutDown (void)
- window_t * AddWindow (const char *windowName, unsigned int width=defaultWindowWidth, unsigned int height=defaultWindowHeight, int colourBits=8, int depthBits=8, int stencilBits=8)
- int GetNumWindows (void)
- TinyWindow::uiVec2 GetMousePositionInScreen (void)
- void SetMousePositionInScreen (TinyWindow::uiVec2 mousePosition)
- void SetMousePositionInScreen (unsigned int x, unsigned int y)
- TinyWindow::uiVec2 GetScreenResolution (void)
- std::error_code SetWindowResolution (window_t *window, TinyWindow::uiVec2 resolution)
- std::error_code SetWindowResolution (window_t *window, unsigned int width, unsigned int height)
- std::error_code SetWindowPosition (window_t *window, TinyWindow::uiVec2 windowPosition)
- std::error_code SetWindowPosition (window_t *window, unsigned int x, unsigned int y)
- std::error_code SetMousePositionInWindow (window_t *window, TinyWindow::uiVec2 mousePosition)
- std::error_code SetMousePositionInWindow (window_t *window, unsigned int x, unsigned int y)
- std::error_code SwapWindowBuffers (window_t *window)
- std::error_code MakeWindowCurrentContext (window_t *window)
- std::error_code SetFullScreen (window_t *window, bool newState)
- std::error_code MinimizeWindow (window_t *window, bool newState)
- std::error_code MaximizeWindow (window_t *window, bool newState)
- std::error_code SetWindowTitleBar (window_t *window, const char *newTitle)
- std::error_code SetWindowIcon (void)
- std::error_code FocusWindow (window_t *window, bool newState)
- std::error_code RestoreWindow (window_t *window)
- void PollForEvents (void)
- void WaitForEvents (void)
- std::error_code RemoveWindow (window_t *window)
- std::error_code SetWindowStyle (window_t *window, style_t windowStyle)
- std::error_code EnableWindowDecorators (window_t *window, unsigned int decorators)
- std::error_code DisableWindowDecorators (window_t *window, unsigned int decorators)

Private Member Functions

- void Platform_InitializeWindow (window_t *window)
- std::error_code Platform_InitializeGL (window_t *window)
- void Platform_SetMousePositionInScreen ()
- void Platform_GetScreenResolution (uiVec2 resolution)
- void Platform_SetWindowResolution (window_t *window)
- void Platform_SetWindowPosition (window_t *window, unsigned int x, unsigned int y)
- void Platform_SetMousePositionInWindow (window_t *window, unsigned int x, unsigned int y)
- void ShutdownWindow (window_t *window)

Private Attributes

- std::vector< std::unique_ptr< window_t > > windowList
- TinyWindow::uiVec2 screenResolution
- TinyWindow::uiVec2 screenMousePosition

7.5.1 Detailed Description

Definition at line 478 of file TinyWindow.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 TinyWindow::windowManager::windowManager (void) [inline]

Definition at line 484 of file TinyWindow.h.

References screenResolution.

Referenced by main().

```
00485     {
00486 #if defined(TW_WINDOWS)
00487         CreateTerminal(); //feel free to comment this out
00488         RECT desktop;
00489 
00490         HWND desktopHandle = GetDesktopWindow();
00491 
00492         if (desktopHandle)
00493         {
00494             GetWindowRect(desktopHandle, &desktop);
00495 
00496             screenResolution.x = desktop.right;
00497             screenResolution.y = desktop.bottom;
00498             return;
00499         }
00500 #elif defined(TW_LINUX)
00501         currentDisplay = XOpenDisplay(0);
00502 
00503         if (!currentDisplay)
00504         {
00505             return;
00506         }
00507 
00508         screenResolution.x = WidthOfScreen(
00509             XScreenOfDisplay(currentDisplay,
00510                 DefaultScreen(currentDisplay)));
00511 
00512         screenResolution.y = HeightOfScreen(
00513             XScreenOfDisplay(currentDisplay,
00514                 DefaultScreen(currentDisplay)));
00515 #endif
00516     }
```

7.5.2.2 TinyWindow::windowManager::~windowManager (void) [inline]

Shutdown and delete all windows in the manager

Definition at line 521 of file TinyWindow.h.

References ShutDown().

```
00522     {
00523         ShutDown();
00524     }
```

7.5.3 Member Function Documentation

7.5.3.1 window_t* TinyWindow::windowManager::AddWindow (const char * *windowName*, unsigned int *width* = defaultWindowWidth, unsigned int *height* = defaultWindowHeight, int *colourBits* = 8, int *depthBits* = 8, int *stencilBits* = 8) [inline]

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

Definition at line 540 of file TinyWindow.h.

Referenced by main().

```
00541     {
00542         if (windowName != nullptr)
00543         {
00544             std::unique_ptr<window_t> newWindow(new window_t);
00545             newWindow->name = windowName;
00546             newWindow->resolution.width = width;
00547             newWindow->resolution.height = height;
00548             newWindow->colorBits = colourBits;
00549             newWindow->depthBits = depthBits;
00550             newWindow->stencilBits = stencilBits;
00551             newWindow->iD = GetNumWindows();
00552 
00553             windowList.push_back(std::move(newWindow));
00554             Platform_InitializeWindow(windowList.back().get());
00555 
00556             return windowList.back().get();
00557         }
00558         //PrintErrorMessage(std::error_code(invalidWindowName));
00559         return nullptr;
00560     }
```

7.5.3.2 std::error_code TinyWindow::windowManager::DisableWindowDecorators (window_t * *window*, unsigned int *decorators*) [inline]

Disable windows decorators by name

Definition at line 1181 of file TinyWindow.h.

References TinyWindow::border, TinyWindow::closeButton, TinyWindow::window_t::currentWindowStyle, TinyWindow::icon, TinyWindow::maximizeButton, TinyWindow::minimizeButton, TinyWindow::sizeableBorder, and TinyWindow::titleBar.

```
01182     {
01183         if (window != nullptr)
01184         {
01185 #if defined(TW_WINDOWS)
01186             if (decorators & border)
01187             {
01188                 window->currentWindowStyle &= ~WS_BORDER;
01189             }
01190 
01191             if (decorators & titleBar)
01192             {
01193                 window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
01194             }
01195 
01196             if (decorators & icon)
01197             {
01198                 window->currentWindowStyle &= ~WS_ICONIC;
01199             }
01200 
01201             if (decorators & closeButton)
01202             {
01203                 window->currentWindowStyle &= ~WS_SYSMENU;
01204             }
```

```
01205
01206             if (decorators & minimizeButton)
01207             {
01208                 window->currentWindowStyle &= ~WS_MINIMIZEBOX;
01209             }
01210
01211             if (decorators & maximizeButton)
01212             {
01213                 window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
01214             }
01215
01216             if (decorators & sizeableBorder)
01217             {
01218                 window->currentWindowStyle &= ~WS_SIZEBOX;
01219             }
01220
01221             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
01222                 window->currentWindowStyle | WS_VISIBLE);
01223 #elif defined(TW_LINUX)
01224             if (decorators & closeButton)
01225             {
01226                 //I hate doing this but it is necessary to keep functionality going.
01227                 bool minimizeEnabled = false;
01228                 bool maximizeEnabled = false;
01229
01230                 if (decorators & maximizeButton)
01231                 {
01232                     maximizeEnabled = true;
01233                 }
01234
01235                 if (decorators & minimizeButton)
01236                 {
01237                     minimizeEnabled = true;
01238                 }
01239
01240                 window->currentWindowStyle &= ~linuxClose;
01241
01242                 if (maximizeEnabled)
01243                 {
01244                     window->currentWindowStyle |= linuxMaximize;
01245                 }
01246
01247                 if (minimizeEnabled)
01248                 {
01249                     window->currentWindowStyle |= linuxMinimize;
01250                 }
01251
01252                 window->decorators = 1;
01253             }
01254
01255             if (decorators & minimizeButton)
01256             {
01257                 window->currentWindowStyle &= ~linuxMinimize;
01258                 window->decorators = 1;
01259             }
01260
01261             if (decorators & maximizeButton)
01262             {
01263                 bool minimizeEnabled = false;
01264
01265                 if (decorators & minimizeButton)
01266                 {
01267                     minimizeEnabled = true;
01268                 }
01269
01270                 window->currentWindowStyle &= ~linuxMaximize;
01271
01272                 if (minimizeEnabled)
01273                 {
01274                     window->currentWindowStyle |= linuxMinimize;
01275                 }
01276
01277                 window->decorators = 1;
01278             }
01279
01280             if (decorators & icon)
01281             {
01282                 //Linux (at least cinnamon) does not have icons in the window. only in the taskb ar icon
01283             }
01284
01285             //just need to set it to 1 to enable all decorators that include title bar
01286             if (decorators & titleBar)
01287             {
01288                 window->decorators = linuxBorder;
01289             }
01290
01291             if (decorators & border)
```

```
01292             {
01293                 window->decorators = 0;
01294             }
01295
01296             if (decorators & sizeableBorder)
01297             {
01298                 window->decorators = 0;
01299             }
01300
01301             long hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0, 0 };
01302
01303             XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
01304                 PropModeReplace, (unsigned char*)hints, 5);
01305
01306             XMapWindow(currentDisplay, window->windowHandle);
01307 #endif
01308             return TinyWindow::error_t::success;
01309         }
01310         return TinyWindow::error_t::windowInvalid;
01311     }
```

7.5.3.3 std::error_code TinyWindow::windowManager::EnableWindowDecorators (window_t * *window*, unsigned int *decorators*) [inline]

Enable window decorators by name

Definition at line 1082 of file TinyWindow.h.

References TinyWindow::border, TinyWindow::closeButton, TinyWindow::window_t::currentWindowStyle, TinyWindow::icon, TinyWindow::maximizeButton, TinyWindow::minimizeButton, TinyWindow::sizeableBorder, and TinyWindow::titleBar.

Referenced by SetWindowStyle().

```
01083     {
01084         if (window != nullptr)
01085         {
01086 #if defined(TW_WINDOWS)
01087             window->currentWindowStyle = WS_VISIBLE | WS_CLIPSIBLINGS;
01088
01089             if (decorators & border)
01090             {
01091                 window->currentWindowStyle |= WS_BORDER;
01092             }
01093
01094             if (decorators & titleBar)
01095             {
01096                 window->currentWindowStyle |= WS_CAPTION;
01097             }
01098
01099             if (decorators & icon)
01100             {
01101                 window->currentWindowStyle |= WS_ICONIC;
01102             }
01103
01104             if (decorators & closeButton)
01105             {
01106                 window->currentWindowStyle |= WS_SYSMENU;
01107             }
01108
01109             if (decorators & minimizeButton)
01110             {
01111                 window->currentWindowStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
01112             }
01113
01114             if (decorators & maximizeButton)
01115             {
01116                 window->currentWindowStyle |= WS_MAXIMIZEBOX | WS_SYSMENU;
01117             }
01118
01119             if (decorators & sizeableBorder)
01120             {
01121                 window->currentWindowStyle |= WS_SIZEBOX;
01122             }
01123
```

```
01124             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
01125                 window->currentWindowStyle);
01126 #elif defined(TW_LINUX)
01127             if (decorators & closeButton)
01128             {
01129                 window->currentWindowStyle |= linuxClose;
01130                 window->decorators = 1;
01131             }
01132
01133             if (decorators & minimizeButton)
01134             {
01135                 window->currentWindowStyle |= linuxMinimize;
01136                 window->decorators = 1;
01137             }
01138
01139             if (decorators & maximizeButton)
01140             {
01141                 window->currentWindowStyle |= linuxMaximize;
01142                 window->decorators = 1;
01143             }
01144
01145             if (decorators & icon)
01146             {
01147                 //Linux (at least cinnamon) does not have icons in the window. only in the task bar icon
01148             }
01149
01150             //just need to set it to 1 to enable all decorators that include title bar
01151             if (decorators & titleBar)
01152             {
01153                 window->decorators = 1;
01154             }
01155
01156             if (decorators & border)
01157             {
01158                 window->decorators = 1;
01159             }
01160
01161             if (decorators & sizeableBorder)
01162             {
01163                 window->decorators = 1;
01164             }
01165
01166             long hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0, 0 };
01167
01168             XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
01169                 PropModeReplace, (unsigned char*)hints, 5);
01170
01171             XMapWindow(currentDisplay, window->windowHandle);
01172 #endif
01173             return TinyWindow::error_t::success;
01174         }
01175         return TinyWindow::error_t::windowInvalid;
01176     }
```

7.5.3.4 std::error_code TinyWindow::windowManager::FocusWindow (window_t * *window*, bool *newState*) [inline]

Set the window to be in focus by name

Definition at line 895 of file TinyWindow.h.

```
00896     {
00897         if (window != nullptr)
00898         {
00899             if (newState)
00900             {
00901 #if defined(TW_WINDOWS)
00902                 SetFocus(window->windowHandle);
00903 #elif defined(TW_LINUX)
00904                 XMapWindow(currentDisplay, window->windowHandle);
00905 #endif
00906             }
00907
00908             else
00909             {
00910 #if defined(_WIN32) || defined(_WIN64)
00911                 SetFocus(nullptr);
00912 #elif defined(TW_LINUX)
00913                 XUnmapWindow(currentDisplay, window->windowHandle);
00914 #endif
```

```
00915             }
00916             return TinyWindow::error_t::success;
00917         }
00918         return TinyWindow::error_t::windowInvalid;
00919     }
```

7.5.3.5 TinyWindow::uiVec2 TinyWindow::windowManager::GetMousePositionInScreen (void) [inline]

Return the mouse position in screen co-ordinates

Definition at line 573 of file TinyWindow.h.

References screenMousePosition.

```
00574     {
00575         return screenMousePosition;
00576     }
```

7.5.3.6 int TinyWindow::windowManager::GetNumWindows (void) [inline]

Return the total amount of windows the manager has

Definition at line 565 of file TinyWindow.h.

```
00566     {
00567         return windowList.size();
00568     }
```

7.5.3.7 TinyWindow::uiVec2 TinyWindow::windowManager::GetScreenResolution (void) [inline]

Return the Resolution of the current screen

Definition at line 602 of file TinyWindow.h.

References Platform_GetScreenResolution().

Referenced by SetFullScreen().

```
00603     {
00604         uiVec2 resolution;
00605         Platform_GetScreenResolution(resolution);
00606         return resolution;
00607     }
```

7.5.3.8 std::error_code TinyWindow::windowManager::MakeWindowCurrentContext (window_t * *window*) [inline]

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

Definition at line 722 of file TinyWindow.h.

```
00723     {
00724         if (window != nullptr)
00725         {
00726 #if defined(TW_WINDOWS)
00727             wglMakeCurrent(window->deviceContextHandle,
00728                 window->glRenderingContextHandle);
00729 #elif defined(TW_LINUX)
00730             glXMakeCurrent(currentDisplay, window->windowHandle,
00731                 window->context);
00732 #endif
00733             return TinyWindow::error_t::success;
00734         }
00735         return TinyWindow::error_t::windowInvalid;
00736     }
```

7.5.3.9 std::error_code TinyWindow::windowManager::MaximizeWindow (window_t * *window,* bool *newState*) [inline]

Toggle the maximization state of the current window

Definition at line 809 of file TinyWindow.h.

References TinyWindow::window_t::currentState, TinyWindow::maximized, and TinyWindow::normal.

```
00810     {
00811         if (window != nullptr)
00812         {
00813             if (newState)
00814             {
00815                 window->currentState = state_t::maximized;
00816 #if defined(TW_WINDOWS)
00817                 ShowWindow(window->windowHandle, SW_MAXIMIZE);
00818 #elif defined(TW_LINUX)
00819                 XEvent currentEvent;
00820                 memset(&currentEvent, 0, sizeof(currentEvent));
00821 
00822                 currentEvent.xany.type = ClientMessage;
00823                 currentEvent.xclient.message_type = AtomState;
00824                 currentEvent.xclient.format = 32;
00825                 currentEvent.xclient.window = window->windowHandle;
00826                 currentEvent.xclient.data.l[0] = (window->currentState ==
      state_t::maximized);
00827                 currentEvent.xclient.data.l[1] = AtomMaxVert;
00828                 currentEvent.xclient.data.l[2] = AtomMaxHorz;
00829 
00830                 XSendEvent(currentDisplay,
00831                     XDefaultRootWindow(currentDisplay),
00832                     0, SubstructureNotifyMask, &currentEvent);
00833 #endif
00834             }
00835 
00836             else
00837             {
00838                 window->currentState = state_t::normal;
00839 #if defined(TW_WINDOWS)
00840                 ShowWindow(window->windowHandle, SW_RESTORE);
00841 #elif defined(TW_LINUX)
00842                 XEvent currentEvent;
00843                 memset(&currentEvent, 0, sizeof(currentEvent));
00844 
00845                 currentEvent.xany.type = ClientMessage;
00846                 currentEvent.xclient.message_type = AtomState;
00847                 currentEvent.xclient.format = 32;
00848                 currentEvent.xclient.window = window->windowHandle;
00849                 currentEvent.xclient.data.l[0] = (window->currentState ==
      state_t::maximized);
00850                 currentEvent.xclient.data.l[1] = AtomMaxVert;
```

```
00851                 currentEvent.xclient.data.l[2] = AtomMaxHorz;
00852
00853                 XSendEvent(currentDisplay,
00854                     XDefaultRootWindow(currentDisplay),
00855                     0, SubstructureNotifyMask, &currentEvent);
00856 #endif
00857             }
00858             return TinyWindow::error_t::success;
00859         }
00860         return TinyWindow::error_t::windowInvalid;
00861     }
```

7.5.3.10 std::error_code TinyWindow::windowManager::MinimizeWindow (window_t * *window*, bool *newState*) [inline]

Toggle the minimization state of the given window

Definition at line 776 of file TinyWindow.h.

References TinyWindow::window_t::currentState, TinyWindow::minimized, and TinyWindow::normal.

```
00777     {
00778         if (window != nullptr)
00779         {
00780             if (newState)
00781             {
00782                 window->currentState = state_t::minimized;
00783
00784 #if defined(TW_WINDOWS)
00785                 ShowWindow(window->windowHandle, SW_MINIMIZE);
00786 #elif defined(TW_LINUX)
00787                 XIconifyWindow(currentDisplay,
00788                     window->windowHandle, 0);
00789 #endif
00790             }
00791
00792             else
00793             {
00794                 window->currentState = state_t::normal;
00795 #if defined(TW_WINDOWS)
00796                 ShowWindow(window->windowHandle, SW_RESTORE);
00797 #elif defined(TW_LINUX)
00798                 XMapWindow(currentDisplay, window->windowHandle);
00799 #endif
00800             }
00801             return TinyWindow::error_t::success;
00802         }
00803         return TinyWindow::error_t::windowInvalid;
00804     }
```

7.5.3.11 void TinyWindow::windowManager::Platform_GetScreenResolution (uiVec2 *resolution*) [inline], [private]

Definition at line 1395 of file TinyWindow.h.

Referenced by GetScreenResolution().

```
01396     {
01397 #if defined(TW_WINDOWS)
01398         RECT screen;
01399         HWND desktop = GetDesktopWindow();
01400         GetWindowRect(desktop, &screen);
01401         resolution.width = screen.right;
01402         resolution.height = screen.bottom;
01403 #elif defined(TW_LINUX)
01404         resolution.width = WidthOfScreen(XDefaultScreenOfDisplay(currentDisplay));
01405         resolution.height = HeightOfScreen(XDefaultScreenOfDisplay(currentDisplay));
01406
01407         screenResolution.x = resolution.width;
01408         screenResolution.y = resolution.height;
01409 #endif
01410     }
```

7.5.3.12 std::error_code TinyWindow::windowManager::Platform_InitializeGL (window_t * *window*) [inline], [private]

Definition at line 1329 of file TinyWindow.h.

References TinyWindow::window_t::contextCreated.

```
01330     {
01331 #if defined(TW_WINDOWS)
01332         window->deviceContextHandle = GetDC(window->windowHandle);
01333         InitializePixelFormat(window);
01334         window->glRenderingContextHandle = wglCreateContext(window->deviceContextHandle);
01335         wglMakeCurrent(window->deviceContextHandle, window->glRenderingContextHandle);
01336 
01337         window->contextCreated = (window->glRenderingContextHandle != nullptr);
01338 
01339         if (window->contextCreated)
01340         {
01341             return TinyWindow::error_t::success;
01342         }
01343 
01344         return TinyWindow::error_t::invalidContext;
01345 #elif defined(TW_LINUX)
01346         if (!window->context)
01347         {
01348             window->context = glXCreateContext(
01349                 currentDisplay,
01350                 window->visualInfo,
01351                 0,
01352                 true);
01353 
01354             if (window->context)
01355             {
01356                 glXMakeCurrent(currentDisplay,
01357                     window->windowHandle,
01358                     window->context);
01359 
01360                 XWindowAttributes l_Attributes;
01361 
01362                 XGetWindowAttributes(currentDisplay,
01363                     window->windowHandle, &l_Attributes);
01364                 window->position.x = l_Attributes.x;
01365                 window->position.y = l_Attributes.y;
01366 
01367                 window->contextCreated = true;
01368                 InitializeAtoms();
01369                 return TinyWindow::error_t::success;
01370             }
01371             return TinyWindow::error_t::linuxCannotConnectXServer
    ;
01372         }
01373 
01374         else
01375         {
01376             return TinyWindow::error_t::existingContext;
01377         }
01378         return TinyWindow::error_t::existingContext;
01379 #endif
01380     }
```

7.5.3.13 void TinyWindow::windowManager::Platform_InitializeWindow (window_t * *window*) [inline], [private]

Definition at line 1320 of file TinyWindow.h.

```
01321     {
01322 #if defined(TW_WINDOWS)
01323         Windows_InitializeWindow(window);
01324 #elif defined(TW_LINUX)
01325         Linux_InitializeWindow(window);
01326 #endif
01327     }
```

7.5.3.14 void TinyWindow::windowManager::Platform_SetMousePositionInScreen ( ) [inline], [private]

Definition at line 1382 of file TinyWindow.h.

References screenMousePosition.

Referenced by SetMousePositionInScreen().

```
01383     {
01384 #if defined(TW_WINDOWS)
01385         SetCursorPos(screenMousePosition.y,
      screenMousePosition.y);
01386 #elif defined(TW_LINUX)
01387         XWarpPointer(currentDisplay, None,
01388             XDefaultRootWindow(currentDisplay), 0, 0,
01389             screenResolution.x,
01390             screenResolution.y,
01391             screenMousePosition.x, screenMousePosition.
      y);
01392 #endif
01393     }
```

7.5.3.15 void TinyWindow::windowManager::Platform_SetMousePositionInWindow (window_t * *window*, unsigned int *x*, unsigned int *y*) [inline], [private]

Definition at line 1443 of file TinyWindow.h.

Referenced by SetMousePositionInWindow().

```
01444     {
01445 #if defined(TW_WINDOWS)
01446         POINT mousePoint;
01447         mousePoint.x = x;
01448         mousePoint.y = y;
01449         ScreenToClient(window->windowHandle, &mousePoint);
01450         SetCursorPos(mousePoint.x, mousePoint.y);
01451 #elif defined(TW_LINUX)
01452         XWarpPointer(
01453             currentDisplay,
01454             window->windowHandle, window->windowHandle,
01455             window->position.x, window->position.y,
01456             window->resolution.width, window->resolution.height,
01457             x, y);
01458 #endif
01459     }
```

7.5.3.16 void TinyWindow::windowManager::Platform_SetWindowPosition (window_t * *window*, unsigned int *x*, unsigned int *y*) [inline], [private]

Definition at line 1425 of file TinyWindow.h.

References TinyWindow::window_t::resolution.

Referenced by SetWindowPosition().

```
01426     {
01427 #if defined(TW_WINDOWS)
01428         SetWindowPos(window->windowHandle, HWND_TOP, x, y,
01429             window->resolution.x, window->resolution.y,
01430             SWP_SHOWWINDOW | SWP_NOSIZE);
01431 #elif defined(TW_LINUX)
01432         XWindowChanges windowChanges;
01433
01434         windowChanges.x = x;
01435         windowChanges.y = y;
01436
01437         XConfigureWindow(
01438             currentDisplay,
01439             window->windowHandle, CWX | CWY, &windowChanges);
01440 #endif
01441     }
```

7.5.3.17 void TinyWindow::windowManager::Platform_SetWindowResolution (window_t * *window*) [inline], [private]

Definition at line 1412 of file TinyWindow.h.

References TinyWindow::window_t::position, and TinyWindow::window_t::resolution.

Referenced by SetWindowResolution().

```
01413     {
01414 #if defined(TW_WINDOWS)
01415         SetWindowPos(window->windowHandle, HWND_TOP,
01416             window->position.x, window->position.y,
01417             window->resolution.x, window->resolution.y,
01418             SWP_SHOWWINDOW | SWP_NOMOVE);
01419 #elif defined(TW_LINUX)
01420         XResizeWindow(currentDisplay,
01421             window->windowHandle, window->resolution.x, window->resolution.y);
01422 #endif
01423     }
```

7.5.3.18 void TinyWindow::windowManager::PollForEvents (void) [inline]

Ask the window manager to poll for events

Definition at line 941 of file TinyWindow.h.

Referenced by main().

```
00942     {
00943 #if defined(TW_WINDOWS)
00944         //only process events if there are any to process
00945         if (PeekMessage(&winMessage, 0, 0, 0, PM_REMOVE))
00946         {
00947             TranslateMessage(&winMessage);
00948             DispatchMessage(&winMessage);
00949         }
00950 #elif defined(TW_LINUX)
00951         //if there are any events to process
00952         if (XEventsQueued(currentDisplay, QueuedAfterReading))
00953         {
00954             XNextEvent(currentDisplay, &currentEvent);
00955             Linux_ProcessEvents(currentEvent);
00956         }
00957 #endif
00958     }
```

7.5.3.19 std::error_code TinyWindow::windowManager::RemoveWindow (window_t * *window*) [inline]

Remove window from the manager by name

Definition at line 980 of file TinyWindow.h.

References ShutdownWindow().

```
00981     {
00982         if (window != nullptr)
00983         {
00984             ShutdownWindow(window);
00985             return TinyWindow::error_t::success;
00986         }
00987         return TinyWindow::error_t::windowInvalid;
00988     }
```

7.5.3.20 std::error_code TinyWindow::windowManager::RestoreWindow (window_t * *window*) [inline]

Restore the window by name

Definition at line 924 of file TinyWindow.h.

```
00925     {
00926         if (window != nullptr)
00927         {
00928 #if defined(TW_WINDOWS)
00929             ShowWindow(window->windowHandle, SW_RESTORE);
00930 #elif defined(TW_LINUX)
00931             XMapWindow(currentDisplay, window->windowHandle);
00932 #endif
00933             return TinyWindow::error_t::success;
00934         }
00935         return TinyWindow::error_t::windowInvalid;
00936     }
```

7.5.3.21 std::error_code TinyWindow::windowManager::SetFullScreen (window_t * *window,* bool *newState*) [inline]

Toggle the given window's full screen mode

Definition at line 741 of file TinyWindow.h.

References TinyWindow::window_t::currentState, TinyWindow::fullscreen, GetScreenResolution(), and TinyWindow::normal.

```
00742     {
00743         if (window != nullptr)
00744         {
00745             window->currentState = (newState == true) ? state_t::fullscreen :
    state_t::normal;
00746
00747 #if defined(TW_WINDOWS)
00748             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
00749                 WS_SYSMENU | WS_POPUP | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_VISIBLE);
00750
00751             MoveWindow(window->windowHandle, 0, 0,
    windowManager::GetScreenResolution().width,
00752                 windowManager::GetScreenResolution().
    height, true);
00753 #elif defined(TW_LINUX)
00754             XEvent currentEvent;
00755             memset(&currentEvent, 0, sizeof(currentEvent));
00756
00757             currentEvent.xany.type = ClientMessage;
00758             currentEvent.xclient.message_type = AtomState;
00759             currentEvent.xclient.format = 32;
00760             currentEvent.xclient.window = window->windowHandle;
00761             currentEvent.xclient.data.l[0] = window->currentState ==
    state_t::fullscreen;
00762             currentEvent.xclient.data.l[1] = AtomFullScreen;
00763
00764             XSendEvent(currentDisplay,
00765                 XDefaultRootWindow(currentDisplay),
00766                 0, SubstructureNotifyMask, &currentEvent);
00767 #endif
00768             return TinyWindow::error_t::success;
00769         }
00770         return TinyWindow::error_t::windowInvalid;
00771     }
```

7.5.3.22 void TinyWindow::windowManager::SetMousePositionInScreen (TinyWindow::uiVec2 *mousePosition*) [inline]

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

Definition at line 581 of file TinyWindow.h.

References Platform_SetMousePositionInScreen(), and screenMousePosition.

```
00582     {
00583         screenMousePosition.x = mousePosition.x;
00584         screenMousePosition.y = mousePosition.y;
00585 
00586         Platform_SetMousePositionInScreen();
00587     }
```

7.5.3.23 void TinyWindow::windowManager::SetMousePositionInScreen (unsigned int *x*, unsigned int *y*) [inline]

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

Definition at line 591 of file TinyWindow.h.

References Platform_SetMousePositionInScreen(), and screenMousePosition.

```
00592     {
00593         screenMousePosition.x = x;
00594         screenMousePosition.y = y;
00595 
00596         Platform_SetMousePositionInScreen();
00597     }
```

7.5.3.24 std::error_code TinyWindow::windowManager::SetMousePositionInWindow (window_t * *window*, TinyWindow::uiVec2 *mousePosition*) [inline]

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

Definition at line 674 of file TinyWindow.h.

References TinyWindow::window_t::mousePosition, and Platform_SetMousePositionInWindow().

```
00675     {
00676         if (window != nullptr)
00677         {
00678             window->mousePosition.x = mousePosition.x;
00679             window->mousePosition.y = mousePosition.y;
00680 
00681             Platform_SetMousePositionInWindow(window, mousePosition.
      x, mousePosition.y);
00682             return TinyWindow::error_t::success;
00683         }
00684         return TinyWindow::error_t::windowInvalid;
00685     }
```

7.5.3.25 std::error_code TinyWindow::windowManager::SetMousePositionInWindow (window_t * *window*, unsigned int *x*, unsigned int *y*) [inline]

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

Definition at line 689 of file TinyWindow.h.

References TinyWindow::window_t::mousePosition, and Platform_SetMousePositionInWindow().

```
00690     {
00691         if (window != nullptr)
00692         {
00693             window->mousePosition.x = x;
00694             window->mousePosition.y = y;
00695 
00696             Platform_SetMousePositionInWindow(window, x, y);
00697             return TinyWindow::error_t::success;
00698         }
00699         return TinyWindow::error_t::windowInvalid;
00700     }
```

7.5.3.26 std::error_code TinyWindow::windowManager::SetWindowIcon (void) [inline]

Set the window icon by name (currently not functional)

Definition at line 887 of file TinyWindow.h.

```
00888     {
00889         return TinyWindow::error_t::functionNotImplemented;
00890     }
```

7.5.3.27 std::error_code TinyWindow::windowManager::SetWindowPosition (window_t * *window*, TinyWindow::uiVec2 *windowPosition*) [inline]

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

Definition at line 643 of file TinyWindow.h.

References Platform_SetWindowPosition(), and TinyWindow::window_t::position.

```
00644     {
00645         if (window != nullptr)
00646         {
00647             window->position.x = windowPosition.x;
00648             window->position.y = windowPosition.y;
00649 
00650             Platform_SetWindowPosition(window, windowPosition.
      x, windowPosition.y);
00651             return TinyWindow::error_t::success;
00652         }
00653         return TinyWindow::error_t::windowInvalid;
00654     }
```

7.5.3.28 std::error_code TinyWindow::windowManager::SetWindowPosition (window_t * *window*, unsigned int *x*, unsigned int *y*) [inline]

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

Definition at line 658 of file TinyWindow.h.

References Platform_SetWindowPosition(), and TinyWindow::window_t::position.

```
00659     {
00660         if (window != nullptr)
00661         {
00662             window->position.x = x;
00663             window->position.y = y;
00664 
00665             Platform_SetWindowPosition(window, x, y);
00666             return TinyWindow::error_t::success;
00667         }
00668         return TinyWindow::error_t::windowInvalid;
00669     }
```

7.5.3.29 std::error_code TinyWindow::windowManager::SetWindowResolution (window_t * *window*, TinyWindow::uiVec2 *resolution*) [inline]

Set the Size/Resolution of the given window

Definition at line 612 of file TinyWindow.h.

References Platform_SetWindowResolution(), and TinyWindow::window_t::resolution.

```
00613     {
00614         if (window != nullptr)
00615         {
00616             window->resolution.width = resolution.width;
00617             window->resolution.height = resolution.height;
00618 
00619             Platform_SetWindowResolution(window);
00620             return TinyWindow::error_t::success;
00621         }
00622         return TinyWindow::error_t::windowInvalid;
00623     }
```

7.5.3.30 std::error_code TinyWindow::windowManager::SetWindowResolution (window_t * *window*, unsigned int *width*, unsigned int *height*) [inline]

Set the Size/Resolution of the given window

Definition at line 627 of file TinyWindow.h.

References Platform_SetWindowResolution(), and TinyWindow::window_t::resolution.

```
00628     {
00629         if (window != nullptr)
00630         {
00631             window->resolution.width = width;
00632             window->resolution.height = height;
00633 
00634             Platform_SetWindowResolution(window);
00635             return TinyWindow::error_t::success;
00636         }
00637         return TinyWindow::error_t::windowInvalid;
00638     }
```

7.5.3.31 std::error_code TinyWindow::windowManager::SetWindowStyle (window_t * *window*, style_t *windowStyle*) [inline]

Set the window style preset by name

Definition at line 993 of file TinyWindow.h.

References TinyWindow::bare, TinyWindow::border, TinyWindow::closeButton, EnableWindowDecorators(), TinyWindow::maximizeButton, TinyWindow::minimizeButton, TinyWindow::normal, TinyWindow::popup, and TinyWindow::titleBar.

```
00994     {
00995         if (window != nullptr)
00996         {
00997 #if defined(TW_WINDOWS)
00998             switch (windowStyle)
00999             {
01000             case style_t::normal:
01001             {
01002                 EnableWindowDecorators(window, titleBar |
      border |
01003                     closeButton | minimizeButton |
      maximizeButton);
01004                 break;
01005             }
01006
01007             case style_t::popup:
01008             {
01009                 EnableWindowDecorators(window, 0);
01010                 break;
01011             }
01012
01013             case style_t::bare:
01014             {
01015                 EnableWindowDecorators(window, titleBar |
      border);
01016                 break;
01017             }
01018
01019             default:
01020             {
01021                 return TinyWindow::error_t::invalidWindowStyle;
01022             }
01023             }
01024
01025 #elif defined(TW_LINUX)
01026             switch (windowStyle)
01027             {
01028             case style_t::normal:
01029             {
01030                 window->decorators = (1L << 2);
01031                 window->currentWindowStyle = linuxMove | linuxClose |
01032                     linuxMaximize | linuxMinimize;
01033                 long Hints[5] = { hint_t::function | hint_t::decorator, window->currentWindowStyle, window
      ->decorators, 0, 0 };
01034
01035                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01036                     (unsigned char*)Hints, 5);
01037
01038                 XMapWindow(currentDisplay, window->windowHandle);
01039                 break;
01040             }
01041
01042             case style_t::bare:
01043             {
01044                 window->decorators = (1L << 2);
01045                 window->currentWindowStyle = (1L << 2);
01046                 long Hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0,
      0 };
01047
01048                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01049                     (unsigned char*)Hints, 5);
01050
01051                 XMapWindow(currentDisplay, window->windowHandle);
01052                 break;
01053             }
01054
01055             case style_t::popup:
```

```
01056             {
01057                 window->decorators = 0;
01058                 window->currentWindowStyle = (1L << 2);
01059                 long Hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0,
      0 };
01060
01061                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01062                     (unsigned char*)Hints, 5);
01063
01064                 XMapWindow(currentDisplay, window->windowHandle);
01065                 break;
01066             }
01067
01068             default:
01069             {
01070                 return TinyWindow::error_t::invalidWindowStyle;
01071             }
01072             }
01073 #endif
01074             return TinyWindow::error_t::success;
01075         }
01076         return TinyWindow::error_t::windowInvalid;
01077     }
```

7.5.3.32 std::error_code TinyWindow::windowManager::SetWindowTitleBar (window_t * *window*, const char * *newTitle*) [inline]

Set the window title bar by name

Definition at line 866 of file TinyWindow.h.

```
00867     {
00868         if (newTitle != nullptr)
00869         {
00870             if (window != nullptr)
00871             {
00872 #if defined(TW_WINDOWS)
00873                 SetWindowText(window->windowHandle, newTitle);
00874 #elif defined(TW_LINUX)
00875                 XStoreName(currentDisplay, window->windowHandle, newTitle);
00876 #endif
00877                 return TinyWindow::error_t::success;
00878             }
00879             return TinyWindow::error_t::windowInvalid;
00880         }
00881         return TinyWindow::error_t::invalidTitlebar;
00882     }
```

7.5.3.33 void TinyWindow::windowManager::ShutDown (void) [inline]

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

Definition at line 529 of file TinyWindow.h.

Referenced by main(), and ~windowManager().

```
00530     {
00531 #if defined(__linux__)
00532         Linux_Shutdown();
00533 #endif
00534         windowList.clear();
00535     }
```

7.5.3.34 void TinyWindow::windowManager::ShutdownWindow (window_t * *window*) [inline], [private]

Definition at line 1461 of file TinyWindow.h.

References TinyWindow::window_t::name.

Referenced by RemoveWindow().

```
01462     {
01463 #if defined(TW_WINDOWS)
01464         if (window->glRenderingContextHandle)
01465         {
01466             wglMakeCurrent(nullptr, nullptr);
01467             wglDeleteContext(window->glRenderingContextHandle);
01468         }
01469 
01470         if (window->paletteHandle)
01471         {
01472             DeleteObject(window->paletteHandle);
01473         }
01474         ReleaseDC(window->windowHandle, window->deviceContextHandle);
01475         UnregisterClass(window->name, window->instanceHandle);
01476 
01477         FreeModule(window->instanceHandle);
01478 
01479         window->deviceContextHandle = nullptr;
01480         window->windowHandle = nullptr;
01481         window->glRenderingContextHandle = nullptr;
01482 
01483         if (windowList.size() > 1)
01484         {
01485             windowList.erase(windowList.begin() + window->iD);
01486         }
01487 
01488         else
01489         {
01490             windowList.erase(windowList.begin());
01491         }
01492 #elif defined(TW_LINUX)
01493         if (window->currentState == state_t::fullscreen)
01494         {
01495             RestoreWindow(window);
01496         }
01497 
01498         glXDestroyContext(currentDisplay, window->context);
01499         XUnmapWindow(currentDisplay, window->windowHandle);
01500         XDestroyWindow(currentDisplay, window->windowHandle);
01501         window->windowHandle = 0;
01502         window->context = 0;
01503 #endif
01504     }
```

7.5.3.35 std::error_code TinyWindow::windowManager::SwapWindowBuffers (window_t * *window*) [inline]

Swap the draw buffers of the given window

Definition at line 705 of file TinyWindow.h.

Referenced by main().

```
00706     {
00707         if (window != nullptr)
00708         {
00709 #if defined(TW_WINDOWS)
00710             SwapBuffers(window->deviceContextHandle);
00711 #elif defined(TW_LINUX)
00712             glXSwapBuffers(currentDisplay, window->windowHandle);
00713 #endif
00714             return TinyWindow::error_t::success;
00715         }
00716         return TinyWindow::error_t::windowInvalid;
00717     }
```

7.5.3.36 void TinyWindow::windowManager::WaitForEvents (void) [inline]

Ask the window manager to wait for events

Definition at line 963 of file TinyWindow.h.

```
00964     {
00965 #if defined(TW_WINDOWS)
00966         //process even if there aren't any to process
00967         GetMessage(&winMessage, 0, 0, 0);
00968         TranslateMessage(&winMessage);
00969         DispatchMessage(&winMessage);
00970 #elif defined(TW_LINUX)
00971         //even if there aren't any events to process
00972         XNextEvent(currentDisplay, &currentEvent);
00973         Linux_ProcessEvents(currentEvent);
00974 #endif
00975     }
```

7.5.4 Field Documentation

7.5.4.1 TinyWindow::uiVec2 TinyWindow::windowManager::screenMousePosition [private]

Definition at line 1318 of file TinyWindow.h.

Referenced by GetMousePositionInScreen(), Platform_SetMousePositionInScreen(), and SetMousePositionIn↩Screen().

7.5.4.2 TinyWindow::uiVec2 TinyWindow::windowManager::screenResolution [private]

Definition at line 1317 of file TinyWindow.h.

Referenced by windowManager().

7.5.4.3 std::vector< std::unique_ptr<window_t> > TinyWindow::windowManager::windowList [private]

Definition at line 1315 of file TinyWindow.h.

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

- C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h

Chapter 8

File Documentation

8.1 C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Example/Example.cpp File Reference

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

Functions

- void handleKeyPresses (unsigned int key, keyState_t keyState)
- int main ()

8.1.1 Function Documentation

8.1.1.1 void handleKeyPresses (unsigned int *key,* keyState_t *keyState*)

Definition at line 4 of file Example.cpp.

References TinyWindow::down.

```
00005 {
00006     if(keyState == keyState_t::down)
00007     {
00008         printf("%c \t", key);
00009     }
00010 }
```

8.1.1.2 int main ( )

Definition at line 12 of file Example.cpp.

References TinyWindow::windowManager::AddWindow(), TinyWindow::window_t::decorators, TinyWindow::windowManager::PollForEvents(), TinyWindow::window_t::shouldClose, TinyWindow::windowManager::ShutDown(), TinyWindow::windowManager::SwapWindowBuffers(), and TinyWindow::windowManager::windowManager().

```
00013 {
00014     windowManager* manager = new windowManager();
00015     window_t* window = nullptr;
00016 
00017     window = manager->AddWindow("Example");
00018     window->keyEvent = handleKeyPresses;
00019 
00020     while (!window->shouldClose)
00021     {
00022         glClearColor(0.25f, 0.25f, 0.25f, 1.0f);
00023         manager->PollForEvents();// or WaitForEvents
00024 
00025         //manager->MakeWindowCurrentContext(window);
00026         manager->SwapWindowBuffers(window);
00027         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00028     }
00029 
00030     manager->ShutDown();
00031     return 0;
00032 }
```

8.2 Example.cpp

```
00001 #include "TinyWindow.h"
00002 
00003 using namespace TinyWindow;
00004 void handleKeyPresses(unsigned int key, keyState_t keyState)
00005 {
00006     if(keyState == keyState_t::down)
00007     {
00008         printf("%c \t", key);
00009     }
00010 }
00011 
00012 int main()
00013 {
00014     windowManager* manager = new windowManager();
00015     window_t* window = nullptr;
00016 
00017     window = manager->AddWindow("Example");
00018     window->keyEvent = handleKeyPresses;
00019 
00020     while (!window->shouldClose)
00021     {
00022         glClearColor(0.25f, 0.25f, 0.25f, 1.0f);
00023         manager->PollForEvents();// or WaitForEvents
00024 
00025         //manager->MakeWindowCurrentContext(window);
00026         manager->SwapWindowBuffers(window);
00027         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00028     }
00029 
00030     manager->ShutDown();
00031     return 0;
00032 }
```

8.3 C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/Include/TinyWindow.h File Reference

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <vector>
#include <limits.h>
#include <string.h>
#include <functional>
#include <memory>
#include <system_error>
```

Data Structures

- struct TinyWindow::uiVec2
- class TinyWindow::errorCategory_t
- struct std::is_error_code_enum< TinyWindow::error_t >
- struct TinyWindow::window_t
- class TinyWindow::windowManager

Namespaces

- TinyWindow

Typedefs

- typedef std::function< void(unsigned int, keyState_t)> TinyWindow::keyEvent_t
- typedef std::function< void(mouseButton_t, buttonState_t)> TinyWindow::mouseButtonEvent_t
- typedef std::function< void(mouseScroll_t)> TinyWindow::mouseWheelEvent_t
- typedef std::function< void(void)> TinyWindow::destroyedEvent_t
- typedef std::function< void(void)> TinyWindow::maximizedEvent_t
- typedef std::function< void(void)> TinyWindow::minimizedEvent_t
- typedef std::function< void(bool)> TinyWindow::focusEvent_t
- typedef std::function< void(unsigned int, unsigned int)> TinyWindow::movedEvent_t
- typedef std::function< void(unsigned int, unsigned int)> TinyWindow::resizeEvent_t
- typedef std::function< void(unsigned int, unsigned int, unsigned int, unsigned int)> TinyWindow::mouse↩ MoveEvent_t

Enumerations

- enum TinyWindow::keyState_t { TinyWindow::bad, TinyWindow::keyState_t::up, TinyWindow::keyState_t↩ ::down }
- enum TinyWindow::key_t {
  TinyWindow::bad = -1, TinyWindow::bad, TinyWindow::first = 256 + 1, TinyWindow::F1,
  TinyWindow::F2, TinyWindow::F3, TinyWindow::F4, TinyWindow::F5,
  TinyWindow::F6, TinyWindow::F7, TinyWindow::F8, TinyWindow::F9,
  TinyWindow::F10, TinyWindow::F11, TinyWindow::F12, TinyWindow::capsLock,
  TinyWindow::leftShift, TinyWindow::rightShift, TinyWindow::leftControl, TinyWindow::rightControl,
  TinyWindow::leftWindow, TinyWindow::rightWindow, TinyWindow::leftAlt, TinyWindow::rightAlt,
  TinyWindow::enter, TinyWindow::printScreen, TinyWindow::scrollLock, TinyWindow::numLock,
  TinyWindow::pause, TinyWindow::insert, TinyWindow::home, TinyWindow::end,
  TinyWindow::pageUp, TinyWindow::pageDown, TinyWindow::arrowDown, TinyWindow::arrowUp,
  TinyWindow::arrowLeft, TinyWindow::arrowRight, TinyWindow::keypadDivide, TinyWindow::keypadMultiply,
  TinyWindow::keypadSubtract, TinyWindow::keypadAdd, TinyWindow::keypadEnter, TinyWindow::keypad↩ Period,
  TinyWindow::keypad0, TinyWindow::keypad1, TinyWindow::keypad2, TinyWindow::keypad3,
  TinyWindow::keypad4, TinyWindow::keypad5, TinyWindow::keypad6, TinyWindow::keypad7,
  TinyWindow::keypad8, TinyWindow::keypad9, TinyWindow::backspace, TinyWindow::tab,
  TinyWindow::del, TinyWindow::escape, TinyWindow::last = escape }

- enum TinyWindow::buttonState_t { TinyWindow::buttonState_t::up, TinyWindow::buttonState_t::down }
- enum TinyWindow::mouseButton_t { TinyWindow::mouseButton_t::left, TinyWindow::mouseButton_t::right, TinyWindow::mouseButton_t::middle, TinyWindow::mouseButton_t::last }
- enum TinyWindow::mouseScroll_t { TinyWindow::mouseScroll_t::down, TinyWindow::mouseScroll_t::up }
- enum TinyWindow::style_t { TinyWindow::style_t::bare, TinyWindow::style_t::normal, TinyWindow::style_t::popup }
- enum TinyWindow::state_t { TinyWindow::state_t::normal, TinyWindow::state_t::maximized, TinyWindow::state_t::minimized, TinyWindow::state_t::fullscreen }
- enum TinyWindow::decorator_t {
  TinyWindow::titleBar = 0x01, TinyWindow::icon = 0x02, TinyWindow::border = 0x04, TinyWindow::minimizeButton = 0x08,
  TinyWindow::maximizeButton = 0x010, TinyWindow::closeButton = 0x20, TinyWindow::sizeableBorder = 0x40 }
- enum TinyWindow::error_t : int {
  TinyWindow::error_t::success, TinyWindow::error_t::invalidWindowName, TinyWindow::error_t::invalidIconPath, TinyWindow::error_t::invalidWindowIndex,
  TinyWindow::error_t::invalidWindowState, TinyWindow::error_t::invalidResolution, TinyWindow::error_t::invalidContext, TinyWindow::error_t::existingContext,
  TinyWindow::error_t::notInitialized, TinyWindow::error_t::alreadyInitialized, TinyWindow::error_t::invalidTitlebar, TinyWindow::error_t::invalidCallback,
  TinyWindow::error_t::windowInvalid, TinyWindow::error_t::invalidWindowStyle, TinyWindow::error_t::functionNotImplemented, TinyWindow::error_t::linuxCannotConnectXServer,
  TinyWindow::error_t::linuxInvalidVisualinfo, TinyWindow::error_t::linuxCannotCreateWindow, TinyWindow::error_t::linuxFunctionNotImplemented, TinyWindow::error_t::windowsCannotCreateWindows,
  TinyWindow::error_t::windowsCannotInitialize, TinyWindow::error_t::windowsFunctionNotImplemented }

Functions

- std::error_code TinyWindow::make_error_code (error_t eCode)

Variables

- const int TinyWindow::defaultWindowWidth = 1280
- const int TinyWindow::defaultWindowHeight = 720

8.4 TinyWindow.h

```
00001 //created by Ziyad Barakat 2014 - 2015
00002
00003 #ifndef TINYWINDOW_H
00004 #define TINYWINDOW_H
00005
00006 #if defined(_WIN32) || defined(_WIN64)
00007 #define TW_WINDOWS
00008 #include <Windows.h>
00009 #include <gl/GL.h>
00010 #include <io.h>
00011 #include <fcntl.h>
00012 #if defined(_MSC_VER)
00013 //this automatically loads the OpenGL library if you are using Visual studio. feel free to comment out
00014 #pragma comment (lib, "opengl32.lib")
00015 //this makes sure that the entry point of your program is main() not Winmain(). feel free to comment
      out
00016 #pragma comment(linker, "/subsystem:windows /ENTRY:mainCRTStartup")
00017 #endif //_MSC_VER
00018 #endif  //_WIN32 || _WIN64
00019
00020 #if defined(__linux__)
00021 #define TW_LINUX
00022 #include <GL/glx.h>
```

```
00023 #include <X11/X.h>
00024 #include <X11/keysym.h>
00025 #include <X11/Xatom.h>
00026 #include <X11/XKBlib.h>
00027 #endif //__linux__
00028
00029 #include <stdio.h>
00030 #include <stdlib.h>
00031 #include <vector>
00032 #include <limits.h>
00033 #include <string.h>
00034 #include <functional>
00035 #include <memory>
00036 #include <system_error>
00037
00038 namespace TinyWindow
00039 {
00040
00041     const int defaultWindowWidth = 1280;
00042     const int defaultWindowHeight = 720;
00043
00044     struct uiVec2
00045     {
00046         uiVec2()
00047         {
00048             this->x = 0;
00049             this->y = 0;
00050         }
00051
00052         uiVec2(unsigned int x, unsigned int y)
00053         {
00054             this->x = x;
00055             this->y = y;
00056         }
00057
00058         union
00059         {
00060             unsigned int x;
00061             unsigned int width;
00062         };
00063
00064         union
00065         {
00066             unsigned int y;
00067             unsigned int height;
00068         };
00069
00070         static uiVec2 Zero()
00071         {
00072             return uiVec2(0, 0);
00073         }
00074     };
00075
00076     enum class keyState_t
00077     {
00078         bad,                                /**< If get key state fails (could not name it
      ERROR) */
00079         up,                                 /**< The key is currently up */
00080         down,                               /**< The key is currently down */
00081     };
00082
00083     enum key_t
00084     {
00085         bad = -1,                           /**< The key pressed is considered invalid */
00086         first = 256 + 1,                    /**< The first key that is not a char */
00087         F1,                                 /**< The F1 key */
00088         F2,                                 /**< The F2 key */
00089         F3,                                 /**< The F3 key */
00090         F4,                                 /**< The F4 key */
00091         F5,                                 /**< The F5 key */
00092         F6,                                 /**< The F6 key */
00093         F7,                                 /**< The F7 key */
00094         F8,                                 /**< The F8 key */
00095         F9,                                 /**< The F9 key */
00096         F10,                                /**< The F10 key */
00097         F11,                                /**< The F11 key */
00098         F12,                                /**< The F12 key */
00099         capsLock,                           /**< The CapsLock key */
00100         leftShift,                          /**< The left Shift key */
00101         rightShift,                         /**< The right Shift key */
00102         leftControl,                        /**< The left Control key */
00103         rightControl,                       /**< The right Control key */
00104         leftWindow,                         /**< The left Window key */
00105         rightWindow,                        /**< The right Window key */
00106         leftAlt,                            /**< The left Alternate key */
00107         rightAlt,                           /**< The right Alternate key */
00108         enter,                              /**< The Enter/Return key */
```

```
00109         printScreen,                    /**< The PrintScreen key */
00110         scrollLock,                     /**< The ScrollLock key */
00111         numLock,                        /**< The NumLock key */
00112         pause,                          /**< The pause/break key */
00113         insert,                         /**< The insert key */
00114         home,                           /**< The Home key */
00115         end,                            /**< The End key */
00116         pageUp,                         /**< The PageUp key */
00117         pageDown,                       /**< The PageDown key */
00118         arrowDown,                      /**< The ArrowDown key */
00119         arrowUp,                        /**< The ArrowUp key */
00120         arrowLeft,                      /**< The ArrowLeft key */
00121         arrowRight,                     /**< The ArrowRight key */
00122         keypadDivide,                   /**< The KeyPad Divide key */
00123         keypadMultiply,                 /**< The Keypad Multiply key */
00124         keypadSubtract,                 /**< The Keypad Subtract key */
00125         keypadAdd,                      /**< The Keypad Add key */
00126         keypadEnter,                    /**< The Keypad Enter key */
00127         keypadPeriod,                   /**< The Keypad Period/Decimal key */
00128         keypad0,                        /**< The Keypad 0 key */
00129         keypad1,                        /**< The Keypad 1 key */
00130         keypad2,                        /**< The Keypad 2 key */
00131         keypad3,                        /**< The Keypad 3 key */
00132         keypad4,                        /**< The Keypad 4 key */
00133         keypad5,                        /**< The Keypad 5 key */
00134         keypad6,                        /**< The Keypad 6 key */
00135         keypad7,                        /**< The Keypad 7 key */
00136         keypad8,                        /**< The keypad 8 key */
00137         keypad9,                        /**< The Keypad 9 key */
00138         backspace,                      /**< The Backspace key */
00139         tab,                            /**< The Tab key */
00140         del,                            /**< The Delete key */
00141         escape,                         /**< The Escape key */
00142         last = escape,                  /**< The last key to be supported */
00143     };
00144
00145     enum class buttonState_t
00146     {
00147         up,                             /**< The mouse button is currently up */
00148         down                            /**< The mouse button is currently down */
00149     };
00150
00151     enum class mouseButton_t
00152     {
00153         left,                           /**< The left mouse button */
00154         right,                          /**< The right mouse button */
00155         middle,                         /**< The middle mouse button / ScrollWheel */
00156         last,                           /**< The last mouse button to be supported */
00157     };
00158
00159     enum class mouseScroll_t
00160     {
00161         down,                           /**< The mouse wheel up */
00162         up                              /**< The mouse wheel down */
00163     };
00164
00165     enum class style_t
00166     {
00167         bare,                           /**< The window has no decorators but the window
      border and title bar */
00168         normal,                         /**< The default window style for the respective
      platform */
00169         popup,                          /**< The window has no decorators */
00170     };
00171
00172     enum class state_t
00173     {
00174         normal,                         /**< The window is in its default state */
00175         maximized,                      /**< The window is currently maximized */
00176         minimized,                      /**< The window is currently minimized */
00177         fullscreen,                     /**< The window is currently full screen */
00178     };
00179
00180     enum decorator_t
00181     {
00182         titleBar = 0x01,                /**< The title bar decoration of the
      window */
00183         icon = 0x02,                    /**< The icon decoration of the window */
00184         border = 0x04,                  /**< The border decoration of the window */
00185         minimizeButton = 0x08,          /**< The minimize button decoration
      of the window */
00186         maximizeButton = 0x010,         /**< The maximize button decoration
      pf the window */
00187         closeButton = 0x20,             /**< The close button decoration of
      the window */
00188         sizeableBorder = 0x40,          /**< The sizable border decoration
      of the window */
```

```
00189     };
00190
00191     enum class error_t : int
00192     {
00193         success,                                /**< If a function call was successful*/
00194         invalidWindowName,                      /**< If an invalid window name was
      given */
00195         invalidIconPath,                        /**< If an invalid icon path was given
      */
00196         invalidWindowIndex,                     /**< If an invalid window index was
      given */
00197         invalidWindowState,                     /**< If an invalid window state was
      given */
00198         invalidResolution,                      /**< If an invalid window resolution
      was given */
00199         invalidContext,                         /**< If the OpenGL context for the
      window is invalid */
00200         existingContext,                        /**< If the window already has an
      OpenGL context */
00201         notInitialized,                         /**< If the window is being used without
      being initialized */
00202         alreadyInitialized,                     /**< If the window was already
      initialized */
00203         invalidTitlebar,                        /**< If the Title-bar text given was
      invalid */
00204         invalidCallback,                        /**< If the given event callback was
      invalid */
00205         windowInvalid,                          /**< If the window given was invalid */
00206         invalidWindowStyle,                     /**< If the window style gives is
      invalid */
00207         functionNotImplemented,                 /**< If the function has not yet
      been implemented in the current version of the API */
00208         linuxCannotConnectXServer,              /**< Linux: if cannot connect
      to an X11 server */
00209         linuxInvalidVisualinfo,                 /**< Linux: if visual
      information given was invalid */
00210         linuxCannotCreateWindow,                /**< Linux: when X11 fails to
      create a new window */
00211         linuxFunctionNotImplemented,            /**< Linux: when the
      function has not yet been implemented on the Linux in the current version of the API */
00212         windowsCannotCreateWindows,             /**< Windows: when Win32
      cannot create a window */
00213         windowsCannotInitialize,                /**< Windows: when Win32 cannot
      initialize */
00214         windowsFunctionNotImplemented,          /**< Windows: when a
      function has yet to be implemented on the Windows platform in the current version of the API */
00215     };
00216
00217     typedef std::function<void(unsigned int, keyState_t)>
              keyEvent_t;
00218     typedef std::function<void(mouseButton_t,
      buttonState_t)>
      mouseButtonEvent_t;
00219     typedef std::function<void(mouseScroll_t)>
              mouseWheelEvent_t;
00220     typedef std::function<void(void)>
      destroyedEvent_t;
00221     typedef std::function<void(void)>
      maximizedEvent_t;
00222     typedef std::function<void(void)>
      minimizedEvent_t;
00223     typedef std::function<void(bool)>
      focusEvent_t;
00224     typedef std::function<void(unsigned int, unsigned int)>
      movedEvent_t;
00225     typedef std::function<void(unsigned int, unsigned int)>
      resizeEvent_t;
00226     typedef std::function<void(unsigned int, unsigned int, unsigned int, unsigned int)>
      mouseMoveEvent_t;
00227
00228     class errorCategory_t : public std::error_category
00229     {
00230     public:
00231
00232         const char* name() const throw() override
00233         {
00234             return "tinyWindow";
00235         }
00236
00237         /**
00238         * return the error message associated with the given error number
00239         */
00240         virtual std::string message(int errorValue) const override
00241         {
00242             error_t err = (error_t)errorValue;
00243             switch (err)
00244             {
```

```
00245             case error_t::invalidWindowName:
00246             {
00247                 return "Error: invalid window name \n";
00248             }
00249
00250             case error_t::invalidIconPath:
00251             {
00252                 return "Error: invalid icon path \n";
00253             }
00254
00255             case error_t::invalidWindowIndex:
00256             {
00257                 return "Error: invalid window index \n";
00258             }
00259
00260             case error_t::invalidWindowState:
00261             {
00262                 return "Error: invalid window state \n";
00263             }
00264
00265             case error_t::invalidResolution:
00266             {
00267                 return "Error: invalid resolution \n";
00268             }
00269
00270             case error_t::invalidContext:
00271             {
00272                 return "Error: Failed to create OpenGL context \n";
00273             }
00274
00275             case error_t::existingContext:
00276             {
00277                 return "Error: context already created \n";
00278             }
00279
00280             case error_t::notInitialized:
00281             {
00282                 return "Error: Window manager not initialized \n";
00283             }
00284
00285             case error_t::alreadyInitialized:
00286             {
00287                 return "Error: window has already been initialized \n";
00288             }
00289
00290             case error_t::invalidTitlebar:
00291             {
00292                 return "Error: invalid title bar name (cannot be null or nullptr) \n";
00293             }
00294
00295             case error_t::invalidCallback:
00296             {
00297                 return "Error: invalid event callback given \n";
00298             }
00299
00300             case error_t::windowInvalid:
00301             {
00302                 return "Error: window was not found \n";
00303             }
00304
00305             case error_t::invalidWindowStyle:
00306             {
00307                 return "Error: invalid window style given \n";
00308             }
00309
00310             case error_t::functionNotImplemented:
00311             {
00312                 return "Error: I'm sorry but this function has not been implemented yet :(\n";
00313             }
00314
00315             case error_t::linuxCannotConnectXServer:
00316             {
00317                 return "Error: cannot connect to X server \n";
00318             }
00319
00320             case error_t::linuxInvalidVisualinfo:
00321             {
00322                 return "Error: Invalid visual information given \n";
00323             }
00324
00325             case error_t::linuxCannotCreateWindow:
00326             {
00327                 return "Error: failed to create window \n";
00328             }
00329
00330             case error_t::linuxFunctionNotImplemented:
00331             {
```

```
00332                 return "Error: function not implemented on Linux platform yet. sorry :(\n";
00333             }
00334
00335             case error_t::windowsCannotCreateWindows:
00336             {
00337                 return "Error: failed to create window \n";
00338             }
00339
00340             case error_t::windowsFunctionNotImplemented:
00341             {
00342                 return "Error: function not implemented on Windows platform yet. sorry ;(\n";
00343             }
00344
00345             case error_t::success:
00346             {
00347                 return "function call was successful \n";
00348             }
00349
00350             default:
00351             {
00352                 return "Error: unspecified Error \n";
00353             }
00354             }
00355         }
00356         errorCategory_t() {};
00357
00358         const static errorCategory_t& get()
00359         {
00360             const static errorCategory_t category;
00361             return category;
00362
00363         }
00364     };
00365
00366     std::error_code make_error_code(error_t eCode)
00367     {
00368         return std::error_code(static_cast<int>(eCode), errorCategory_t
      ::get());
00369     }
00370 };
00371 //ugh I hate this hack
00372 namespace std
00373 {
00374     template<> struct is_error_code_enum<TinyWindow::error_t> : std::true_type {
      };
00375 };
00376
00377 namespace TinyWindow
00378 {
00379
00380 struct window_t
00381 {
00382
00383     const char*                     name;
      /**< Name of the window */
00384     unsigned int                    iD;
      /**< ID of the Window. (where it belongs in the window manager) */
00385     int                             colorBits;
      /**< Color format of the window. (defaults to 32 bit color) */
00386     int                             depthBits;
      /**< Size of the Depth buffer. (defaults to 8 bit depth) */
00387     int                             stencilBits;
        /**< Size of the stencil buffer, (defaults to 8 bit) */
00388     keyState_t                      keys[last];
              /**< Record of keys that are either pressed or released in the respective window */
00389     buttonState_t                   mouseButton[(unsigned int)
      mouseButton_t::last];         /**< Record of mouse buttons that are either presses or
       released */
00390     TinyWindow::uiVec2              resolution;
                      /**< Resolution/Size of the window stored in an array */
00391     TinyWindow::uiVec2              position;
                    /**< Position of the Window relative to the screen co-ordinates */
00392     TinyWindow::uiVec2              mousePosition;
                         /**< Position of the Mouse cursor relative to the window co-ordinates */
00393     bool                            shouldClose;
        /**< Whether the Window should be closing */
00394     bool                            inFocus;
     /**< Whether the Window is currently in focus(if it is the current window be used) */
00395
00396     bool                            initialized;
        /**< Whether the window has been successfully initialized */
00397     bool                            contextCreated;
           /**< Whether the OpenGL context has been successfully created */
00398     bool                            isCurrentContext;
          /**< Whether the window is the current window being drawn to */
00399
00400     state_t                         currentState;
```

```
                /**< The current state of the window. these states include Normal, Minimized, Maximized and
    Full screen */
00401     unsigned int                    currentWindowStyle;
                /**< The current style of the window */
00402
00403     keyEvent_t                      keyEvent;
                /**< This is the callback to be used when a key has been pressed */
00404     mouseButtonEvent_t              mouseButtonEvent;
                            /**< This is the callback to be used when a mouse button has been pressed */
00405     mouseWheelEvent_t               mouseWheelEvent;
                          /**< This is the callback to be used when the mouse wheel has been scrolled. */
00406     destroyedEvent_t                destroyedEvent;
                         /**< This is the callback to be used when the window has been closed in a
    non-programmatic fashion */
00407     maximizedEvent_t                maximizedEvent;
                         /**< This is the callback to be used when the window has been maximized in a
    non-programmatic fashion */
00408     minimizedEvent_t                minimizedEvent;
                         /**< This is the callback to be used when the window has been minimized in a
    non-programmatic fashion */
00409     focusEvent_t                    focusEvent;
                    /**< This is the callback to be used when the window has been given focus in a
    non-programmatic fashion */
00410     movedEvent_t                    movedEvent;
                     /**< This is the callback to be used the window has been moved in a non-programmatic
    fashion */
00411     resizeEvent_t                   resizeEvent;
                      /**< This is a callback to be used when the window has been resized in a non-programmatic
    fashion */
00412     mouseMoveEvent_t                mouseMoveEvent;
                         /**< This is a callback to be used when the mouse has been moved */
00413
00414 #if defined(TW_WINDOWS)
00415
00416     HDC                             deviceContextHandle;                                    /**< A
    handle to a device context */
00417     HGLRC                           glRenderingContextHandle;                               /**< A
    handle to an OpenGL rendering context*/
00418     HPALETTE                        paletteHandle;                                          /**< A
    handle to a Win32 palette*/
00419     PIXELFORMATDESCRIPTOR           pixelFormatDescriptor;                                  /**<
    Describes the pixel format of a drawing surface*/
00420     WNDCLASS                        windowClass;                                            /**<
    Contains the window class attributes */
00421     HWND                            windowHandle;                                           /**< A
    handle to A window */
00422     HINSTANCE                       instanceHandle;
00423
00424 #else
00425
00426     Window                          windowHandle;
                /**< The X11 handle to the window. I wish they didn't name the type 'Window' */
00427     GLXContext                      context;
               /**< The handle to the GLX rendering context */
00428     XVisualInfo*                    visualInfo;
                   /**< The handle to the Visual Information. similar purpose to PixelformatDesriptor */
00429     int*                            attributes;
         /**< Attributes of the window. RGB, depth, stencil, etc */
00430     XSetWindowAttributes            setAttributes;
                              /**< The attributes to be set for the window */
00431     unsigned int                    decorators;
          /**< Enabled window decorators */
00432
00433 #endif
00434
00435     window_t(const char* name = nullptr, unsigned int iD = 0,
00436         unsigned int colorBits = 0, unsigned int depthBits = 0, unsigned int stencilBits = 0,
00437         bool shouldClose = false, state_t currentState = state_t::
    normal,
00438         keyEvent_t keyEvent = nullptr,
00439         mouseButtonEvent_t mouseButtonEvent = nullptr,
00440         mouseWheelEvent_t mouseWheelEvent = nullptr,
00441         destroyedEvent_t destroyedEvent = nullptr,
00442         maximizedEvent_t maximizedEvent = nullptr,
00443         minimizedEvent_t minimizedEvent = nullptr,
00444         focusEvent_t focusEvent = nullptr,
00445         movedEvent_t movedEvent = nullptr,
00446         resizeEvent_t resizeEvent = nullptr,
00447         mouseMoveEvent_t mouseMoveEvent = nullptr)
00448     {
00449         this->name = name;
00450         this->iD = iD;
00451         this->colorBits = colorBits;
00452         this->depthBits = depthBits;
00453         this->stencilBits = stencilBits;
00454         this->shouldClose = shouldClose;
00455         this->currentState = currentState;
```

```
00456
00457         this->keyEvent = keyEvent;
00458         this->mouseButtonEvent = mouseButtonEvent;
00459         this->mouseWheelEvent = mouseWheelEvent;
00460         this->destroyedEvent = destroyedEvent;
00461         this->maximizedEvent = maximizedEvent;
00462         this->minimizedEvent = minimizedEvent;
00463         this->focusEvent = focusEvent;
00464         this->movedEvent = movedEvent;
00465         this->resizeEvent = resizeEvent;
00466         this->mouseMoveEvent = mouseMoveEvent;
00467
00468         initialized = false;
00469         contextCreated = false;
00470         currentWindowStyle = (unsigned int)style_t
      ::normal;
00471
00472 #if defined(__linux__)
00473         context = 0;
00474 #endif
00475     }
00476 };
00477
00478 class windowManager
00479 {
00480     enum error_t : int;
00481
00482 public:
00483
00484     windowManager(void)
00485     {
00486 #if defined(TW_WINDOWS)
00487         CreateTerminal(); //feel free to comment this out
00488         RECT desktop;
00489
00490         HWND desktopHandle = GetDesktopWindow();
00491
00492         if (desktopHandle)
00493         {
00494             GetWindowRect(desktopHandle, &desktop);
00495
00496             screenResolution.x = desktop.right;
00497             screenResolution.y = desktop.bottom;
00498             return;
00499         }
00500 #elif defined(TW_LINUX)
00501         currentDisplay = XOpenDisplay(0);
00502
00503         if (!currentDisplay)
00504         {
00505             return;
00506         }
00507
00508         screenResolution.x = WidthOfScreen(
00509             XScreenOfDisplay(currentDisplay,
00510                 DefaultScreen(currentDisplay)));
00511
00512         screenResolution.y = HeightOfScreen(
00513             XScreenOfDisplay(currentDisplay,
00514                 DefaultScreen(currentDisplay)));
00515 #endif
00516     }
00517
00518     /**
00519      * Shutdown and delete all windows in the manager
00520      */
00521     ~windowManager(void)
00522     {
00523         ShutDown();
00524     }
00525
00526     /**
00527      * Use this to shutdown the window manager when your program is finished
00528      */
00529      void ShutDown(void)
00530     {
00531 #if defined(__linux__)
00532         Linux_Shutdown();
00533 #endif
00534         windowList.clear();
00535     }
00536
00537     /**
00538      * Use this to add a window to the manager. returns a pointer to the manager which allows for the
      easy creation of multiple windows
00539      */
00540     window_t* AddWindow(const char* windowName, unsigned int width =
```

```
      defaultWindowWidth, unsigned int height = defaultWindowHeight, int
      colourBits = 8, int depthBits = 8, int stencilBits = 8)
00541     {
00542         if (windowName != nullptr)
00543         {
00544             std::unique_ptr<window_t> newWindow(new window_t);
00545             newWindow->name = windowName;
00546             newWindow->resolution.width = width;
00547             newWindow->resolution.height = height;
00548             newWindow->colorBits = colourBits;
00549             newWindow->depthBits = depthBits;
00550             newWindow->stencilBits = stencilBits;
00551             newWindow->iD = GetNumWindows();
00552 
00553             windowList.push_back(std::move(newWindow));
00554             Platform_InitializeWindow(windowList.back().get());
00555 
00556             return windowList.back().get();
00557         }
00558         //PrintErrorMessage(std::error_code(invalidWindowName));
00559         return nullptr;
00560     }
00561 
00562     /**
00563      * Return the total amount of windows the manager has
00564      */
00565     int GetNumWindows(void)
00566     {
00567         return windowList.size();
00568     }
00569 
00570     /**
00571     * Return the mouse position in screen co-ordinates
00572     */
00573     TinyWindow::uiVec2 GetMousePositionInScreen(void)
00574     {
00575         return screenMousePosition;
00576     }
00577 
00578     /**
00579      * Set the position of the mouse cursor relative to screen co-ordinates
00580      */
00581     void SetMousePositionInScreen(TinyWindow::
      uiVec2 mousePosition)
00582     {
00583         screenMousePosition.x = mousePosition.x;
00584         screenMousePosition.y = mousePosition.y;
00585 
00586         Platform_SetMousePositionInScreen(
      );
00587     }
00588     /**
00589     * Set the position of the mouse cursor relative to screen co-ordinates
00590     */
00591     void SetMousePositionInScreen(unsigned int x, unsigned int y)
00592     {
00593         screenMousePosition.x = x;
00594         screenMousePosition.y = y;
00595 
00596         Platform_SetMousePositionInScreen(
      );
00597     }
00598 
00599     /**
00600     * Return the Resolution of the current screen
00601     */
00602     TinyWindow::uiVec2 GetScreenResolution(void)
00603     {
00604         uiVec2 resolution;
00605         Platform_GetScreenResolution(resolution
      );
00606         return resolution;
00607     }
00608 
00609     /**
00610      * Set the Size/Resolution of the given window
00611      */
00612     std::error_code SetWindowResolution(window_t* window,
      TinyWindow::uiVec2 resolution)
00613     {
00614         if (window != nullptr)
00615         {
00616             window->resolution.width = resolution.width;
00617             window->resolution.height = resolution.height;
00618 
00619             Platform_SetWindowResolution(window
      );
```

```
00620             return TinyWindow::error_t::success;
00621         }
00622         return TinyWindow::error_t::windowInvalid;
00623     }
00624     /**
00625      * Set the Size/Resolution of the given window
00626      */
00627     std::error_code SetWindowResolution(window_t* window, unsigned int
       width, unsigned int height)
00628     {
00629         if (window != nullptr)
00630         {
00631             window->resolution.width = width;
00632             window->resolution.height = height;
00633 
00634             Platform_SetWindowResolution(window
      );
00635             return TinyWindow::error_t::success;
00636         }
00637         return TinyWindow::error_t::windowInvalid;
00638     }
00639 
00640     /**
00641      * Set the Position of the given window relative to screen co-ordinates
00642      */
00643     std::error_code SetWindowPosition(window_t* window,
      TinyWindow::uiVec2 windowPosition)
00644     {
00645         if (window != nullptr)
00646         {
00647             window->position.x = windowPosition.x;
00648             window->position.y = windowPosition.y;
00649 
00650             Platform_SetWindowPosition(window, windowPosition.x
      , windowPosition.y);
00651             return TinyWindow::error_t::success;
00652         }
00653         return TinyWindow::error_t::windowInvalid;
00654     }
00655     /**
00656     * Set the Position of the given window relative to screen co-ordinates
00657     */
00658     std::error_code SetWindowPosition(window_t* window, unsigned int x,
      unsigned int y)
00659     {
00660         if (window != nullptr)
00661         {
00662             window->position.x = x;
00663             window->position.y = y;
00664 
00665             Platform_SetWindowPosition(window, x
      , y);
00666             return TinyWindow::error_t::success;
00667         }
00668         return TinyWindow::error_t::windowInvalid;
00669     }
00670 
00671     /**
00672      * Set the mouse Position of the given window's co-ordinates
00673      */
00674     std::error_code SetMousePositionInWindow(
      window_t* window, TinyWindow::uiVec2 mousePosition)
00675     {
00676         if (window != nullptr)
00677         {
00678             window->mousePosition.x = mousePosition.x;
00679             window->mousePosition.y = mousePosition.y;
00680 
00681             Platform_SetMousePositionInWindow
      (window, mousePosition.x, mousePosition.y);
00682             return TinyWindow::error_t::success;
00683         }
00684         return TinyWindow::error_t::windowInvalid;
00685     }
00686     /**
00687     * Set the mouse Position of the given window's co-ordinates
00688     */
00689     std::error_code SetMousePositionInWindow(
      window_t* window, unsigned int x, unsigned int y)
00690     {
00691         if (window != nullptr)
00692         {
00693             window->mousePosition.x = x;
00694             window->mousePosition.y = y;
00695 
00696             Platform_SetMousePositionInWindow
      (window, x, y);
```

```
00697             return TinyWindow::error_t::success;
00698         }
00699         return TinyWindow::error_t::windowInvalid;
00700     }
00701
00702     /**
00703      * Swap the draw buffers of the given window
00704      */
00705     inline std::error_code SwapWindowBuffers(window_t* window)
00706     {
00707         if (window != nullptr)
00708         {
00709 #if defined(TW_WINDOWS)
00710             SwapBuffers(window->deviceContextHandle);
00711 #elif defined(TW_LINUX)
00712             glXSwapBuffers(currentDisplay, window->windowHandle);
00713 #endif
00714             return TinyWindow::error_t::success;
00715         }
00716         return TinyWindow::error_t::windowInvalid;
00717     }
00718
00719     /**
00720      * Make the given window be the current OpenGL Context to be drawn to
00721      */
00722     std::error_code MakeWindowCurrentContext(
      window_t* window)
00723     {
00724         if (window != nullptr)
00725         {
00726 #if defined(TW_WINDOWS)
00727             wglMakeCurrent(window->deviceContextHandle,
00728                 window->glRenderingContextHandle);
00729 #elif defined(TW_LINUX)
00730             glXMakeCurrent(currentDisplay, window->windowHandle,
00731                 window->context);
00732 #endif
00733             return TinyWindow::error_t::success;
00734         }
00735         return TinyWindow::error_t::windowInvalid;
00736     }
00737
00738     /**
00739      * Toggle the given window's full screen mode
00740      */
00741     std::error_code SetFullScreen(window_t* window, bool newState)
00742     {
00743         if (window != nullptr)
00744         {
00745             window->currentState = (newState == true) ?
      state_t::fullscreen : state_t::normal;
00746
00747 #if defined(TW_WINDOWS)
00748             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
00749                 WS_SYSMENU | WS_POPUP | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_VISIBLE);
00750
00751             MoveWindow(window->windowHandle, 0, 0, windowManager
      ::GetScreenResolution().width,
00752                 windowManager::GetScreenResolution
      ().height, true);
00753 #elif defined(TW_LINUX)
00754             XEvent currentEvent;
00755             memset(&currentEvent, 0, sizeof(currentEvent));
00756
00757             currentEvent.xany.type = ClientMessage;
00758             currentEvent.xclient.message_type = AtomState;
00759             currentEvent.xclient.format = 32;
00760             currentEvent.xclient.window = window->windowHandle;
00761             currentEvent.xclient.data.l[0] = window->currentState == state_t::fullscreen;
00762             currentEvent.xclient.data.l[1] = AtomFullScreen;
00763
00764             XSendEvent(currentDisplay,
00765                 XDefaultRootWindow(currentDisplay),
00766                 0, SubstructureNotifyMask, &currentEvent);
00767 #endif
00768             return TinyWindow::error_t::success;
00769         }
00770         return TinyWindow::error_t::windowInvalid;
00771     }
00772
00773     /**
00774      * Toggle the minimization state of the given window
00775      */
00776     std::error_code MinimizeWindow(window_t* window, bool newState)
00777     {
00778         if (window != nullptr)
00779         {
```

```
00780             if (newState)
00781             {
00782                 window->currentState = state_t::
      minimized;
00783 
00784 #if defined(TW_WINDOWS)
00785                 ShowWindow(window->windowHandle, SW_MINIMIZE);
00786 #elif defined(TW_LINUX)
00787                 XIconifyWindow(currentDisplay,
00788                     window->windowHandle, 0);
00789 #endif
00790             }
00791 
00792             else
00793             {
00794                 window->currentState = state_t::
      normal;
00795 #if defined(TW_WINDOWS)
00796                 ShowWindow(window->windowHandle, SW_RESTORE);
00797 #elif defined(TW_LINUX)
00798                 XMapWindow(currentDisplay, window->windowHandle);
00799 #endif
00800             }
00801             return TinyWindow::error_t::success;
00802         }
00803         return TinyWindow::error_t::windowInvalid;
00804     }
00805 
00806     /**
00807      * Toggle the maximization state of the current window
00808      */
00809     std::error_code MaximizeWindow(window_t* window, bool newState)
00810     {
00811         if (window != nullptr)
00812         {
00813             if (newState)
00814             {
00815                 window->currentState = state_t::
      maximized;
00816 #if defined(TW_WINDOWS)
00817                 ShowWindow(window->windowHandle, SW_MAXIMIZE);
00818 #elif defined(TW_LINUX)
00819                 XEvent currentEvent;
00820                 memset(&currentEvent, 0, sizeof(currentEvent));
00821 
00822                 currentEvent.xany.type = ClientMessage;
00823                 currentEvent.xclient.message_type = AtomState;
00824                 currentEvent.xclient.format = 32;
00825                 currentEvent.xclient.window = window->windowHandle;
00826                 currentEvent.xclient.data.l[0] = (window->currentState == state_t::maximized);
00827                 currentEvent.xclient.data.l[1] = AtomMaxVert;
00828                 currentEvent.xclient.data.l[2] = AtomMaxHorz;
00829 
00830                 XSendEvent(currentDisplay,
00831                     XDefaultRootWindow(currentDisplay),
00832                     0, SubstructureNotifyMask, &currentEvent);
00833 #endif
00834             }
00835 
00836             else
00837             {
00838                 window->currentState = state_t::
      normal;
00839 #if defined(TW_WINDOWS)
00840                 ShowWindow(window->windowHandle, SW_RESTORE);
00841 #elif defined(TW_LINUX)
00842                 XEvent currentEvent;
00843                 memset(&currentEvent, 0, sizeof(currentEvent));
00844 
00845                 currentEvent.xany.type = ClientMessage;
00846                 currentEvent.xclient.message_type = AtomState;
00847                 currentEvent.xclient.format = 32;
00848                 currentEvent.xclient.window = window->windowHandle;
00849                 currentEvent.xclient.data.l[0] = (window->currentState == state_t::maximized);
00850                 currentEvent.xclient.data.l[1] = AtomMaxVert;
00851                 currentEvent.xclient.data.l[2] = AtomMaxHorz;
00852 
00853                 XSendEvent(currentDisplay,
00854                     XDefaultRootWindow(currentDisplay),
00855                     0, SubstructureNotifyMask, &currentEvent);
00856 #endif
00857             }
00858             return TinyWindow::error_t::success;
00859         }
00860         return TinyWindow::error_t::windowInvalid;
00861     }
00862 
```

```
00863     /**
00864      * Set the window title bar by name
00865      */
00866     std::error_code SetWindowTitleBar(window_t* window, const char*
      newTitle)
00867     {
00868         if (newTitle != nullptr)
00869         {
00870             if (window != nullptr)
00871             {
00872 #if defined(TW_WINDOWS)
00873                 SetWindowText(window->windowHandle, newTitle);
00874 #elif defined(TW_LINUX)
00875                 XStoreName(currentDisplay, window->windowHandle, newTitle);
00876 #endif
00877                 return TinyWindow::error_t::success;
00878             }
00879             return TinyWindow::error_t::windowInvalid;
00880         }
00881         return TinyWindow::error_t::invalidTitlebar;
00882     }
00883
00884     /**
00885     * Set the window icon by name (currently not functional)
00886     */
00887     std::error_code SetWindowIcon(void)//const char* windowName, const char* icon,
       unsigned int width, unsigned int height)
00888     {
00889         return TinyWindow::error_t::functionNotImplemented;
00890     }
00891
00892     /**
00893     * Set the window to be in focus by name
00894     */
00895     std::error_code FocusWindow(window_t* window, bool newState)
00896     {
00897         if (window != nullptr)
00898         {
00899             if (newState)
00900             {
00901 #if defined(TW_WINDOWS)
00902                 SetFocus(window->windowHandle);
00903 #elif defined(TW_LINUX)
00904                 XMapWindow(currentDisplay, window->windowHandle);
00905 #endif
00906             }
00907
00908             else
00909             {
00910 #if defined(_WIN32) || defined(_WIN64)
00911                 SetFocus(nullptr);
00912 #elif defined(TW_LINUX)
00913                 XUnmapWindow(currentDisplay, window->windowHandle);
00914 #endif
00915             }
00916             return TinyWindow::error_t::success;
00917         }
00918         return TinyWindow::error_t::windowInvalid;
00919     }
00920
00921     /**
00922      * Restore the window by name
00923      */
00924     std::error_code RestoreWindow(window_t* window)
00925     {
00926         if (window != nullptr)
00927         {
00928 #if defined(TW_WINDOWS)
00929             ShowWindow(window->windowHandle, SW_RESTORE);
00930 #elif defined(TW_LINUX)
00931             XMapWindow(currentDisplay, window->windowHandle);
00932 #endif
00933             return TinyWindow::error_t::success;
00934         }
00935         return TinyWindow::error_t::windowInvalid;
00936     }
00937
00938     /**
00939     * Ask the window manager to poll for events
00940     */
00941     inline void PollForEvents(void)
00942     {
00943 #if defined(TW_WINDOWS)
00944         //only process events if there are any to process
00945         if (PeekMessage(&winMessage, 0, 0, 0, PM_REMOVE))
00946         {
00947             TranslateMessage(&winMessage);
```

```
00948             DispatchMessage(&winMessage);
00949         }
00950 #elif defined(TW_LINUX)
00951         //if there are any events to process
00952         if (XEventsQueued(currentDisplay, QueuedAfterReading))
00953         {
00954             XNextEvent(currentDisplay, &currentEvent);
00955             Linux_ProcessEvents(currentEvent);
00956         }
00957 #endif
00958     }
00959 
00960     /**
00961     * Ask the window manager to wait for events
00962     */
00963     inline void WaitForEvents(void)
00964     {
00965 #if defined(TW_WINDOWS)
00966         //process even if there aren't any to process
00967         GetMessage(&winMessage, 0, 0, 0);
00968         TranslateMessage(&winMessage);
00969         DispatchMessage(&winMessage);
00970 #elif defined(TW_LINUX)
00971         //even if there aren't any events to process
00972         XNextEvent(currentDisplay, &currentEvent);
00973         Linux_ProcessEvents(currentEvent);
00974 #endif
00975     }
00976 
00977     /**
00978     * Remove window from the manager by name
00979     */
00980     std::error_code RemoveWindow(window_t* window)
00981     {
00982         if (window != nullptr)
00983         {
00984             ShutdownWindow(window);
00985             return TinyWindow::error_t::success;
00986         }
00987         return TinyWindow::error_t::windowInvalid;
00988     }
00989 
00990     /**
00991     * Set the window style preset by name
00992     */
00993     std::error_code SetWindowStyle(window_t* window,
      style_t windowStyle)
00994     {
00995         if (window != nullptr)
00996         {
00997 #if defined(TW_WINDOWS)
00998             switch (windowStyle)
00999             {
01000             case style_t::normal:
01001             {
01002                 EnableWindowDecorators(window,
      titleBar | border |
01003                     closeButton | minimizeButton |
      maximizeButton);
01004                 break;
01005             }
01006 
01007             case style_t::popup:
01008             {
01009                 EnableWindowDecorators(window, 0);
01010                 break;
01011             }
01012 
01013             case style_t::bare:
01014             {
01015                 EnableWindowDecorators(window,
      titleBar | border);
01016                 break;
01017             }
01018 
01019             default:
01020             {
01021                 return TinyWindow::error_t::invalidWindowStyle;
01022             }
01023             }
01024 
01025 #elif defined(TW_LINUX)
01026             switch (windowStyle)
01027             {
01028             case style_t::normal:
01029             {
01030                 window->decorators = (1L << 2);
```

```
01031                 window->currentWindowStyle = linuxMove | linuxClose |
01032                     linuxMaximize | linuxMinimize;
01033                 long Hints[5] = { hint_t::function | hint_t::decorator, window->currentWindowStyle,
      window->decorators, 0, 0 };
01034
01035                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01036                     (unsigned char*)Hints, 5);
01037
01038                 XMapWindow(currentDisplay, window->windowHandle);
01039                 break;
01040             }
01041
01042             case style_t::bare:
01043             {
01044                 window->decorators = (1L << 2);
01045                 window->currentWindowStyle = (1L << 2);
01046                 long Hints[5] = { function | decorator, window->currentWindowStyle, window->decorators
      , 0, 0 };
01047
01048                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01049                     (unsigned char*)Hints, 5);
01050
01051                 XMapWindow(currentDisplay, window->windowHandle);
01052                 break;
01053             }
01054
01055             case style_t::popup:
01056             {
01057                 window->decorators = 0;
01058                 window->currentWindowStyle = (1L << 2);
01059                 long Hints[5] = { function | decorator, window->currentWindowStyle, window->decorators
      , 0, 0 };
01060
01061                 XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
      PropModeReplace,
01062                     (unsigned char*)Hints, 5);
01063
01064                 XMapWindow(currentDisplay, window->windowHandle);
01065                 break;
01066             }
01067
01068             default:
01069             {
01070                 return TinyWindow::error_t::invalidWindowStyle;
01071             }
01072             }
01073 #endif
01074             return TinyWindow::error_t::success;
01075         }
01076         return TinyWindow::error_t::windowInvalid;
01077     }
01078
01079     /**
01080     * Enable window decorators by name
01081     */
01082     std::error_code EnableWindowDecorators(
      window_t* window, unsigned int decorators)
01083     {
01084         if (window != nullptr)
01085         {
01086 #if defined(TW_WINDOWS)
01087             window->currentWindowStyle = WS_VISIBLE | WS_CLIPSIBLINGS;
01088
01089             if (decorators & border)
01090             {
01091                 window->currentWindowStyle |= WS_BORDER;
01092             }
01093
01094             if (decorators & titleBar)
01095             {
01096                 window->currentWindowStyle |= WS_CAPTION;
01097             }
01098
01099             if (decorators & icon)
01100             {
01101                 window->currentWindowStyle |= WS_ICONIC;
01102             }
01103
01104             if (decorators & closeButton)
01105             {
01106                 window->currentWindowStyle |= WS_SYSMENU;
01107             }
01108
01109             if (decorators & minimizeButton)
01110             {
```

```
01111                 window->currentWindowStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
01112             }
01113
01114             if (decorators & maximizeButton)
01115             {
01116                 window->currentWindowStyle |= WS_MAXIMIZEBOX | WS_SYSMENU;
01117             }
01118
01119             if (decorators & sizeableBorder)
01120             {
01121                 window->currentWindowStyle |= WS_SIZEBOX;
01122             }
01123
01124             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
01125                 window->currentWindowStyle);
01126 #elif defined(TW_LINUX)
01127             if (decorators & closeButton)
01128             {
01129                 window->currentWindowStyle |= linuxClose;
01130                 window->decorators = 1;
01131             }
01132
01133             if (decorators & minimizeButton)
01134             {
01135                 window->currentWindowStyle |= linuxMinimize;
01136                 window->decorators = 1;
01137             }
01138
01139             if (decorators & maximizeButton)
01140             {
01141                 window->currentWindowStyle |= linuxMaximize;
01142                 window->decorators = 1;
01143             }
01144
01145             if (decorators & icon)
01146             {
01147                 //Linux (at least cinnamon) does not have icons in the window. only in the task bar
       icon
01148             }
01149
01150             //just need to set it to 1 to enable all decorators that include title bar
01151             if (decorators & titleBar)
01152             {
01153                 window->decorators = 1;
01154             }
01155
01156             if (decorators & border)
01157             {
01158                 window->decorators = 1;
01159             }
01160
01161             if (decorators & sizeableBorder)
01162             {
01163                 window->decorators = 1;
01164             }
01165
01166             long hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0,
       0 };
01167
01168             XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
01169                 PropModeReplace, (unsigned char*)hints, 5);
01170
01171             XMapWindow(currentDisplay, window->windowHandle);
01172 #endif
01173             return TinyWindow::error_t::success;
01174         }
01175         return TinyWindow::error_t::windowInvalid;
01176     }
01177
01178     /**
01179     * Disable windows decorators by name
01180     */
01181     std::error_code DisableWindowDecorators(
      window_t* window, unsigned int decorators)
01182     {
01183         if (window != nullptr)
01184         {
01185 #if defined(TW_WINDOWS)
01186             if (decorators & border)
01187             {
01188                 window->currentWindowStyle &= ~WS_BORDER;
01189             }
01190
01191             if (decorators & titleBar)
01192             {
01193                 window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
01194             }
```

```
01195
01196             if (decorators & icon)
01197             {
01198                 window->currentWindowStyle &= ~WS_ICONIC;
01199             }
01200
01201             if (decorators & closeButton)
01202             {
01203                 window->currentWindowStyle &= ~WS_SYSMENU;
01204             }
01205
01206             if (decorators & minimizeButton)
01207             {
01208                 window->currentWindowStyle &= ~WS_MINIMIZEBOX;
01209             }
01210
01211             if (decorators & maximizeButton)
01212             {
01213                 window->currentWindowStyle &= ~WS_MAXIMIZEBOX;
01214             }
01215
01216             if (decorators & sizeableBorder)
01217             {
01218                 window->currentWindowStyle &= ~WS_SIZEBOX;
01219             }
01220
01221             SetWindowLongPtr(window->windowHandle, GWL_STYLE,
01222                 window->currentWindowStyle | WS_VISIBLE);
01223 #elif defined(TW_LINUX)
01224             if (decorators & closeButton)
01225             {
01226                 //I hate doing this but it is necessary to keep functionality going.
01227                 bool minimizeEnabled = false;
01228                 bool maximizeEnabled = false;
01229
01230                 if (decorators & maximizeButton)
01231                 {
01232                     maximizeEnabled = true;
01233                 }
01234
01235                 if (decorators & minimizeButton)
01236                 {
01237                     minimizeEnabled = true;
01238                 }
01239
01240                 window->currentWindowStyle &= ~linuxClose;
01241
01242                 if (maximizeEnabled)
01243                 {
01244                     window->currentWindowStyle |= linuxMaximize;
01245                 }
01246
01247                 if (minimizeEnabled)
01248                 {
01249                     window->currentWindowStyle |= linuxMinimize;
01250                 }
01251
01252                 window->decorators = 1;
01253             }
01254
01255             if (decorators & minimizeButton)
01256             {
01257                 window->currentWindowStyle &= ~linuxMinimize;
01258                 window->decorators = 1;
01259             }
01260
01261             if (decorators & maximizeButton)
01262             {
01263                 bool minimizeEnabled = false;
01264
01265                 if (decorators & minimizeButton)
01266                 {
01267                     minimizeEnabled = true;
01268                 }
01269
01270                 window->currentWindowStyle &= ~linuxMaximize;
01271
01272                 if (minimizeEnabled)
01273                 {
01274                     window->currentWindowStyle |= linuxMinimize;
01275                 }
01276
01277                 window->decorators = 1;
01278             }
01279
01280             if (decorators & icon)
01281             {
```

```
01282                 //Linux (at least cinnamon) does not have icons in the window. only in the taskb ar
      icon
01283             }
01284
01285             //just need to set it to 1 to enable all decorators that include title bar
01286             if (decorators & titleBar)
01287             {
01288                 window->decorators = linuxBorder;
01289             }
01290
01291             if (decorators & border)
01292             {
01293                 window->decorators = 0;
01294             }
01295
01296             if (decorators & sizeableBorder)
01297             {
01298                 window->decorators = 0;
01299             }
01300
01301             long hints[5] = { function | decorator, window->currentWindowStyle, window->decorators, 0,
      0 };
01302
01303             XChangeProperty(currentDisplay, window->windowHandle, AtomHints, XA_ATOM, 32,
01304                 PropModeReplace, (unsigned char*)hints, 5);
01305
01306             XMapWindow(currentDisplay, window->windowHandle);
01307 #endif
01308             return TinyWindow::error_t::success;
01309         }
01310         return TinyWindow::error_t::windowInvalid;
01311     }
01312
01313 private:
01314
01315     std::vector< std::unique_ptr<window_t> >
      windowList;
01316
01317     TinyWindow::uiVec2
      screenResolution;
01318     TinyWindow::uiVec2
      screenMousePosition;
01319
01320     void Platform_InitializeWindow(window_t* window)
01321     {
01322 #if defined(TW_WINDOWS)
01323         Windows_InitializeWindow(window);
01324 #elif defined(TW_LINUX)
01325         Linux_InitializeWindow(window);
01326 #endif
01327     }
01328
01329     std::error_code Platform_InitializeGL(
      window_t* window)
01330     {
01331 #if defined(TW_WINDOWS)
01332         window->deviceContextHandle = GetDC(window->windowHandle);
01333         InitializePixelFormat(window);
01334         window->glRenderingContextHandle = wglCreateContext(window->deviceContextHandle);
01335         wglMakeCurrent(window->deviceContextHandle, window->glRenderingContextHandle);
01336
01337         window->contextCreated = (window->glRenderingContextHandle != nullptr);
01338
01339         if (window->contextCreated)
01340         {
01341             return TinyWindow::error_t::success;
01342         }
01343
01344         return TinyWindow::error_t::invalidContext;
01345 #elif defined(TW_LINUX)
01346         if (!window->context)
01347         {
01348             window->context = glXCreateContext(
01349                 currentDisplay,
01350                 window->visualInfo,
01351                 0,
01352                 true);
01353
01354             if (window->context)
01355             {
01356                 glXMakeCurrent(currentDisplay,
01357                     window->windowHandle,
01358                     window->context);
01359
01360                 XWindowAttributes l_Attributes;
01361
01362                 XGetWindowAttributes(currentDisplay,
```

```
01363                   window->windowHandle, &l_Attributes);
01364               window->position.x = l_Attributes.x;
01365               window->position.y = l_Attributes.y;
01366
01367               window->contextCreated = true;
01368               InitializeAtoms();
01369               return TinyWindow::error_t::success;
01370           }
01371           return TinyWindow::error_t::linuxCannotConnectXServer;
01372       }
01373
01374       else
01375       {
01376           return TinyWindow::error_t::existingContext;
01377       }
01378       return TinyWindow::error_t::existingContext;
01379 #endif
01380   }
01381
01382   void Platform_SetMousePositionInScreen()
01383   {
01384 #if defined(TW_WINDOWS)
01385       SetCursorPos(screenMousePosition.y,
      screenMousePosition.y);
01386 #elif defined(TW_LINUX)
01387       XWarpPointer(currentDisplay, None,
01388           XDefaultRootWindow(currentDisplay), 0, 0,
01389           screenResolution.x,
01390           screenResolution.y,
01391           screenMousePosition.x, screenMousePosition.y);
01392 #endif
01393   }
01394
01395   void Platform_GetScreenResolution(
      uiVec2 resolution)
01396   {
01397 #if defined(TW_WINDOWS)
01398       RECT screen;
01399       HWND desktop = GetDesktopWindow();
01400       GetWindowRect(desktop, &screen);
01401       resolution.width = screen.right;
01402       resolution.height = screen.bottom;
01403 #elif defined(TW_LINUX)
01404       resolution.width = WidthOfScreen(XDefaultScreenOfDisplay(currentDisplay));
01405       resolution.height = HeightOfScreen(XDefaultScreenOfDisplay(currentDisplay));
01406
01407       screenResolution.x = resolution.width;
01408       screenResolution.y = resolution.height;
01409 #endif
01410   }
01411
01412   void Platform_SetWindowResolution(
      window_t* window)
01413   {
01414 #if defined(TW_WINDOWS)
01415       SetWindowPos(window->windowHandle, HWND_TOP,
01416           window->position.x, window->position.y,
01417           window->resolution.x, window->resolution.y,
01418           SWP_SHOWWINDOW | SWP_NOMOVE);
01419 #elif defined(TW_LINUX)
01420       XResizeWindow(currentDisplay,
01421           window->windowHandle, window->resolution.x, window->resolution.y);
01422 #endif
01423   }
01424
01425   void Platform_SetWindowPosition(window_t* window, unsigned
      int x, unsigned int y)
01426   {
01427 #if defined(TW_WINDOWS)
01428       SetWindowPos(window->windowHandle, HWND_TOP, x, y,
01429           window->resolution.x, window->resolution.y,
01430           SWP_SHOWWINDOW | SWP_NOSIZE);
01431 #elif defined(TW_LINUX)
01432       XWindowChanges windowChanges;
01433
01434       windowChanges.x = x;
01435       windowChanges.y = y;
01436
01437       XConfigureWindow(
01438           currentDisplay,
01439           window->windowHandle, CWX | CWY, &windowChanges);
01440 #endif
01441   }
01442
01443   void Platform_SetMousePositionInWindow(
      window_t* window, unsigned int x, unsigned int y)
01444   {
```

```
01445 #if defined(TW_WINDOWS)
01446         POINT mousePoint;
01447         mousePoint.x = x;
01448         mousePoint.y = y;
01449         ScreenToClient(window->windowHandle, &mousePoint);
01450         SetCursorPos(mousePoint.x, mousePoint.y);
01451 #elif defined(TW_LINUX)
01452         XWarpPointer(
01453             currentDisplay,
01454             window->windowHandle, window->windowHandle,
01455             window->position.x, window->position.y,
01456             window->resolution.width, window->resolution.height,
01457             x, y);
01458 #endif
01459     }
01460
01461     void ShutdownWindow(window_t* window)
01462     {
01463 #if defined(TW_WINDOWS)
01464         if (window->glRenderingContextHandle)
01465         {
01466             wglMakeCurrent(nullptr, nullptr);
01467             wglDeleteContext(window->glRenderingContextHandle);
01468         }
01469
01470         if (window->paletteHandle)
01471         {
01472             DeleteObject(window->paletteHandle);
01473         }
01474         ReleaseDC(window->windowHandle, window->deviceContextHandle);
01475         UnregisterClass(window->name, window->instanceHandle);
01476
01477         FreeModule(window->instanceHandle);
01478
01479         window->deviceContextHandle = nullptr;
01480         window->windowHandle = nullptr;
01481         window->glRenderingContextHandle = nullptr;
01482
01483         if (windowList.size() > 1)
01484         {
01485             windowList.erase(windowList.begin() + window->iD);
01486         }
01487
01488         else
01489         {
01490             windowList.erase(windowList.begin());
01491         }
01492 #elif defined(TW_LINUX)
01493         if (window->currentState == state_t::fullscreen)
01494         {
01495             RestoreWindow(window);
01496         }
01497
01498         glXDestroyContext(currentDisplay, window->context);
01499         XUnmapWindow(currentDisplay, window->windowHandle);
01500         XDestroyWindow(currentDisplay, window->windowHandle);
01501         window->windowHandle = 0;
01502         window->context = 0;
01503 #endif
01504     }
01505
01506 #if defined(TW_WINDOWS)
01507
01508     enum keyLong_t
01509     {
01510         leftControlDown = 29,
01511         rightControlDown = 285,
01512         leftShiftDown = 42,
01513         rightShiftDown = 54,
01514         leftAltDown = 8248,
01515         rightAltDown = 8504,
01516
01517         leftControlUp = 49181,
01518         rightControlUp = 49437,
01519         leftShiftUp = 49194,
01520         rightShiftUp = 49206,
01521         leftAltUp = 49208,
01522         rightAltUp = 49464,
01523     };
01524
01525     MSG     winMessage;
01526     HDC     deviceContextHandle;
01527
01528     //the window procedure for all windows. This is used mainly to handle window events
01529     static LRESULT CALLBACK WindowProcedure(HWND windowHandle, unsigned int winMessage, WPARAM
      wordParam, LPARAM longParam)
01530     {
```

```
01531         windowManager* manager = (windowManager*)GetWindowLongPtr(
      windowHandle, GWLP_USERDATA);
01532         window_t* window = nullptr;
01533         if (manager != nullptr)
01534         {
01535             window = manager->GetWindowByHandle(windowHandle);
01536         }
01537 
01538             switch (winMessage)
01539             {
01540             case WM_DESTROY:
01541             {
01542                 if (manager != nullptr)
01543                 {
01544                     window->shouldClose = true;
01545 
01546                     if (window->destroyedEvent != nullptr)
01547                     {
01548                         window->destroyedEvent();
01549                     }
01550 
01551                     manager->ShutdownWindow(window);
01552                 }
01553                 break;
01554             }
01555             case WM_MOVE:
01556             {
01557                 window->position.x = LOWORD(longParam);
01558                 window->position.y = HIWORD(longParam);
01559 
01560                 if (window->movedEvent != nullptr)
01561                 {
01562                     window->movedEvent(window->position.x, window->position.y);
01563                 }
01564 
01565                 break;
01566             }
01567 
01568             case WM_MOVING:
01569             {
01570                 window->position.x = LOWORD(longParam);
01571                 window->position.y = HIWORD(longParam);
01572 
01573                 if (window->movedEvent != nullptr)
01574                 {
01575                     window->movedEvent(window->position.x, window->position.y);
01576                 }
01577                 break;
01578             }
01579 
01580             case WM_SIZE:
01581             {
01582                 window->resolution.width = (unsigned int)LOWORD(longParam);
01583                 window->resolution.height = (unsigned int)HIWORD(longParam);
01584 
01585                 switch (wordParam)
01586                 {
01587                 case SIZE_MAXIMIZED:
01588                 {
01589                     if (window->maximizedEvent != nullptr)
01590                     {
01591                         window->maximizedEvent();
01592                     }
01593 
01594                     break;
01595                 }
01596 
01597                 case SIZE_MINIMIZED:
01598                 {
01599                     if (window->minimizedEvent != nullptr)
01600                     {
01601                         window->minimizedEvent();
01602                     }
01603                     break;
01604                 }
01605 
01606                 default:
01607                 {
01608                     if (window->resizeEvent != nullptr)
01609                     {
01610                         window->resizeEvent(window->resolution.width,
01611                             window->resolution.height);
01612                     }
01613                     break;
01614                 }
01615                 }
01616                 break;
```

```
01617             }
01618
01619             case WM_SIZING:
01620             {
01621                 window->resolution.width = (unsigned int)LOWORD(longParam);
01622                 window->resolution.height = (unsigned int)HIWORD(longParam);
01623
01624                 if (window->resizeEvent != nullptr)
01625                 {
01626                     window->resizeEvent(window->resolution.width,
01627                         window->resolution.height);
01628                 }
01629                 break;
01630             }
01631
01632             case WM_KEYDOWN:
01633             {
01634                 unsigned int translatedKey = 0;
01635
01636                 switch (HIWORD(longParam))
01637                 {
01638                 case leftControlDown:
01639                 {
01640                     window->keys[leftControl] = keyState_t
      ::down;
01641                     translatedKey = leftControl;
01642                     break;
01643                 }
01644
01645                 case rightControlDown:
01646                 {
01647                     window->keys[rightControl] =
      keyState_t::down;
01648                     translatedKey = rightControl;
01649                     break;
01650                 }
01651
01652                 case leftShiftDown:
01653                 {
01654                     window->keys[leftShift] = keyState_t
      ::down;
01655                     translatedKey = leftShift;
01656                     break;
01657                 }
01658
01659                 case rightShiftDown:
01660                 {
01661                     window->keys[rightShift] = keyState_t
      ::down;
01662                     translatedKey = rightShift;
01663                     break;
01664                 }
01665
01666                 default:
01667                 {
01668                     translatedKey = Windows_TranslateKey(wordParam);
01669                     window->keys[translatedKey] = keyState_t
      ::down;
01670                     break;
01671                 }
01672                 }
01673
01674                 if (window->keyEvent != nullptr)
01675                 {
01676                     window->keyEvent(translatedKey, keyState_t::down);
01677                 }
01678                 break;
01679             }
01680
01681             case WM_KEYUP:
01682             {
01683                 unsigned int translatedKey = 0;
01684
01685                 switch (HIWORD(longParam))
01686                 {
01687                 case leftControlUp:
01688                 {
01689                     window->keys[leftControl] = keyState_t
      ::up;
01690                     translatedKey = leftControl;
01691                     break;
01692                 }
01693
01694                 case rightControlUp:
01695                 {
01696                     window->keys[rightControl] =
      keyState_t::up;
```

```
01697                   translatedKey = rightControl;
01698                   break;
01699               }
01700
01701               case leftShiftUp:
01702               {
01703                   window->keys[leftShift] = keyState_t
      ::up;
01704                   translatedKey = leftShift;
01705                   break;
01706               }
01707
01708               case rightShiftUp:
01709               {
01710                   window->keys[rightShift] = keyState_t
      ::up;
01711                   translatedKey = rightShift;
01712                   break;
01713               }
01714
01715               default:
01716               {
01717                   translatedKey = Windows_TranslateKey(wordParam);
01718                   window->keys[translatedKey] = keyState_t
      ::up;
01719                   break;
01720               }
01721               }
01722
01723               if (window->keyEvent != nullptr)
01724               {
01725                   window->keyEvent(translatedKey, keyState_t::up);
01726               }
01727               break;
01728           }
01729
01730           case WM_SYSKEYDOWN:
01731           {
01732               unsigned int translatedKey = 0;
01733               switch (HIWORD(longParam))
01734               {
01735               case leftAltDown:
01736               {
01737                   window->keys[leftAlt] = keyState_t
      ::down;
01738                   translatedKey = leftAlt;
01739                   break;
01740               }
01741
01742
01743               case rightAltDown:
01744               {
01745                   window->keys[rightAlt] = keyState_t
      ::down;
01746                   translatedKey = rightAlt;
01747               }
01748
01749               default:
01750               {
01751                   break;
01752               }
01753               }
01754
01755               if (window->keyEvent != nullptr)
01756               {
01757                   window->keyEvent(translatedKey, keyState_t::down);
01758               }
01759
01760               break;
01761           }
01762
01763           case WM_SYSKEYUP:
01764           {
01765               unsigned int translatedKey = 0;
01766               switch (HIWORD(longParam))
01767               {
01768               case leftAltUp:
01769               {
01770                   window->keys[leftAlt] = keyState_t
      ::up;
01771                   translatedKey = leftAlt;
01772                   break;
01773               }
01774
01775
01776               case rightAltUp:
01777               {
```

```
01778                       window->keys[rightAlt] = keyState_t
      ::up;
01779                       translatedKey = rightAlt;
01780                       break;
01781                   }
01782
01783                   default:
01784                   {
01785                       break;
01786                   }
01787                   }
01788
01789                   if (window->keyEvent != nullptr)
01790                   {
01791                       window->keyEvent(translatedKey, keyState_t::up);
01792                   }
01793                   break;
01794               }
01795
01796               //WM_KEYUP/DOWN cannot tell between uppercase and lowercase.
01797               /*case WM_CHAR:
01798               {
01799                   int keyDown = longParam & 0x31;
01800                   if (keyDown == 1)
01801                   {
01802                       window->keys[wordParam] = tinyWindowKeyState_t::DOWN;
01803                   }
01804
01805                   else if (keyDown == 0)
01806                   {
01807                       window->keys[wordParam] = tinyWindowKeyState_t::UP;
01808                   }
01809
01810                   if (window->keyEvent != nullptr)
01811                   {
01812                       window->keyEvent(wordParam, (tinyWindowKeyState_t)keyDown);
01813                   }
01814               }*/
01815
01816               case WM_MOUSEMOVE:
01817               {
01818                   window->mousePosition.x = (unsigned int)LOWORD(longParam);
01819                   window->mousePosition.y = (unsigned int)HIWORD(longParam);
01820
01821                   POINT point;
01822                   point.x = (LONG)window->mousePosition.x;
01823                   point.y = (LONG)window->mousePosition.y;
01824
01825                   ClientToScreen(windowHandle, &point);
01826
01827                   if (window->mouseMoveEvent != nullptr)
01828                   {
01829                       window->mouseMoveEvent(window->mousePosition.x,
01830                           window->mousePosition.y, point.x, point.y);
01831                   }
01832                   break;
01833               }
01834
01835               case WM_LBUTTONDOWN:
01836               {
01837                   window->mouseButton[(unsigned int)
      mouseButton_t::left] = buttonState_t::down;
01838
01839                   if (window->mouseButtonEvent != nullptr)
01840                   {
01841                       window->mouseButtonEvent(mouseButton_t::left, buttonState_t::down);
01842                   }
01843                   break;
01844               }
01845
01846               case WM_LBUTTONUP:
01847               {
01848                   window->mouseButton[(unsigned int)
      mouseButton_t::left] = buttonState_t::up;
01849
01850                   if (window->mouseButtonEvent != nullptr)
01851                   {
01852                       window->mouseButtonEvent(mouseButton_t::left, buttonState_t::up);
01853                   }
01854                   break;
01855               }
01856
01857               case WM_RBUTTONDOWN:
01858               {
01859                   window->mouseButton[(unsigned int)
      mouseButton_t::right] = buttonState_t::down;
01860
```

```
01861                 if (window->mouseButtonEvent != nullptr)
01862                 {
01863                     window->mouseButtonEvent(mouseButton_t::right, buttonState_t::down);
01864                 }
01865                 break;
01866             }
01867 
01868             case WM_RBUTTONUP:
01869             {
01870                 window->mouseButton[(unsigned int)
      mouseButton_t::right] = buttonState_t::up;
01871 
01872                 if (window->mouseButtonEvent != nullptr)
01873                 {
01874                     window->mouseButtonEvent(mouseButton_t::right, buttonState_t::up);
01875                 }
01876                 break;
01877             }
01878 
01879             case WM_MBUTTONDOWN:
01880             {
01881                 window->mouseButton[(unsigned int)
      mouseButton_t::middle] = buttonState_t::down;
01882 
01883                 if (window->mouseButtonEvent != nullptr)
01884                 {
01885                     window->mouseButtonEvent(mouseButton_t::middle, buttonState_t::down);
01886                 }
01887                 break;
01888             }
01889 
01890             case WM_MBUTTONUP:
01891             {
01892                 window->mouseButton[(unsigned int)
      mouseButton_t::middle] = buttonState_t::up;
01893 
01894                 if (window->mouseButtonEvent != nullptr)
01895                 {
01896                     window->mouseButtonEvent(mouseButton_t::middle, buttonState_t::up);
01897                 }
01898                 break;
01899             }
01900 
01901             case WM_MOUSEWHEEL:
01902             {
01903                 if ((wordParam % WHEEL_DELTA) > 0)
01904                 {
01905                     if (window->mouseWheelEvent != nullptr)
01906                     {
01907                         window->mouseWheelEvent(mouseScroll_t::down);
01908                     }
01909                 }
01910 
01911                 else
01912                 {
01913                     if (window->mouseWheelEvent != nullptr)
01914                     {
01915                         window->mouseWheelEvent(mouseScroll_t::up);
01916                     }
01917 
01918                 }
01919                 break;
01920             }
01921 
01922             default:
01923             {
01924                 //windowList[getWindow]
01925                 return DefWindowProc(windowHandle, winMessage, wordParam, longParam);
01926             }
01927             }
01928             return 0;
01929 
01930     }
01931 
01932     //get the window that is associated with this Win32 window handle
01933     window_t* GetWindowByHandle(HWND windowHandle)
01934     {
01935         for (unsigned int iter = 0; iter < windowList.size(); iter++)
01936         {
01937             if (windowList[iter]->windowHandle == windowHandle)
01938             {
01939                 return windowList[iter].get();
01940             }
01941         }
01942         return nullptr;
01943     }
01944 
```

```
01945     //initialize the given window using Win32
01946     void Windows_InitializeWindow(window_t* window,
01947         UINT style = CS_OWNDC | CS_HREDRAW | CS_DROPSHADOW,
01948         int clearScreenExtra = 0,
01949         int windowExtra = 0,
01950         HINSTANCE winInstance = GetModuleHandle(0),
01951         HICON icon = LoadIcon(0, IDI_APPLICATION),
01952         HCURSOR cursor = LoadCursor(0, IDC_ARROW),
01953         HBRUSH brush = (HBRUSH)BLACK_BRUSH)
01954     {
01955         window->instanceHandle = winInstance;
01956         window->windowClass.style = style;
01957         window->windowClass.lpfnWndProc = windowManager::WindowProcedure;
01958         window->windowClass.cbClsExtra = clearScreenExtra;
01959         window->windowClass.cbWndExtra = windowExtra;
01960         window->windowClass.hInstance = window->instanceHandle;
01961         window->windowClass.hIcon = icon;
01962         window->windowClass.hCursor = cursor;
01963         window->windowClass.hbrBackground = brush;
01964         window->windowClass.lpszMenuName = window->name;
01965         window->windowClass.lpszClassName = window->name;
01966         RegisterClass(&window->windowClass);
01967
01968         window->windowHandle =
01969             CreateWindow(window->name, window->name, WS_OVERLAPPEDWINDOW, 0,
01970             0, window->resolution.width,
01971             window->resolution.height,
01972             0, 0, 0, 0);
01973
01974         SetWindowLongPtr(window->windowHandle, GWLP_USERDATA, (long)this);
01975
01976         Platform_InitializeGL(window);
01977
01978         ShowWindow(window->windowHandle, true);
01979         UpdateWindow(window->windowHandle);
01980     }
01981
01982     //initialize the pixel format for the selected window
01983     void InitializePixelFormat(window_t* window)
01984     {
01985         window->pixelFormatDescriptor = {
01986             sizeof(PIXELFORMATDESCRIPTOR), /* size */
01987             1, /* version */
01988             PFD_SUPPORT_OPENGL |
01989             PFD_DRAW_TO_WINDOW |
01990             PFD_DOUBLEBUFFER, /* support double-buffering */
01991             PFD_TYPE_RGBA, /* color type */
01992             (BYTE)window->colorBits, 0, /* preferred color depth */
01993             0, 0,
01994             0, 0,
01995             0, 0,
01996             0, /* color bits (ignored) */ /* no alpha buffer */ /* alpha bits (ignored) */
01997             0, /* no accumulation buffer */
01998             0, 0, 0, 0, /* accum bits (ignored) */
01999             (BYTE)window->depthBits, /* depth buffer */
02000             (BYTE)window->stencilBits, /* no stencil buffer */
02001             0, /* no auxiliary buffers */
02002             PFD_MAIN_PLANE, /* main layer */
02003             0, /* reserved */
02004             0, 0, 0, /* no layer, visible, damage masks */
02005         };
02006
02007         int LocalPixelFormat = ChoosePixelFormat(window->deviceContextHandle,
02008             &window->pixelFormatDescriptor);
02009
02010         if (LocalPixelFormat)
02011         {
02012             SetPixelFormat(window->deviceContextHandle, LocalPixelFormat,
02013                 &window->pixelFormatDescriptor);
02014             return;
02015         }
02016         return;
02017     }
02018
02019     void Windows_Shutown(void)
02020     {
02021
02022     }
02023
02024     void CreateTerminal(void)
02025     {
02026         int conHandle;
02027         long stdHandle;
02028         FILE* fp;
02029
02030         // allocate a console for this app
02031         AllocConsole();
```

```
02032
02033         // redirect unbuffered STDOUT to the console
02034         stdHandle = (long)GetStdHandle(STD_OUTPUT_HANDLE);
02035         conHandle = _open_osfhandle(stdHandle, _O_TEXT);
02036         fp = _fdopen(conHandle, "w");
02037         *stdout = *fp;
02038
02039         setvbuf(stdout, nullptr, _IONBF, 0);
02040     }
02041
02042     static unsigned int Windows_TranslateKey(WPARAM wordParam)
02043     {
02044         switch (wordParam)
02045         {
02046             case VK_ESCAPE:
02047             {
02048                 return escape;
02049             }
02050
02051             case VK_F1:
02052             {
02053                 return F1;
02054             }
02055
02056             case VK_F2:
02057             {
02058                 return F2;
02059             }
02060
02061             case VK_F3:
02062             {
02063                 return F3;
02064             }
02065
02066             case VK_F4:
02067             {
02068                 return F4;
02069             }
02070
02071             case VK_F5:
02072             {
02073                 return F5;
02074             }
02075
02076             case VK_F6:
02077             {
02078                 return F6;
02079             }
02080
02081             case VK_F7:
02082             {
02083                 return F7;
02084             }
02085
02086             case VK_F8:
02087             {
02088                 return F8;
02089             }
02090
02091             case VK_F9:
02092             {
02093                 return F9;
02094             }
02095
02096             case VK_F10:
02097             {
02098                 return F10;
02099             }
02100
02101             case VK_F11:
02102             {
02103                 return F11;
02104             }
02105
02106             case VK_F12:
02107             {
02108                 return F12;
02109             }
02110
02111             case VK_BACK:
02112             {
02113                 return backspace;
02114             }
02115
02116             case VK_TAB:
02117             {
02118                 return tab;
```

```
02119             }
02120
02121             case VK_CAPITAL:
02122             {
02123                 return capsLock;
02124             }
02125
02126             case VK_RETURN:
02127             {
02128                 return enter;
02129             }
02130
02131             case VK_PRINT:
02132             {
02133                 return printScreen;
02134             }
02135
02136             case VK_SCROLL:
02137             {
02138                 return scrollLock;
02139             }
02140
02141             case VK_PAUSE:
02142             {
02143                 return pause;
02144             }
02145
02146             case VK_INSERT:
02147             {
02148                 return insert;
02149             }
02150
02151             case VK_HOME:
02152             {
02153                 return home;
02154             }
02155
02156             case VK_DELETE:
02157             {
02158                 return del;
02159             }
02160
02161             case VK_END:
02162             {
02163                 return end;
02164             }
02165
02166             case VK_PRIOR:
02167             {
02168                 return pageUp;
02169             }
02170
02171             case VK_NEXT:
02172             {
02173                 return pageDown;
02174             }
02175
02176             case VK_DOWN:
02177             {
02178                 return arrowDown;
02179             }
02180
02181             case VK_UP:
02182             {
02183                 return arrowUp;
02184             }
02185
02186             case VK_LEFT:
02187             {
02188                 return arrowLeft;
02189             }
02190
02191             case VK_RIGHT:
02192             {
02193                 return arrowRight;
02194             }
02195
02196             case VK_DIVIDE:
02197             {
02198                 return keypadDivide;
02199             }
02200
02201             case VK_MULTIPLY:
02202             {
02203                 return keypadMultiply;
02204             }
02205
```

```
02206             case VK_SUBTRACT:
02207             {
02208                 return keypadDivide;
02209             }
02210 
02211             case VK_ADD:
02212             {
02213                 return keypadAdd;
02214             }
02215 
02216             case VK_DECIMAL:
02217             {
02218                 return keypadPeriod;
02219             }
02220 
02221             case VK_NUMPAD0:
02222             {
02223                 return keypad0;
02224             }
02225 
02226             case VK_NUMPAD1:
02227             {
02228                 return keypad1;
02229             }
02230 
02231             case VK_NUMPAD2:
02232             {
02233                 return keypad2;
02234             }
02235 
02236             case VK_NUMPAD3:
02237             {
02238                 return keypad3;
02239             }
02240 
02241             case VK_NUMPAD4:
02242             {
02243                 return keypad4;
02244             }
02245 
02246             case VK_NUMPAD5:
02247             {
02248                 return keypad5;
02249             }
02250 
02251             case VK_NUMPAD6:
02252             {
02253                 return keypad6;
02254             }
02255 
02256             case VK_NUMPAD7:
02257             {
02258                 return keypad7;
02259             }
02260 
02261             case VK_NUMPAD8:
02262             {
02263                 return keypad8;
02264             }
02265 
02266             case VK_NUMPAD9:
02267             {
02268                 return keypad9;
02269             }
02270 
02271             case VK_LWIN:
02272             {
02273                 return leftWindow;
02274             }
02275 
02276             case VK_RWIN:
02277             {
02278                 return rightWindow;
02279             }
02280 
02281             default:
02282             {
02283                 return wordParam;
02284             }
02285         }
02286     }
02287 
02288     static void Windows_SetWindowIcon(window_t* window, const char* icon, unsigned int width,
      unsigned int height)
02289     {
02290         SendMessage(window->windowHandle, (UINT)WM_SETICON, ICON_BIG,
02291             (LPARAM)LoadImage(window->instanceHandle, icon, IMAGE_ICON, (int)width, (int)height,
```

```
    LR_LOADFROMFILE));
02292     }
02293
02294 #elif defined(TW_LINUX)
02295
02296     enum decorator_t
02297     {
02298         linuxBorder = 1L << 1,
02299         linuxMove = 1L << 2,
02300         linuxMinimize = 1L << 3,
02301         linuxMaximize = 1L << 4,
02302         linuxClose = 1L << 5,
02303     };
02304
02305     enum hint_t
02306     {
02307         function = 1,
02308         decorator,
02309     };
02310
02311     Display*            currentDisplay;
02312     XEvent              currentEvent;
02313     /* these atoms are needed to change window states via the extended window manager*/
02314     Atom                AtomState;                  /**< Atom for the state of the
    window */                           // _NET_WM_STATE
02315     Atom                AtomHidden;                 /**< Atom for the current hidden
    state of the window */            // _NET_WM_STATE_HIDDEN
02316     Atom                AtomFullScreen;             /**< Atom for the full
    screen state of the window */             // _NET_WM_STATE_FULLSCREEN
02317     Atom                AtomMaxHorz;                /**< Atom for the maximized
    horizontally state of the window */     // _NET_WM_STATE_MAXIMIZED_HORZ
02318     Atom                AtomMaxVert;                /**< Atom for the maximized
    vertically state of the window */       // _NET_WM_STATE_MAXIMIZED_VERT
02319     Atom                AtomClose;                  /**< Atom for closing the window
    */                                  // _NET_WM_CLOSE_WINDOW
02320     Atom                AtomActive;                 /**< Atom for the active window
    */                                  // _NET_ACTIVE_WINDOW
02321     Atom                AtomDemandsAttention;       /**< Atom for when the
    window demands attention */               // _NET_WM_STATE_DEMANDS_ATTENTION
02322     Atom                AtomFocused;                /**< Atom for the focused state
    of the window */                    // _NET_WM_STATE_FOCUSED
02323     Atom                AtomCardinal;               /**< Atom for cardinal
    coordinates */                          // _NET_WM_CARDINAL
02324     Atom                AtomIcon;                   /**< Atom for the icon of the
    window */                           // _NET_WM_ICON
02325     Atom                AtomHints;                  /**< Atom for the window
    decorations */                          // _NET_WM_HINTS
02326
02327     Atom                AtomWindowType;             /**< Atom for the type of
    window */
02328     Atom                AtomWindowTypeDesktop;      /**< Atom for the
    desktop window type */                      //_NET_WM_WINDOW_TYPE_SPLASH
02329     Atom                AtomWindowTypeSplash;       /**< Atom for the
    splash screen window type */
02330     Atom                AtomWindowTypeNormal;       /**< Atom for the
    normal splash screen window type */
02331
02332     Atom                AtomAllowedActions;         /**< Atom for allowed
    window actions */
02333     Atom                AtomActionResize;           /**< Atom for allowing the
    window to be resized */
02334     Atom                AtomActionMinimize;         /**< Atom for allowing
    the window to be minimized */
02335     Atom                AtomActionShade;            /**< Atom for allowing the
    window to be shaded */
02336     Atom                AtomActionMaximizeHorz;     /**< Atom for
    allowing the window to be maximized horizontally */
02337     Atom                AtomActionMaximizeVert;     /**< Atom for
    allowing the window to be maximized vertically */
02338     Atom                AtomActionClose;            /**< Atom for allowing the
    window to be closed */
02339
02340     Atom                AtomDesktopGeometry;        /**< Atom for Desktop
    Geometry */
02341
02342     window_t* GetWindowByHandle(Window
    windowHandle)
02343     {
02344         for(unsigned int iter = 0; iter < windowList.size();
    iter++)
02345         {
02346             if (windowList[iter]->windowHandle ==
    windowHandle)
02347             {
02348                 return windowList[iter].get();
02349             }
02350         }
```

```
02351         return nullptr;
02352     }
02353
02354     window_t* GetWindowByEvent(XEvent
      currentEvent)
02355     {
02356         switch(currentEvent.type)
02357         {
02358             case Expose:
02359             {
02360                 return GetWindowByHandle(currentEvent.
      xexpose.window);
02361             }
02362
02363             case DestroyNotify:
02364             {
02365                 return GetWindowByHandle(currentEvent.
      xdestroywindow.window);
02366             }
02367
02368             case CreateNotify:
02369             {
02370                 return GetWindowByHandle(currentEvent.
      xcreatewindow.window);
02371             }
02372
02373             case KeyPress:
02374             {
02375                 return GetWindowByHandle(currentEvent.
      xkey.window);
02376             }
02377
02378             case KeyRelease:
02379             {
02380                 return GetWindowByHandle(currentEvent.
      xkey.window);
02381             }
02382
02383             case ButtonPress:
02384             {
02385                 return GetWindowByHandle(currentEvent.
      xbutton.window);
02386             }
02387
02388             case ButtonRelease:
02389             {
02390                 return GetWindowByHandle(currentEvent.
      xbutton.window);
02391             }
02392
02393             case MotionNotify:
02394             {
02395                 return GetWindowByHandle(currentEvent.
      xmotion.window);
02396             }
02397
02398             case FocusIn:
02399             {
02400                 return GetWindowByHandle(currentEvent.
      xfocus.window);
02401             }
02402
02403             case FocusOut:
02404             {
02405                 return GetWindowByHandle(currentEvent.
      xfocus.window);
02406             }
02407
02408             case ResizeRequest:
02409             {
02410                 return GetWindowByHandle(currentEvent.
      xresizerequest.window);
02411             }
02412
02413             case ConfigureNotify:
02414             {
02415                 return GetWindowByHandle(currentEvent.
      xconfigure.window);
02416             }
02417
02418             case PropertyNotify:
02419             {
02420                 return GetWindowByHandle(currentEvent.
      xproperty.window);
02421             }
02422
02423             case GravityNotify:
```

```
02424             {
02425                 return GetWindowByHandle(currentEvent.
      xgravity.window);
02426             }
02427 
02428             case ClientMessage:
02429             {
02430                 return GetWindowByHandle(currentEvent.
      xclient.window);
02431             }
02432 
02433             case VisibilityNotify:
02434             {
02435                 return GetWindowByHandle(currentEvent.
      xvisibility.window);
02436             }
02437 
02438             default:
02439             {
02440                 return nullptr;
02441             }
02442         }
02443     }
02444 
02445     void InitializeAtoms()
02446     {
02447         AtomState = XInternAtom(currentDisplay, "_NET_WM_STATE",
      false);
02448         AtomFullScreen = XInternAtom(
      currentDisplay, "_NET_WM_STATE_FULLSCREEN", false);
02449         AtomMaxHorz = XInternAtom(currentDisplay,
      "_NET_WM_STATE_MAXIMIZED_HORZ", false);
02450         AtomMaxVert = XInternAtom(currentDisplay,
      "_NET_WM_STATE_MAXIMIZED_VERT", false);
02451         AtomClose = XInternAtom(currentDisplay, "WM_DELETE_WINDOW",
      false);
02452         AtomHidden = XInternAtom(currentDisplay,
      "_NET_WM_STATE_HIDDEN", false);
02453         AtomActive = XInternAtom(currentDisplay,
      "_NET_ACTIVE_WINDOW", false);
02454         AtomDemandsAttention = XInternAtom(
      currentDisplay, "_NET_WM_STATE_DEMANDS_ATTENTION", false);
02455         AtomFocused = XInternAtom(currentDisplay,
      "_NET_WM_STATE_FOCUSED", false);
02456         AtomCardinal = XInternAtom(currentDisplay, "CARDINAL",
      false);
02457         AtomIcon = XInternAtom(currentDisplay, "_NET_WM_ICON", false)
      ;
02458         AtomHints = XInternAtom(currentDisplay, "_MOTIF_WM_HINTS",
      true);
02459 
02460         AtomWindowType = XInternAtom(
      currentDisplay, "_NET_WM_WINDOW_TYPE", false);
02461         AtomWindowTypeDesktop = XInternAtom(
      currentDisplay, "_NET_WM_WINDOW_TYPE_UTILITY", false);
02462         AtomWindowTypeSplash = XInternAtom(
      currentDisplay, "_NET_WM_WINDOW_TYPE_SPLASH", false);
02463         AtomWindowTypeNormal = XInternAtom(
      currentDisplay, "_NET_WM_WINDOW_TYPE_NORMAL", false);
02464 
02465         AtomAllowedActions = XInternAtom(
      currentDisplay, "_NET_WM_ALLOWED_ACTIONS", false);
02466         AtomActionResize = XInternAtom(
      currentDisplay, "WM_ACTION_RESIZE", false);
02467         AtomActionMinimize = XInternAtom(
      currentDisplay, "_WM_ACTION_MINIMIZE", false);
02468         AtomActionShade = XInternAtom(
      currentDisplay, "WM_ACTION_SHADE", false);
02469         AtomActionMaximizeHorz = XInternAtom(
      currentDisplay, "_WM_ACTION_MAXIMIZE_HORZ", false);
02470         AtomActionMaximizeVert = XInternAtom(
      currentDisplay, "_WM_ACTION_MAXIMIZE_VERT", false);
02471         AtomActionClose = XInternAtom(
      currentDisplay, "_WM_ACTION_CLOSE", false);
02472 
02473         AtomDesktopGeometry = XInternAtom(
      currentDisplay, "_NET_DESKTOP_GEOMETRY", false);
02474     }
02475 
02476     std::error_code Linux_InitializeWindow(
      window_t* window)
02477     {
02478         window->attributes = new int[ 5]{
02479             GLX_RGBA,
02480             GLX_DOUBLEBUFFER,
02481             GLX_DEPTH_SIZE,
02482             window->depthBits,
```

```
02483             None};
02484
02485         window->decorators = 1;
02486         window->currentWindowStyle |= linuxClose |
      linuxMaximize | linuxMinimize | linuxMove;
02487
02488         if (!currentDisplay)
02489         {
02490             return TinyWindow::error_t::
      linuxCannotConnectXServer;
02491         }
02492
02493         //window->VisualInfo = glXGetVisualFromFBConfig(GetDisplay(),
       GetBestFrameBufferConfig(window));
02494
02495         window->visualInfo = glXChooseVisual(
      currentDisplay, 0, window->attributes);
02496
02497         if (!window->visualInfo)
02498         {
02499             return TinyWindow::error_t::
      linuxInvalidVisualinfo;
02500         }
02501
02502         window->setAttributes.colormap =
      XCreateColormap(currentDisplay,
02503             DefaultRootWindow(currentDisplay),
02504             window->visualInfo->visual, AllocNone);
02505
02506         window->setAttributes.event_mask =
      ExposureMask | KeyPressMask
02507             | KeyReleaseMask | MotionNotify |
      ButtonPressMask | ButtonReleaseMask
02508             | FocusIn | FocusOut | Button1MotionMask |
      Button2MotionMask | Button3MotionMask |
02509             Button4MotionMask | Button5MotionMask |
      PointerMotionMask | FocusChangeMask
02510             | VisibilityChangeMask | PropertyChangeMask |
      SubstructureNotifyMask;
02511
02512         window->windowHandle = XCreateWindow(
      currentDisplay,
02513             XDefaultRootWindow(currentDisplay), 0, 0,
02514             window->resolution.width, window->
      resolution.height,
02515             0, window->visualInfo->depth, InputOutput,
02516             window->visualInfo->visual, CWColormap |
      CWEventMask,
02517             &window->setAttributes);
02518
02519         if(!window->windowHandle)
02520         {
02521             return TinyWindow::error_t::
      linuxCannotCreateWindow;
02522             exit(0);
02523         }
02524
02525         XMapWindow(currentDisplay, window->
      windowHandle);
02526         XStoreName(currentDisplay, window->
      windowHandle,
02527             window->name);
02528
02529         XSetWMProtocols(currentDisplay, window->
      windowHandle, &AtomClose, true);
02530
02531         Platform_InitializeGL(window);
02532         return TinyWindow::error_t::success;
02533     }
02534
02535     void Linux_ShutdownWindow(window_t* window)
02536     {
02537         XDestroyWindow(currentDisplay, window->
      windowHandle);
02538     }
02539
02540     void Linux_Shutdown(void)
02541     {
02542         for(unsigned int iter = 0; iter < windowList.size();
      iter++)
02543         {
02544             Linux_ShutdownWindow(windowList[
      iter].get());
02545         }
02546
02547         XCloseDisplay(currentDisplay);
02548     }
```

```
02549
02550     void Linux_ProcessEvents(XEvent currentEvent)
02551     {
02552         window_t* window = GetWindowByEvent(
      currentEvent);
02553
02554         switch (currentEvent.type)
02555         {
02556             case Expose:
02557             {
02558                 break;
02559             }
02560
02561             case DestroyNotify:
02562             {
02563                 printf("shutting down \n");
02564                 if (window->destroyedEvent != nullptr)
02565                 {
02566                     window->destroyedEvent();
02567                 }
02568                 ShutdownWindow(window);
02569
02570                 break;
02571             }
02572
02573             /*case CreateNotify:
02574             {
02575             printf("Window was created\n");
02576             l_Window->InitializeGL();
02577
02578             if(IsValid(l_Window->m_OnCreated))
02579             {
02580             l_Window->m_OnCreated();
02581             }
02582
02583             break;
02584             } */
02585
02586             case KeyPress:
02587             {
02588                 unsigned int functionKeysym =
      XkbKeycodeToKeysym(
02589                     currentDisplay, currentEvent.
      xkey.keycode, 0, currentEvent.xkey.state &
      ShiftMask ? 1 : 0);
02590
02591                 if (functionKeysym <= 255)
02592                 {
02593                     window->keys[ functionKeysym] =
      keyState_t::down;
02594                     if (window->keyEvent != nullptr)
02595                     {
02596                         window->keyEvent(functionKeysym,
      keyState_t::down);
02597                     }
02598                 }
02599
02600                 else
02601                 {
02602                     window->keys[ Linux_TranslateKey(
      functionKeysym)] = keyState_t::down;
02603
02604                     if (window->keyEvent != nullptr)
02605                     {
02606                         window->keyEvent(Linux_TranslateKey(
      functionKeysym), keyState_t::down);
02607                     }
02608                 }
02609
02610                 break;
02611             }
02612
02613             case KeyRelease:
02614             {
02615                 bool isRetriggered = false;
02616                 if (XEventsQueued(currentDisplay,
      QueuedAfterReading))
02617                 {
02618                     XEvent nextEvent;
02619                     XPeekEvent(currentDisplay, &
      nextEvent);
02620
02621                     if (nextEvent.type == KeyPress &&
02622                         nextEvent.xkey.time ==
      currentEvent.xkey.time &&
02623                         nextEvent.xkey.keycode ==
      currentEvent.xkey.keycode)
```

```
02624                     {
02625                         unsigned int functionKeysym =
      XkbKeycodeToKeysym(
02626                             currentDisplay, currentEvent.
      xkey.keycode, 0,
02627                             currentEvent.xkey.state &
      ShiftMask ? 1 : 0);
02628 
02629                         XNextEvent(currentDisplay, &
      currentEvent);
02630                         window->keyEvent(Linux_TranslateKey(
      functionKeysym), keyState_t::down);
02631                         isRetriggered = true;
02632                     }
02633                 }
02634 
02635                 if (!isRetriggered)
02636                 {
02637                     unsigned int functionKeysym =
      XkbKeycodeToKeysym(
02638                     currentDisplay, currentEvent.
      xkey.keycode, 0, currentEvent.xkey.state &
      ShiftMask ? 1 : 0);
02639 
02640                     if (functionKeysym <= 255)
02641                     {
02642                         window->keys[ functionKeysym] =
      keyState_t::up;
02643 
02644                         if (window->keyEvent != nullptr)
02645                         {
02646                             window->keyEvent(functionKeysym,
      keyState_t::up);
02647                         }
02648                     }
02649 
02650                     else
02651                     {
02652                         window->keys[ Linux_TranslateKey(
      functionKeysym)] = keyState_t::up;
02653 
02654                         if (window->keyEvent != nullptr)
02655                         {
02656                             window->keyEvent(
      Linux_TranslateKey(functionKeysym), keyState_t::
      up);
02657                         }
02658                     }
02659 
02660                     if (window->keyEvent != nullptr)
02661                     {
02662                         window->keyEvent(Linux_TranslateKey(
      functionKeysym), keyState_t::up);
02663                     }
02664                 }
02665 
02666                 break;
02667             }
02668 
02669             case ButtonPress:
02670             {
02671                 switch (currentEvent.xbutton.button)
02672                 {
02673                 case 1:
02674                 {
02675                     window->mouseButton[ (unsigned int)
      mouseButton_t::left] = buttonState_t::down;
02676 
02677                     if (window->mouseButtonEvent != nullptr)
02678                     {
02679                         window->mouseButtonEvent(
      mouseButton_t::left, buttonState_t::down);
02680                     }
02681                     break;
02682                 }
02683 
02684                 case 2:
02685                 {
02686                     window->mouseButton[ (unsigned int)
      mouseButton_t::middle] = buttonState_t::down;
02687 
02688                     if (window->mouseButtonEvent != nullptr)
02689                     {
02690                         window->mouseButtonEvent(
      mouseButton_t::middle, buttonState_t::down);
02691                     }
02692                     break;
```

```
02693                 }
02694
02695                 case 3:
02696                 {
02697                     window->mouseButton[ (unsigned int)
      mouseButton_t::right] = buttonState_t::down;
02698
02699                     if (window->mouseButtonEvent != nullptr)
02700                     {
02701                         window->mouseButtonEvent(
      mouseButton_t::right, buttonState_t::down);
02702                     }
02703                     break;
02704                 }
02705
02706                 case 4:
02707                 {
02708                     window->mouseButton[ (unsigned int)
      mouseScroll_t::up] = buttonState_t::down;
02709
02710                     if (window->mouseWheelEvent != nullptr)
02711                     {
02712                         window->mouseWheelEvent(
      mouseScroll_t::down);
02713                     }
02714                     break;
02715                 }
02716
02717                 case 5:
02718                 {
02719                     window->mouseButton[ (unsigned int)
      mouseScroll_t::down] = buttonState_t::down;
02720
02721                     if (window->mouseWheelEvent != nullptr)
02722                     {
02723                         window->mouseWheelEvent(
      mouseScroll_t::down);
02724                     }
02725                     break;
02726                 }
02727
02728                 default:
02729                 {
02730                     //need to add more mouse buttons
02731                     break;
02732                 }
02733                 }
02734
02735                 break;
02736             }
02737
02738             case ButtonRelease:
02739             {
02740                 switch (currentEvent.xbutton.button)
02741                 {
02742                 case 1:
02743                 {
02744                     //the left mouse button was released
02745                     window->mouseButton[ (unsigned int)
      mouseButton_t::left] = buttonState_t::up;
02746
02747                     if (window->mouseButtonEvent != nullptr)
02748                     {
02749                         window->mouseButtonEvent(
      mouseButton_t::left, buttonState_t::up);
02750                     }
02751                     break;
02752                 }
02753
02754                 case 2:
02755                 {
02756                     //the middle mouse button was released
02757                     window->mouseButton[ (unsigned int)
      mouseButton_t::middle] = buttonState_t::up;
02758
02759                     if (window->mouseButtonEvent != nullptr)
02760                     {
02761                         window->mouseButtonEvent(
      mouseButton_t::middle, buttonState_t::up);
02762                     }
02763                     break;
02764                 }
02765
02766                 case 3:
02767                 {
02768                     //the right mouse button was released
02769                     window->mouseButton[ (unsigned int)
```

```
mouseButton_t::right] = buttonState_t::up;
02770
02771                     if (window->mouseButtonEvent != nullptr)
02772                     {
02773                         window->mouseButtonEvent(
mouseButton_t::right, buttonState_t::up);
02774                     }
02775                     break;
02776                 }
02777
02778                 case 4:
02779                 {
02780                     //the mouse wheel was scrolled up
02781                     window->mouseButton[ (unsigned int)
mouseScroll_t::up] = buttonState_t::down;
02782                     break;
02783                 }
02784
02785                 case 5:
02786                 {
02787                     //the mouse wheel was scrolled down
02788                     window->mouseButton[ (unsigned int)
mouseScroll_t::down] = buttonState_t::down;
02789                     break;
02790                 }
02791
02792                 default:
02793                 {
02794                     //need to add more mouse buttons
02795                     break;
02796                 }
02797                 }
02798                 break;
02799             }
02800
02801             //when the mouse/pointer device is moved
02802             case MotionNotify:
02803             {
02804                 //set the windows mouse position to match the event
02805                 window->mousePosition.x =
02806                     currentEvent.xmotion.x;
02807
02808                 window->mousePosition.y =
02809                     currentEvent.xmotion.y;
02810
02811                 ///set the screen mouse position to match the event
02812                 screenMousePosition.x = currentEvent.
xmotion.x_root;
02813                 screenMousePosition.y = currentEvent.
xmotion.y_root;
02814
02815                 if (window->mouseMoveEvent != nullptr)
02816                 {
02817                     window->mouseMoveEvent(currentEvent.
xmotion.x,
02818                         currentEvent.xmotion.y,
currentEvent.xmotion.x_root,
02819                         currentEvent.xmotion.y_root);
02820                 }
02821                 break;
02822             }
02823
02824             //when the window goes out of focus
02825             case FocusOut:
02826             {
02827                 window->inFocus = false;
02828                 if (window->focusEvent != nullptr)
02829                 {
02830                     window->focusEvent(
02831                         window->inFocus);
02832                 }
02833                 break;
02834             }
02835
02836             //when the window is back in focus (use to call restore callback?)
02837             case FocusIn:
02838             {
02839                 window->inFocus = true;
02840
02841                 if (window->focusEvent != nullptr)
02842                 {
02843                     window->focusEvent(window->
inFocus);
02844                 }
02845                 break;
02846             }
02847
```

```
02848             //when a request to resize the window is made either by
02849             //dragging out the window or programmatically
02850             case ResizeRequest:
02851             {
02852                 window->resolution.width =
      currentEvent.xresizerequest.width;
02853                 window->resolution.height =
      currentEvent.xresizerequest.height;
02854 
02855                 glViewport(0, 0,
02856                     window->resolution.width,
02857                     window->resolution.height);
02858 
02859                 if (window->resizeEvent != nullptr)
02860                 {
02861                     window->resizeEvent(currentEvent.
      xresizerequest.width,
02862                         currentEvent.xresizerequest.
      height);
02863                 }
02864 
02865                 break;
02866             }
02867 
02868             //when a request to configure the window is made
02869             case ConfigureNotify:
02870             {
02871                 glViewport(0, 0, currentEvent.
      xconfigure.width,
02872                     currentEvent.xconfigure.height);
02873 
02874                 //check if window was resized
02875                 if ((unsigned int)currentEvent.xconfigure.
      width != window->resolution.width
02876                     || (unsigned int)currentEvent.xconfigure.
      height != window->resolution.height)
02877                 {
02878                     if (window->resizeEvent != nullptr)
02879                     {
02880                         window->resizeEvent(currentEvent.
      xconfigure.width, currentEvent.xconfigure.
      height);
02881                     }
02882 
02883                     window->resolution.width =
      currentEvent.xconfigure.width;
02884                     window->resolution.height =
      currentEvent.xconfigure.height;
02885                 }
02886 
02887                 //check if window was moved
02888                 if ((unsigned int)currentEvent.xconfigure.
      x != window->position.x
02889                     || (unsigned int)currentEvent.xconfigure.
      y != window->position.y)
02890                 {
02891                     if (window->movedEvent != nullptr)
02892                     {
02893                         window->movedEvent(currentEvent.
      xconfigure.x, currentEvent.xconfigure.y);
02894                     }
02895 
02896                     window->position.x = currentEvent.
      xconfigure.x;
02897                     window->position.y = currentEvent.
      xconfigure.y;
02898                 }
02899                 break;
02900             }
02901 
02902             case PropertyNotify:
02903             {
02904                 //this is needed in order to read from the windows WM_STATE Atomic
02905                 //to determine if the property notify event was caused by a client
02906                 //iconify event(minimizing the window), a maximise event, a focus
02907                 //event and an attention demand event. NOTE these should only be
02908                 //for eventts that are not triggered programatically
02909 
02910                 Atom type;
02911                 int format;
02912                 ulong numItems, bytesAfter;
02913                 unsigned char* properties = nullptr;
02914 
02915                 XGetWindowProperty(currentDisplay,
      currentEvent.xproperty.window,
02916                     AtomState,
02917                     0, LONG_MAX, false, AnyPropertyType,
```

```
02918                       &type, &format, &numItems, &
      bytesAfter,
02919                       & properties);
02920
02921                   if (properties && (format == 32))
02922                   {
02923                       //go through each property and match it to an existing Atomic state
02924                       for (unsigned int currentItem = 0;
      currentItem < numItems; currentItem++)
02925                       {
02926                           Atom currentProperty = ((long*)(
      properties))[ currentItem];
02927
02928                           if (currentProperty == AtomHidden)
02929                           {
02930                               //window was minimized
02931                               if (window->minimizedEvent != nullptr)
02932                               {
02933                                   //if the minimized callback for the window was set

02934                                   window->minimizedEvent();
02935                               }
02936                           }
02937
02938                           if (currentProperty == AtomMaxVert ||
02939                               currentProperty == AtomMaxVert)
02940                           {
02941                               //window was maximized
02942                               if (window->maximizedEvent != nullptr)
02943                               {
02944                                   //if the maximized callback for the window was set
02945                                   window->maximizedEvent();
02946                               }
02947                           }
02948
02949                           if (currentProperty == AtomFocused)
02950                           {
02951                               //window is now in focus. we can ignore this is as FocusIn/FocusOut does
       this anyway
02952                           }
02953
02954                           if (currentProperty ==
      AtomDemandsAttention)
02955                           {
02956                               //the window demands user attention
02957                           }
02958                       }
02959                   }
02960
02961                   break;
02962               }
02963
02964               case GravityNotify:
02965               {
02966                   //this is only supposed to pop up when the parent of this window(if any) has something
       happen
02967                   //to it so that this window can react to said event as well.
02968                   break;
02969               }
02970
02971               //check for events that were created by the TinyWindow manager
02972               case ClientMessage:
02973               {
02974                   const char* atomName = XGetAtomName(
      currentDisplay, currentEvent.xclient.
      message_type);
02975                   if (atomName != nullptr)
02976                   {
02977                       //printf("%s\n", l_AtomName);
02978                   }
02979
02980                   if ((Atom)currentEvent.xclient.data.
      l[ 0] == AtomClose)
02981                   {
02982                       window->shouldClose = true;
02983                       if(window->destroyedEvent != nullptr)
02984                       {
02985                           window->destroyedEvent();
02986                       }
02987                       break;
02988                   }
02989
02990                   //check if full screen
02991                   if ((Atom)currentEvent.xclient.data.
      l[ 1] == AtomFullScreen)
02992                   {
02993                       break;
```

```
02994                 }
02995                 break;
02996
02997             }
02998
02999             default:
03000             {
03001                 return;
03002             }
03003         }
03004     }
03005
03006     //debugging. used to determine what type of event was generated
03007     static const char* Linux_GetEventType(XEvent
      currentEvent)
03008     {
03009         switch (currentEvent.type)
03010         {
03011         case MotionNotify:
03012         {
03013             return "Motion Notify Event\n";
03014         }
03015
03016         case ButtonPress:
03017         {
03018             return "Button Press Event\n";
03019         }
03020
03021         case ButtonRelease:
03022         {
03023             return "Button Release Event\n";
03024         }
03025
03026         case ColormapNotify:
03027         {
03028             return "Color Map Notify event \n";
03029         }
03030
03031         case EnterNotify:
03032         {
03033             return "Enter Notify Event\n";
03034         }
03035
03036         case LeaveNotify:
03037         {
03038             return "Leave Notify Event\n";
03039         }
03040
03041         case Expose:
03042         {
03043             return "Expose Event\n";
03044         }
03045
03046         case GraphicsExpose:
03047         {
03048             return "Graphics expose event\n";
03049         }
03050
03051         case NoExpose:
03052         {
03053             return "No Expose Event\n";
03054         }
03055
03056         case FocusIn:
03057         {
03058             return "Focus In Event\n";
03059         }
03060
03061         case FocusOut:
03062         {
03063             return "Focus Out Event\n";
03064         }
03065
03066         case KeymapNotify:
03067         {
03068             return "Key Map Notify Event\n";
03069         }
03070
03071         case KeyPress:
03072         {
03073             return "Key Press Event\n";
03074         }
03075
03076         case KeyRelease:
03077         {
03078             return "Key Release Event\n";
03079         }
```

```
03080
03081         case PropertyNotify:
03082         {
03083             return "Property Notify Event\n";
03084         }
03085
03086         case ResizeRequest:
03087         {
03088             return "Resize Property Event\n";
03089         }
03090
03091         case CirculateNotify:
03092         {
03093             return "Circulate Notify Event\n";
03094         }
03095
03096         case ConfigureNotify:
03097         {
03098             return "configure Notify Event\n";
03099         }
03100
03101         case DestroyNotify:
03102         {
03103             return "Destroy Notify Request\n";
03104         }
03105
03106         case GravityNotify:
03107         {
03108             return "Gravity Notify Event \n";
03109         }
03110
03111         case MapNotify:
03112         {
03113             return "Map Notify Event\n";
03114         }
03115
03116         case ReparentNotify:
03117         {
03118             return "Reparent Notify Event\n";
03119         }
03120
03121         case UnmapNotify:
03122         {
03123             return "Unmap notify event\n";
03124         }
03125
03126         case MapRequest:
03127         {
03128             return "Map request event\n";
03129         }
03130
03131         case ClientMessage:
03132         {
03133             return "Client Message Event\n";
03134         }
03135
03136         case MappingNotify:
03137         {
03138             return "Mapping notify event\n";
03139         }
03140
03141         case SelectionClear:
03142         {
03143             return "Selection Clear event\n";
03144         }
03145
03146         case SelectionNotify:
03147         {
03148             return "Selection Notify Event\n";
03149         }
03150
03151         case SelectionRequest:
03152         {
03153             return "Selection Request event\n";
03154         }
03155
03156         case VisibilityNotify:
03157         {
03158             return "Visibility Notify Event\n";
03159         }
03160
03161         default:
03162         {
03163             return 0;
03164         }
03165         }
03166     }
```

```
03167
03168     //translate keys from X keys to TinyWindow Keys
03169     static unsigned int Linux_TranslateKey(unsigned int
      keySymbol)
03170     {
03171         switch (keySymbol)
03172         {
03173         case XK_Escape:
03174         {
03175             return escape;
03176         }
03177
03178         case XK_Home:
03179         {
03180             return home;
03181         }
03182
03183         case XK_Left:
03184         {
03185             return arrowLeft;
03186         }
03187
03188         case XK_Right:
03189         {
03190             return arrowRight;
03191         }
03192
03193         case XK_Up:
03194         {
03195             return arrowUp;
03196         }
03197
03198         case XK_Down:
03199         {
03200             return arrowDown;
03201         }
03202
03203         case XK_Page_Up:
03204         {
03205             return pageUp;
03206         }
03207
03208         case XK_Page_Down:
03209         {
03210             return pageDown;
03211         }
03212
03213         case XK_End:
03214         {
03215             return end;
03216         }
03217
03218         case XK_Print:
03219         {
03220             return printScreen;
03221         }
03222
03223         case XK_Insert:
03224         {
03225             return insert;
03226         }
03227
03228         case XK_Num_Lock:
03229         {
03230             return numLock;
03231         }
03232
03233         case XK_KP_Multiply:
03234         {
03235             return keypadMultiply;
03236         }
03237
03238         case XK_KP_Add:
03239         {
03240             return keypadAdd;
03241         }
03242
03243         case XK_KP_Subtract:
03244         {
03245             return keypadSubtract;
03246         }
03247
03248         case XK_KP_Decimal:
03249         {
03250             return keypadPeriod;
03251         }
03252
```

```
03253         case XK_KP_Divide:
03254         {
03255             return keypadDivide;
03256         }
03257
03258         case XK_KP_0:
03259         {
03260             return keypad0;
03261         }
03262
03263         case XK_KP_1:
03264         {
03265             return keypad1;
03266         }
03267
03268         case XK_KP_2:
03269         {
03270             return keypad2;
03271         }
03272
03273         case XK_KP_3:
03274         {
03275             return keypad3;
03276         }
03277
03278         case XK_KP_4:
03279         {
03280             return keypad4;
03281         }
03282
03283         case XK_KP_5:
03284         {
03285             return keypad5;
03286         }
03287
03288         case XK_KP_6:
03289         {
03290             return keypad6;
03291         }
03292
03293         case XK_KP_7:
03294         {
03295             return keypad7;
03296         }
03297
03298         case XK_KP_8:
03299         {
03300             return keypad8;
03301         }
03302
03303         case XK_KP_9:
03304         {
03305             return keypad9;
03306         }
03307
03308         case XK_F1:
03309         {
03310             return F1;
03311         }
03312
03313         case XK_F2:
03314         {
03315             return F2;
03316         }
03317
03318         case XK_F3:
03319         {
03320             return F3;
03321         }
03322
03323         case XK_F4:
03324         {
03325             return F4;
03326         }
03327
03328         case XK_F5:
03329         {
03330             return F5;
03331         }
03332
03333         case XK_F6:
03334         {
03335             return F6;
03336         }
03337
03338         case XK_F7:
03339         {
```

```
03340             return F7;
03341         }
03342
03343         case XK_F8:
03344         {
03345             return F8;
03346         }
03347
03348         case XK_F9:
03349         {
03350             return F9;
03351         }
03352
03353         case XK_F10:
03354         {
03355             return F10;
03356         }
03357
03358         case XK_F11:
03359         {
03360             return F11;
03361         }
03362
03363         case XK_F12:
03364         {
03365             return F12;
03366         }
03367
03368         case XK_Shift_L:
03369         {
03370             return leftShift;
03371         }
03372
03373         case XK_Shift_R:
03374         {
03375             return rightShift;
03376         }
03377
03378         case XK_Control_R:
03379         {
03380             return rightControl;
03381         }
03382
03383         case XK_Control_L:
03384         {
03385             return leftControl;
03386         }
03387
03388         case XK_Caps_Lock:
03389         {
03390             return capsLock;
03391         }
03392
03393         case XK_Alt_L:
03394         {
03395             return leftAlt;
03396         }
03397
03398         case XK_Alt_R:
03399         {
03400             return rightAlt;
03401         }
03402
03403         default:
03404         {
03405             return 0;
03406         }
03407         }
03408     }
03409
03410     std::error_code Linux_SetWindowIcon(void)
      /*std::unique_ptr<window_t> window, const char* icon, unsigned int width, unsigned int height */
03411     {
03412         //sorry :(
03413         return TinyWindow::error_t::
      linuxFunctionNotImplemented;
03414     }
03415
03416     GLXFBConfig GetBestFrameBufferConfig(
      window_t* givenWindow)
03417     {
03418         const int visualAttributes[ ] =
03419         {
03420             GLX_X_RENDERABLE, true,
03421             GLX_DRAWABLE_TYPE, GLX_WINDOW_BIT,
03422             GLX_X_VISUAL_TYPE, GLX_TRUE_COLOR,
03423             GLX_RED_SIZE, givenWindow->colorBits,
```

```
03424             GLX_GREEN_SIZE, givenWindow->colorBits,
03425             GLX_BLUE_SIZE, givenWindow->colorBits,
03426             GLX_ALPHA_SIZE, givenWindow->colorBits,
03427             GLX_DEPTH_SIZE, givenWindow->depthBits,
03428             GLX_STENCIL_SIZE, givenWindow->
      stencilBits,
03429             GLX_DOUBLEBUFFER, true,
03430             None
03431         };
03432
03433         int frameBufferCount;
03434         unsigned int bestBufferConfig;//, bestNumSamples = 0;
03435         GLXFBConfig* configs = glXChooseFBConfig(
      currentDisplay, 0, visualAttributes, &
      frameBufferCount);
03436
03437         for (int currentConfig = 0; currentConfig <
      frameBufferCount; currentConfig++)
03438         {
03439             XVisualInfo* visualInfo =
      glXGetVisualFromFBConfig(currentDisplay,
      configs[currentConfig]);
03440
03441             if (visualInfo)
03442             {
03443                 //printf("%i %i %i\n", VisInfo->depth, VisInfo->bits_per_rgb, VisInfo->colormap_size);
03444                 int samples, sampleBuffer;
03445                 glXGetFBConfigAttrib(currentDisplay,
      configs[ currentConfig], GLX_SAMPLE_BUFFERS, &
      sampleBuffer);
03446                 glXGetFBConfigAttrib(currentDisplay,
      configs[currentConfig], GLX_SAMPLES, &samples);
03447
03448                 if (sampleBuffer && samples > -1)
03449                 {
03450                     bestBufferConfig = currentConfig;
03451                     //bestNumSamples = samples;
03452                 }
03453             }
03454
03455             XFree(visualInfo);
03456         }
03457
03458         GLXFBConfig BestConfig = configs[
      bestBufferConfig];
03459
03460         XFree(configs);
03461
03462         return BestConfig;
03463     }
03464
03465 #endif
03466 };
03467 }
03468
03469 #endif
```

8.5 C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/README.md File Reference

8.6 C:/Users/ziyad/Documents/Portfolio/dependencies/tinywindow/README.md

```
00001 TinyWindow
00002 ==========
00003
00004 a cross platform single header window management API
```

Index