

# **EuGL - An OpenGL interface for Euphoria 2.1+**

**Version 031022**

Mic, 2000/2003

# Contents

CONTENTS.....	2
THE PACKAGE.....	3
WHAT'S NEW .....	5
WHAT IS THIS ?.....	8
USAGE.....	8
<i>Basics</i> .....	8
<i>Creating an application with EuGL</i> .....	8
<i>GL extensions</i> .....	9
<i>WGL extensions</i> .....	10
AN EXAMPLE.....	10
WHAT'S DONE SO FAR.....	11
TO DO.....	11
THANKS TO.....	11
APPENDIX A — GLOBAL ROUTINES.....	12
APPENDIX B — GLOBAL VARIABLES AND CONSTANTS.....	16

# The Package

## Core libraries

- eugl.ew
- gl.ew
- glaux.ew
- glu.ew
- ewin32api.ew

The main file, this is the one to include in your programs.

A bunch of constants & functions.

- " -

- " -

- " -

## Support libraries

- gl\_asc.e
- winbmp.ew

3DStudio ASCII-object loader.

BMP loader used by some of the examples.

## Data files

- atmos.bmp
- circle.bmp
- euglv.bmp
- face.bmp
- floor.bmp
- gumpy.bmp
- hippy.bmp
- mur043.bmp
- floor.tga
- lightmap.tga
- teapot.asc

Texture used by tunnel.exw.

Texture used by circles.exw.

Sample BMP for use with EuGLView.

Texture used by multitexture.exw and sphere\_tex.exw

Texture used by vol\_fog.exw.

Texture used by multitexture.exw.

Texture used by tex\_blend.exw.

Texture used by vol\_fog.exw.

Texture used by lit\_floor.exw

Texture used by lit\_floor.exw

Sample mesh for use with EuGLView.

## Sample programs

- 3d\_text.exw
- accum.exw
- anisotropic.exw
- bitmap\_font.exw
- bounce.exw
- cel\_shade.exw
- circles.exw
- dragnet2k.exw
- drawf.exw
- EuGLView.exw
- fireworks.exw
- lit\_floor.exw
- mouse.exw
- multitexture.exw
- nurbs.exw
- planes.exw
- quadrics.exw

Example of drawing 3-dimensional font characters with the help of `wglUseFontOutlines`.

A test program for the `glAccum` function.

Shows anisotropic texture filtering (requires a video card that supports the `GL_EXT_texture_filter_anisotropic` extension).

Shows how to use `wglUseFontBitmaps` to draw text in an OpenGL scene.

A bouncing lit sphere.

An example of cel (toon) shading.

A bunch of spinning circles that are blended together.

Example of lighting and display lists.

Example of how to use `glBitmap`.

A nice little image-/mesh-viewing program.

A simple fireworks effect.

Shows the usage of some GL and WGL extensions.

An example of using mouse-events in an EuGL program (+fog).

Example of multi-texturing. Requires OpenGL 1.3 or higher to run.

An example of GLU NURBs and surface trimming.

Some paper-planes flying around.

Example of using GLU quadrics for drawing a sphere.

- radial\_blur.exw Fake radial blur effect using texture stretching and filtering.
- rotating\_cube.exw Example of a spinning flat shaded cube.
- rotating\_quad.exw Example of a spinning gouraud shaded quadangle.
- scroller.exw Example of how to use `wglUseFontOutlines` to create 3D objects from TrueType fonts.
- sphere\_tex.exw Example of texture mapping a sphere.
- static\_tri.exw Example of a gouraud shaded triangle.
- tex\_blend.exw Shows a transparent texture-mapped cube.
- tunnel.exw A textured tunnel.
- vol\_fog.exw Shows how to get a single-pass volumetric fog using the `GL_EXT_fog_coord` extension.

## What's new

### 031022

- GL stuff:
  - Added some more GL extensions plus all (to me known) WGL extensions.
  - Added `wgl_supportsExtension` and `wgl_enableExtension` to deal with WGL extensions.
- Main:
  - Added `euglMouseFunc`, `euglMotionFunc`, `euglPassiveMotionFunc` and `euglTimerFunc`.
  - Added one new example program: `lit_floor.exw`, which shows how to use various GL and WGL extensions.
  - Fixed a few bugs in the libraries where incorrect parameters were passed to `define_c_func/proc` (eg. `NULL`).

### 030204

- GL stuff:
  - Added `glColor3ub`, `glGetString`, `glTexEnvf`, `wglCopyContext`, `wglCreateLayerContext`, `wglGetCurrentContext`, `wglGetCurrentDC`, `wglGetProcAddress`, `wglShareLists`, `wglSwapBuffers`, and `wglSwapLayerBuffers`.
  - Added support for some GL extensions. Note that you won't be able to use these unless you have a card that handles them, like a Radeon or GeForce. I've also added two routines for dealing with extensions: `gl_supportsExtension` and `gl_enableExtension` (see the *GL extensions* section further down in this document).
  - Added `gl_getVersion` to get the version number of your OpenGL implementation.
- GLU stuff:
  - Added `gluOrtho2D`.
- Main:
  - Started working on a simplified event model á la GLUT.
  - Added four new example programs: `anisotropic.exw` shows how to do anisotropic texture filtering. `fireworks.exw` shows a simple fireworks effect. `multitexture.exw` shows how to do (single-pass) multi-texturing using the `GL_EXT_multitexture` extension. `vol_fog.exw` shows how to get volumetric fog using the `GL_EXT_fog_coord` extension.

### 030130

- GL stuff:
  - Added `glCopyTexImage2D`, `glGetFloatv`, `glLightModelf`, `glRasterPos2f`, and `wglUseFontBitmaps`.
- Main:
  - Added four new example programs: `bitmap_font.exw` shows how to draw (2D) text. `cel_shade.exw` demonstrates basic cel shading. `radial_blur.exw` does a fake radial blur effect on a rotating spring. `tex_blend.exw` shows how to do alpha blending on a texture-mapped object.

### 030125

- GL stuff:

- Added `glBitmap`, `glCallLists`, `glListBase`, `glPixelStorei`, `glRasterPos2i`, `glTexGeni` and `wglUseFontOutlines`.
- GLU stuff:
  - Added `gluDeleteQuadric`, `gluLookAt`, `gluNewQuadric`, `gluNewTess`, `gluOrtho2D`, `gluQuadricDrawStyle`, `gluQuadricNormals`, `gluQuadricOrientation`, `gluQuadricTextures`, `gluScaleImage`, `gluSphere`, `gluTessBeginContour`, `gluTessBeginPolygon`, `gluTessEndContour`, `gluTessEndPolygon`, `gluTessCallback`, `gluTessProperty` and `gluTessVertex`.
- Main:
  - Added the ability to set fullscreen video mode. Just set `euglDisplayMode` to `EUGL_FULLSCREEN` and set `euglPFD[EUGL_COLORBITS]` to the desired number of bits per pixel before calling `EuGLMain`. See `scroller.exw` (far bottom) for an example.
  - Added six more example programs: `3d_text.exw`, `bounce.exw`, `drawf.exw`, `quadrics.exw`, `scroller.exw` and `sphere_tex.exw`.

### 030124

- GL stuff:
  - Added `glColor4f`, `glPointSize` and `glTexEnvf`.
- GLU stuff:
  - Added `gluBeginPolygon`, `gluBeginSurface`, `gluBeginTrim`, `gluBuild2DMipmaps`, `gluCylinder`, `gluDeleteNurbsRenderer`, `gluDisk`, `gluEndCurve`, `gluEndPolygon`, `gluEndSurface`, `gluEndTrim`, `gluNewNurbsRenderer`, `gluNurbsProperty`, `gluNurbsSurface`, `gluPerspective` and `gluPwlCurve`.
- Main:
  - Added 3 new example programs: `nurbs.exw`, which shows how to use NURBs and trimming, `circles.exw`, which shows texture mapping, mipmapping and texture modulation. And finally `tunnel.exw` which basically shows the same things as `circles.exw` - I just included it cos it looks cool.

### 030122

- Main:
  - Got rid of `eugl.dll`, it's not needed anymore.
  - Cleaned up the example programs, added some comments, etc.
  - Added a new example program: `dragnet2k`, which shows a lit animated surface.
  - I've continued working on the "abstraction layer" of OpenGL/GLU functions (i.e. spare the user from having to use `c_proc/c_func` calls). The most commonly used functions have now been given Euphoria-equivalents. So instead of calling e.g. `c_proc(glVertex3f, {0, 0, 0})` you can call `gl_vertex3f({0, 0, 0})` (notice the small 'v' in the Euphoria procedure!). The example programs all use this method of calling the OpenGL functions, so you can look at them if you run into trouble.

### 000315

- GL stuff:
  - Added about 60-70 functions:

- Just about all `glVertex`-functions are now linked and ready for use. The same goes for `glTexCoord`.
- GLU stuff:
  - -(still no NURBS or tessellators..)
- GLAUX stuff:
  - -
- Main:
  - Added one example program. It doesn't really show anything that the other examples doesn't use, but I just felt like porting it..
  - Added a whole bunch of stuff to `EuGLView`:
    - Can now load `3DStudio .asc` files for viewing.
    - Ability to rotate images/meshes about all 3 axes (not at the same time though..).
    - Fog option.
    - Perspective-correction option (only affects images when filtering is turned off).
  - Work has begun on writing an additional layer of code to let the user call OpenGL functions as a Euphoria-routine (without `"c_proc"/"c_func"`).
  - So far only a few routines have been added (`glBegin`, `glEnd` and some others). These routines are prefixed with `"eu"`, so `c_proc(glEnable, { GL_LIGHTING })` becomes `euglEnable(GL_LIGHTING)`.

### 000308

- GL stuff:
  - Texture-mapping.
  - Display lists.
  - Fog.
  - .. + lotsa other things.
- GLU stuff:
  - One (!) glu function was added (`gluPerspective`).
- GLAUX stuff:
  - -
- Main:
  - Changed interface somewhat.
  - Added 3 more examples.
  - Added lots of constants + functions in `ewin32api.ew`.
  - Went from C to asm in the dll and saved a couple of kB:s.

## What is this ?

EuGL is an interface for writing OpenGL applications in Euphoria to be run on a Win32 platform.

## Usage

### Basics

I haven't had the time to write any OpenGL documentation yet, so you'll have to know OpenGL already in order to get anything out of EuGL. For those of you who *do* know OpenGL; here's the deal:

All routines (and constants) have kept their original names, so if this is the C code:

```
glBegin( GL_TRIANGLES );
```

then the Euphoria code would be:

```
c_proc(glBegin, { GL_TRIANGLES })
```

or, if you prefer it:

```
gl_begin( GL_TRIANGLES )
```

Not too difficult..

The big problem is that not all functions have been wrapped yet. Actually, only a subset of the most common functions is included so far (I you wish to know which functions are included; look at the bottom of gl.ew).

## Creating an application with EuGL

First of all, you need to include eugl.ew. You'll also need to have gl.ew, glu.ew, glaux.ew and ewin32api.ew (this particular version – older ones won't do) in the directory of your program, or in \Euphoria\include\.

Basically, you have two choices: either you write your own WndProc (i.e. an event handler), or you let EuGL handle that and pass on certain events to your program. If you choose the first method you start up your program by calling EuGLMain(integer WndProcID, sequence title, integer width, integer height), where WndProcID is the routine\_id of your WindowProc. If you choose the latter method you exchange the WndProcID parameter in the call the EuGLMain with the constant EUGL\_HANDLE\_EVENTS. You also need to set up some routines for handling those events that are of interest for you. For example:

```
procedure init()  
    -- do stuff  
end procedure  
  
euglInitFunc(routine_id("init"))
```

If you specify an "InitFunc", it will be called after the main window has been created, but before ShowWindow has been invoked. For a list of which events your program can receive, see appendix A.



You can change the members of the window-class (sequence `euglWCEX`) and pixelFormatDescriptor (sequence `euglPFD`) before calling `EuGLMain()`, if you wish to..

You can set the global variable `euglFlags` to `PFD_DOUBLEBUFFER` to make EuGL use double buffering. If you need the window-handle or device-context in some routine of yours, use `glhwnd` and `glhDC`. If you need a RECT-structure, use `glrect`.

All of this probably doesn't make any sense, so the best thing would be to check out the example programs

## GL extensions

There are a number of extensions available for OpenGL that enables features such as multi-texturing, anisotropic filtering, vertex shading and many other things. Some are vendor specific (ATI, NV, SGI), while other are supported by multiple vendors (EXT). Then there are those extensions that have been approved by the OpenGL Architecture Review Board and thereby been promoted to ARB extensions. EuGL supports the use of many of these extensions, but you must make sure that your video card actually supports the extensions you're trying to use. For this purpose I've added `gl_supportsExtension`. The syntax is as follows:

```
function gl_supportsExtension(sequence extension_name)
```

This is not a real OpenGL function but something I've added myself. It just retrieves the entire list of supported extensions and tries to match it against the `extension_name`. If there was a match it will return non-zero, and zero if there was no match. Once you've made sure that an extension is supported by your hardware you can load all the functions associated with that particular extension by calling `gl_enableExtension`:

```
procedure gl_enableExtension(sequence extension_name)
```

After you've enabled an extension you can call the functions associated with that particular extension just like other OpenGL functions. For example:

```
if not gl_supportsExtension("GL_EXT_point_parameters") then
    -- This isn't supported by hardware - report an error
else
    -- Enable the extension
    gl_enableExtension("GL_EXT_point_parameters")
    -- Now use one of the EXT_point_parameters functions
    gl_pointParameterfEXT(GL_POINT_FADE_THRESHOLD_SIZE_EXT, 1.0)
end if
```

There are two extensions that aren't real extensions but rather depend only on which version of OpenGL you have, namely `GL_VERSION_1_2` and `GL_VERSION_1_3`. To check if these are applicable you don't use `gl_supportsExtension` but `gl_getVersion`. An example:

```
if gl_getVersion() >= 1.2 then
    gl_enableExtension("GL_VERSION_1_2")
end if
```

Please note that not all extensions are available in this version of EuGL. An extension might be reported as supported but cause a crash or not working properly when you try to use it because it hasn't been wrapped yet. Those extensions that mainly have been wrapped in this version are ARB\_multitexture, ARB\_texture\_compression, ARB\_transpose\_matrix, EXT\_multitexture, EXT\_point\_parameters and EXT\_fog\_coord. Use `f` or `i` versions of the functions rather than `d`, and use scalar functions instead of those that operate on vectors.

## WGL extensions

To the WGL extensions belong those extensions that are specific to the windowing system. They deal with things such as rendering contexts, pixel buffers and swap intervals. Using WGL extensions in EuGL is similar to using GL extensions. The syntax for checking if a particular extension is supported is:

```
if wgl_supportsExtension(extension) then
    -- supported
else
    -- not supported
end if
```

And to enable a WGL extension you would type:

```
wgl_enableExtension(extension)
```

Please be careful to use `wgl_supports/enableExtension` with WGL extensions and `gl_supports/enableExtension` with GL extensions. Mixing one with the other could end up crashing your program.

## An example

A basic EuGL program could look something like this:

```
include eugl.ew

procedure init()
    -- initialize here.
end procedure

procedure draw()
    --draw stuff here.
end procedure

procedure close()
    -- free resources here.
end procedure

procedure key(integer keycode, integer x, integer y)
    if keycode = VK_ESCAPE then
        ewPostQuitMessage(0)
```

```
        end if
end procedure
```

```
euglFlags = PFD_DOUBLEBUFFER  -- Use double-buffering (optional)
                                -- You can also use other flags
```

```
-- If you want fullscreen, uncomment the next two lines.
-- euglPFD[EUGL_COLORBITS] = 16
-- euglDisplayMode = EUGL_FULLSCREEN
```

```
euglInitFunc(routine_id("init"))
euglExitFunc(routine_id("close"))
euglDisplayFunc(routine_id("draw"))
euglKeyboardFunc(routine_id("key"))
```

```
EuGLMain(EUGL_HANDLE_EVENTS, "Window title", 320, 240)
-- The above starts up the application. 320 and 240 are the width
-- and height of the window, respectively.
```

## What's done so far

- Pretty much all constants.
- Many GL-, and some GLU functions.
- Many GL extensions.
- All (?) WGL extensions.

## To do

- The rest of the GL functions.
- All GLU (and possibly some GLAux) functions.
- The rest of the GL extensions.
- Write a better interface (hide all `c_func/c_proc` calls from the user).
- Add more event handlers (VisibilityFunc, MenuFunc etc.).
- Write some decent documentation.

## Thanks to

- hardCode/Bizarre Creations, Blaine Hodge, Shadow, SGI, and others for OpenGL code/info.
- Todd Riggins, whose `ewin32api` I tweaked a bit for this release..
- Brian Broker, Jiri Babor & Monty King for pointing out an error to me.

## Appendix A – global routines

I'm not listing all the GL/GLU routines here, since there are so many of them. If you want to know which ones have been wrapped, look in the .ew files.

### eugl.ew

```
procedure EuGLMain(integer WndProcID, integer width, integer
                    height, sequence title)
```

This will start up the application for you. It changes the display settings (if `euglDisplayMode = EUGL_FULLSCREEN`), creates a window with the specified width, height and title (caption), calls `euglInit` (if specified), creates and specifies the pixel format to use, enters the window event loop, and finally calls `euglExit` (if specified). `WndProcID` is the `routine_id` of your `WndProc` (i.e. the event handler), or `EUGL_HANDLE_EVENTS` if you'd rather let EuGL handle the events and pass them on to your program if necessary.

```
procedure euglInitDisplayMode(integer mode)
```

Call this before `EuGLMain` to specify the desired display mode. `mode` can be a combination of the following flags:

```
EUGL_SINGLE
EUGL_DOUBLE
EUGL_WINDOWED
EUGL_FULLSCREEN
EUGL_BITS8
EUGL_BITS15
EUGL_BITS16
EUGL_BITS24
EUGL_BITS32
```

```
procedure euglInitWindowPosition(integer x, integer y)
```

Call this before `EuGLMain` to specify the initial position of the main window.

```
procedure euglSetWindowTitle(sequence title)
```

Used to change the title (caption) of the main window after it has been created.

```
procedure euglPostRedisplay()
```

Sends a `WM_PAINT` message to the main window.

```
procedure euglDisplayFunc(integer id)
```

`id` specifies the `routine_id` of a procedure that will be called when the window needs to be redrawn. The procedure should look like:

```
procedure myDisplayFunc()
```

```
procedure euglExitFunc(integer id)
```

`id` specifies the `routine_id` of a procedure that will be called when the application is shut down (i.e. after the message loop has finished). The procedure should look like:

```
procedure myExitFunc()
```

```
procedure euglInitFunc(integer id)
```

id specifies the routine\_id of a procedure that will be called when the application is starting up (i.e. right after the window has been created). The procedure should look like:

```
    procedure myInitFunc()
```

```
procedure euglKeyboardFunc(integer id)
```

id specifies the routine\_id of a procedure that will be called whenever a key is pressed.

The procedure should look like:

```
    procedure myKeyboardFunc(integer keycode, integer x,  
                             integer y)
```

where keycode is the ASCII code and x and y are the coordinates for the mouse position.

```
procedure euglMotionFunc(integer id)
```

```
procedure euglPassiveMotionFunc(integer id)
```

id specifies the routine\_id of a procedure that will be called whenever the mouse cursor is moved. The procedure should look like:

```
    procedure myMotionFunc(integer x, integer y)
```

The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed. The passive motion callback for a window is called when the mouse moves within the window while *no* mouse buttons are pressed.

```
procedure euglMouseFunc(integer id)
```

id specifies the routine\_id of a procedure that will be called whenever a mouse button is pressed or released. The procedure should look like:

```
    procedure myMouseFunc(integer button, integer  
                           state, integer x, integer y)
```

When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of EUGL\_LEFT\_BUTTON, EUGL\_MIDDLE\_BUTTON, or EUGL\_RIGHT\_BUTTON. For systems with only two mouse buttons, it may not be possible to generate EUGL\_MIDDLE\_BUTTON callback. For systems with a single mouse button, it may be possible to generate only a EUGL\_LEFT\_BUTTON callback. The state parameter is either EUGL\_UP or EUGL\_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the position of the cursor at the time of the event.

```
procedure euglReshapeFunc(integer id)
```

id specifies the routine\_id of a procedure that will be called when the window is resized. The procedure should look like:

```
    procedure myReshapeFunc(integer width, integer height)
```

```
procedure euglTimerFunc(integer msec, integer id, integer timerID)
```

id specifies the routine\_id of a timer callback to be triggered in at least msec milliseconds. The callback procedure should look like:

```
    procedure myTimerFunc(integer timerID)
```

The timerID parameter to the timer callback will be passed on to the callback. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. EuGL attempts to deliver the timer callback as soon as possible after the expiration of the

callback's time interval. If a timer callback is registered before EuGLMain is called, it will be called at the nearest timeout following after the main window has been created.

There is no support for canceling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.

### ewin32api.ew

`function ewBeginPaint(atom hwnd, atom ps)`

Calls the Windows API function BeginPaint.

`procedure ewCheckMenuItem(atom hmenu, atom idm, atom mask)`

Calls the Windows API function CheckMenuItem.

`function ewCreateFont(integer a, integer b, integer c, integer d,  
integer e, integer f, integer g, integer h,  
integer i, integer j, integer k, integer l,  
integer m, sequence n)`

Calls the Windows API function CreateFont.

`function ewCreateMenu()`

Calls the Windows API function CreateMenu.

`function ewCreatePopupMenu()`

Calls the Windows API function CreatePopupMenu.

`function ewDefWindowProc(atom hwnd, atom msg, atom wParam, atom  
lParam)`

Calls the Windows API function DefWindowProc.

`procedure ewDestroyMenu(atom hmenu)`

Calls the Windows API function DestroyMenu.

`procedure ewEndPaint(atom hwnd, atom ps)`

Calls the Windows API function EndPaint.

`function ewPostMessage(atom hwnd, atom msg, atom wParam, atom  
lParam)`

Calls the Windows API function PostMessage.

`procedure ewPostQuitMessage(integer status)`

Calls the Windows API function PostQuitMessage.

`function ewSelectObject(atom hdc, atom hgdibobj)`

Calls the Windows API function SelectObject.

`procedure ewSetWindowPos(atom hwnd, atom a, integer x, integer y,  
integer w, integer h, atom b)`

Calls the Windows API function SetWindowPos.

```
procedure ewSetWindowText(atom hwnd, sequence text)
```

    Calls the Windows API function SetWindowText.

```
function ewSwapBuffers(atom dc)
```

    Calls the Windows API function SwapBuffers.

## Appendix B – global variables and constants

### eugl.ew

`integer euglDisplayMode`

Controls the display mode. Can be either `EUGL_WINDOWED` (default) or `EUGL_FULLSCREEN`.

`integer euglDraw`

Should contain the `routine_id` of the routine you use to draw the scene. The routine must be a procedure (i.e. it cannot return anything) and it mustn't have any arguments.

`integer euglExit`

If you want to do something when the application is shutting down (e.g. freeing bitmaps), you can set this variable to the `routine_id` of a procedure that has no arguments.

`integer euglFlags`

If you want to customize the pixel format used (e.g. if you want double buffering), you can do that by altering this variable before calling `EuGLMain`.

`integer euglInit`

If you want to do some initialisation (e.g. freeing bitmaps), you can set this variable to the `routine_id` of a procedure that has no arguments.

`atom dmScreenSettings`

Points to a `DEVMODE` structure. The memory isn't allocated until `EuGLMain` is called.

`atom funcval`

A dummy variable used to hold unimportant values returned by function.

`atom glhDC`

Contains a handle to the main window's device context. Created by `EuGLMain`. This variable is valid at the point where `euglInit` is called.

`atom glhwnd`

Used to hold the handle of the main window. This variable is set after the main window has been created, so it's valid at the point where your "InitFunc" is called.

`atom glrect`

Points to a `RECT` structure. The memory isn't allocated until `EuGLMain` is called.

`atom hInst`

Holds the current instance.

`atom pfd`

Points to a `PIXELFOMATDESCRIPTOR` structure. The memory isn't allocated until `EuGLMain` is called.

`atom ps`



Points to a PAINTSTRUCT structure. The memory isn't allocated until `EuGLMain` is called.

`atom wc`

Points to a WNDCLASSEX structure. The memory isn't allocated until `EuGLMain` is called.

`sequence euglWCEX`

Specifies a window class. When `EuGLMain` creates the application window, the contents of this sequence is poked into the memory pointed to by `wc`.

`sequence euglPFD`

Specifies a pixel format descriptor. When `EuGLMain` creates the application window, the contents of this sequence is poked into the memory pointed to by `pfd`.

`EUGL_HANDLE_EVENTS (= -2)`

Tells EuGL to use its own event handler.

`EUGL_COLORBITS (= 5)`

`EUGL_DEPTHBITS (= 19)`

Indexes into the `euglPFD` sequence.

`EUGL_CURSOR (= 8)`

`EUGL_ICON (= 7)`

`EUGL_SMALLICON (= 12)`

`EUGL_WINDOWCOLOR (= 9)`

Indexes into the `euglWCEX` sequence.

`EUGL_WINDOWED (= 0)`

`EUGL_FULLSCREEN (= 1)`

Display mode enumerators.

`EUGL_SINGLE (= 0)`

`EUGL_DOUBLE (= 2)`

Specifies single- or double-buffered mode.

`EUGL_BITS8 (= 4)`

`EUGL_BITS15 (= 8)`

`EUGL_BITS16 (= 16)`

`EUGL_BITS24 (= 32)`

`EUGL_BITS32 (= 64)`

Bit count enumerators.

`EUGL_LEFT_BUTTON (= 1)`

`EUGL_RIGHT_BUTTON (=2)`

`EUGL_MIDDLE_BUTTON (=4)`

Mouse button enumerators.

`EUGL_UP (= 0)`

`EUGL_DOWN (= 1)`

Mouse button states.

## ewin32api.ew

```
PFD_DOUBLEBUFFER (= 1)
PFD_STEREO (= 2)
PFD_DRAW_TO_WINDOW (= 4)
PFD_DRAW_TO_BITMAP (= 8)
PFD_SUPPORT_GDI (= #10)
PFD_SUPPORT_OPENGL (= #20)
PFD_GENERIC_FORMAT (= #40)
PFD_NEED_PALETTE (= #80)
PFD_NEED_SYSTEM_PALETTE (= #100)
PFD_SWAP_EXCHANGE (= #200)
PFD_SWAP_COPY (= #400)
PFD_SWAP_LAYER_BUFFERS (= #800)
PFD_GENERIC_ACCELERATED (= #1000)
PFD_TYPE_RGBA (= 0)
PFD_TYPE_COLORINDEX (= 1)
PFD_MAIN_PLANE (= 0)
PFD_OVERLAY_PLANE (= 1)
PFD_UNDERLAY_PLANE (= -1)
```

These are all pixel format descriptor flags.

/Mic, 2003

[stabmaster@hotmail.com](mailto:stabmaster@hotmail.com)

<http://www.cyd.liu.se/~micol972/site>