YZ Windows 0.3.5

http://yz-windows.sourceforge.net

API Documentation and Specification

Woodley Packard

1. Introduction

i. Abstract

The YZ Windowing System is an API (Application Programming Interface) to the management of windows and the drawing of graphics on personal computers. It can be used in a way similar to Xlib, the MacOS toolbox, the MS Windows API, or any number of other interfaces — however, YZ is constructed to be a a front end for a lower level windowing system (although it can run as the lowest level window management provider as well under certain environments, specifically Linux with framebuffer and devfs support). It is in some ways comparable to toolkits such as GLUT, which provide a uniform API to windowing across all supported platforms. However, YZ [will be] more flexible than GLUT, being designed not as a quick-and-dirty hack to get an OpenGL context, but as a more complete and usable windowing system.

ii. Supported platforms

Currently, MacOS is the only platform with a conforming libyz 0.3.5. There is a version of YZ to run on the Linux Framebuffer which is fairly close to 0.3.5. It is also likely that in the [near?] future libyz will be available on some other platforms. Thus, a program written for the YZ API should [theoretically] be able to compile on any supported platforms without change to source code.

There is also a project to create Java bindings for YZ - for more information about YZ for Java (JYZ), see http://jyz.sourceforge.net.

For more information about YZ for Linux Framebuffer (KYZ) or YZ for MacOS (MacYZ), see http://yz-windows.sourceforge.net.

Some small steps have been taken in the direction of a libyz for X Windows, too (XYZ).

iii. Conventions

This document describes the functionality of version 0.3.4 the YZ Windowing API. It is broken down into three sections: this introduction, the specification, and a few examples. Sections are numbered 1, 2, 3, and subheadings are numbered i, ii, iii, iv, etc. Code listings in standard C appear in indented monotype font, like this:

```
void yzSomeFunctionWhichDoesntExist1i(int      param);
```

In general, the end of the name of a graphics function signifies the arguments it takes. For example, the function yzLine2i2i() takes two pairs of integers, while the function yzPenColor3i() takes three integers. However, the names of windowing functions do not follow this rule; for example, to select a window for drawing, you simply use yzSelectWindow().

As of YZ Windows 0.2.0, the graphical functions are also available as plain names, e.g. yzLine(). These aliases are defined as macros in the relevant header files.

iv. Conformant Implementations

A “conformant implementation” is one which provides every function defined in this specification. An implementation may choose to always return an error from any functions marked “Optional”, thus not truly implementing them. Such functions shall be called “fake implementation functions”; however, it is essential that every function be available in some form in a “conformant” implementation, so that link errors do not occur. Any program written for the YZ API should successfully link to a “conformant implementation”. An implementation may be called “complete” if it uses no fake implementation functions.

v. Definitions

A “window” is user-distinguishable area on the screen, usually surrounded by some border and with a title on top, into which the owning application program can draw graphics.

Each window has associated with it a current “pen”, which describes the color of graphics to be drawn, the size of lines to be drawn, and possibly the starting location of the next line (called the pen’s location).

A “pointer” is the combination of a physical mouse and an on-screen cursor. When a given window has a pointer’s focus (i.e. the pointer is within the window), the owning application can query the pointer for location, clicks, etc, and can change its shape or hide it.

A “font” is a collection of shapes of characters which can be used to render text into a window. A given incarnation of a font has associated with it a size and attributes such as bold or italics.

An “event” is an action performed by the user which a program may respond to. An event has a type, such as “mouse click,” and data, such as “at 201, 73.” The data associated with an event will vary according to its type.

2. Specification

i. Data Structures

```
typedef    struct      color_t
{
     unsigned short    r, g, b, a;       // red, green, blue, and alpha values
}    color_t;

typedef    struct      pen_t
{
     unsigned short    x, y;             // where next draw will take place if not specified
     unsigned short    size;             // width of next line to be drawn
     color_t                color;       // color of next graphic to be drawn
}    pen_t;

typedef    struct      cursor_t
{
     color_t     *data;                  // width x height array of cursor information
     int         width, height, hotx, hoty;    // width and height of the cursor,
}    cursor_t;                                  // and also where in the bitmap the hotspot is
```

```
typedef    struct      event_t
{
     int   type;
     int   button, x, y;     // which button the click was, and where it was
     int   mouse, x2, y2;    // which mouse it was, and where the drag ended
     int   key;              // the ascii code of the key pressed, or a special key code
     // this structure will expand as more types of events are added:
     int   unused[57];
}    event_t;

typedef    struct      font_t
{
     char  name[128];        // the name-string of the font. either “system” or “fixed”.
     int   size;             // height of a character. unused.
     int   attr;             // bold, italics, etc. unused.
     void  *impl_data;       // each implementation will store fonts in its own way
}    font_t;

typedef    struct      window_t
{
     unsigned long     wseq;             // window ID number
     unsigned long     res1;             // unused
     char              name[64];         // the window’s name
     unsigned short    width, height;    // the window’s height and width

     struct      pen_t *pen;             // the current pen
     struct      font_t *font;           // the current font

     /* implementation specific data may be stored here */
     /* possibly including an event queue */
     void              *impl_data;
}    window_t;
```

ii. Windowing Functions

```
window_t   *yzNewWindow(char *name, unsigned short   width, unsigned short   height);
```

Creates a new window. The window should be entitled by the first parameter, and have width and height as given by the second and third parameters.

Returns a pointer to the window created on success.

Returns NULL (0) on failure.

```
int        yzDeleteWindow(window_t *window);
```

Deletes the window specified as the parameter, erasing it from the screen if necessary. If the specified window is currently selected, the selection is changed to NULL.

Returns 0 on success.

Returns -1 on failure (if, for example, the parameter is not a valid window).

```
window_t   *yzSelectWindow(window_t      *window);
```

Selects the specified window for all future drawing operations until another window is selected.

Returns a pointer to the previously selected window on success (or NULL if no window was previously selected).

Returns NULL on failure (if, for example, the parameter is not a valid window).

iii. Pointer Functions

```
int        yzQueryPointer(int      *x, int     *y);
```

Queries the state and location of the pointer. On success, the x location of the pointer (as measured from the upper left corner of the currently selected window) is stored in the location pointed to by “x”, and the y location is stored in the location pointed to by “y”.

If “x” is NULL, no value is stored in it; similarly, if “y” is NULL, no value is stored in it. These “failures” do not affect the overall success or failure of the function call.

yzQueryPointer() fails and returns -1 if the pointer is not within the currently selected window, i.e. the window does not have the focus (or there is no selected window). Otherwise, yzQueryPointer returns values as follows:

If the pointer has not moved since the last call to yzQueryPointer(), and the mouse button has not changed state, the return value is 0.

Otherwise if the mouse has moved, but the button state has not changed, the return value is 1.

Otherwise, if the mouse button went down, the return value is 2.

Otherwise, the mouse button went up and the return value is 3.

```
int        yzHideCursor();
```

Attempts to hide the cursor. Fails if the pointer is not within the selected window (i.e. the window does not have the focus). On success, 0 is returned. Otherwise, -1 is returned.

```
int        yzShowCursor();
```

Attempts to show the cursor. Fails if the pointer is not within the selected window (i.e. the window does not have the focus). On success, 0 is returned. Otherwise, -1 is returned.

```
int        yzSetCursor(cursor_t    *cursor);
```

Attempts to reshape the cursor. Fails if the pointer is not within the selected window (i.e. the window does not have the focus). On success, 0 is returned and the cursor’s shape changes to that specified by the parameter. Otherwise, -1 is returned.

Note that an implementation need not take into account all the data provided by the cursor_t structure. Indeed, most cursor drivers support only a very limited range of colors, or even only black-and-white. In these cases, the implementation should simply provide as close an approximation to the requested cursor as possible.

```
int             yzWriteCursor(cursor_t  *crsr, char *filename);
```

Optional.

Writes the specified cursor structure to a file named “filename”. The format of the file is as follows: 2 bytes of width, 2 bytes of height, 2 bytes of hotx, 2 bytes of hoty, then width*height*sizeof(color_t) bytes of bitmap data. Byte-ordering should be big-endian.

Returns 0 on success.

Returns -1 on failure (for example, if the file cannot be created).

```
cursor_t   *yzNewCursor(int  width, int  height);
```

Creates a blank new cursor structure with size width*height. The hotx and hoty components should be initialized to width/2 and height/2, respectively.

Returns the address of the newly created cursor structure on success.

Returns NULL on failure (for example, insufficient memory available).

```
cursor_t   *yzLoadCursor(char      *filename);
```

Optional.

Allocates a cursor structure and reads a cursor in from the file “filename”. The format is identical to that described under “yzWriteCursor()”.

Returns the address of the newly allocated cursor structure on success.

Returns NULL on failure (for example, the file didn’t exist).

iv. Drawing Functions

Note that in usually the return values of these functions will not be checked.

```
int  yzLine2i2i(int    x1, int     y1, int     x2, int     y2);
#define    yzLine      yzLine2i2i
```

Draws a line in the selected window from point (x1, y1) to point (x2, y2), using the color and size specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int  yzRect2i2i(int    x1, int     y1, int     x2, int     y2);
#define    yzRect      yzRect2i2i
```

Draws a solid rectangle in the selected window with left edge x1, right edge x2, top edge y1, and bottom edge y2, in the color specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int  yzOutlineRect2i2i(int   x1, int     y1, int     x2, int     y2);
#define    yzOutlineRect     yzOutlineRect2i2i
```

Draws the outline of a rectangle in the selected window with left edge x1, right edge x2, top edge y1, and bottom edge y2, in the color specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int  yzClear();
```

Paints the entire selected window with the color specified in the selected window’s pen structure. Returns 0 on success and -1 on failure.

```
int  yzPixel2i1c(int   x, int      y, color_t  c);
#define    yzPixel     yzPixel2i1c
```

Plots a single pixel at point (x, y) in the selected window, using the color specified by the 3rd parameter “c”.

Returns 0 on success and -1 on failure.

```
int  yzPixel2i3i(int   x, int      y, int      red, int    green, int  blue);
```

Plots a single pixel at point (x, y) in the selected window, using the color specified by the parameters “red”, “green”, and “blue”.

Returns 0 on success and -1 on failure.

```
int  yzPenColor1c(color_t    c);
```

Sets the color entry in the pen structure of the selected window to be equal to “c”.

Returns 0 on success and -1 on failure.

```
int yzPenColor3i(int   r, int      g, int      b);
#define    yzPenColor  yzPenColor3i
```

Sets the red, green, and blue entries of the color entry in the pen structure of the selected window to be “r”, “g”, and “b”, respectively. Sets the alpha entry to 0xFFFF.

Returns 0 on success and -1 on failure.

```
int yzPenColor4i(int   r, int      g, int      b, int      a);
```

Sets the red, green, blue, and alpha entries of the color entry in the pen structure of the selected window to be “r”, “g”, “b”, and “a”, respectively.

Returns 0 on success and -1 on failure.

```
int yzPenSize1i(int    s);
#define    yzPenSize   yzPenSize1i
```

Sets the diameter in pixels of the pen used to draw lines, unfilled circles, etc.

Returns 0 on success and -1 on failure.

```
int yzCircle3i(int     x, int      y, int      r);
#define    yzCircle          yzCircle3i
```

Draws an [unfilled] circle, centered at (x,y) with radius r, using the color and size specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int yzFillCircle3i(int x, int      y, int      r);
#define    yzFillCircle      yzFillCircle3i
```

Draws a filled circle, centered at (x,y) with radius r, using the color specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int yzOval4i(int x, int      y, int      rx, int     ry);
#define    yzOval      yzOval4i
```

Draws an [unfilled] oval, centered at (x,y) with radius rx in the horizontal direction and ry in the vertical direction, using the color and size specified in the selected window’s pen structure.

Returns 0 on success and -1 on failure.

```
int yzFillOval4i(int   x, int      y, int      rx, int     ry);
#define    yzFillOval  yzFillOval4i
```

Draws a filled oval, centered at (x,y) with radius rx in the horizontal direction and ry in the vertical direction, using the color specified in the selected window's pen structure.

Returns 0 on success and -1 on failure.

```
int yzOvalInRect2i2i(int     x1, int     y1, int     x2, int     y2);
#define    yzOvalInRect      yzOvalInRect4i
```

Draws an [unfilled] oval fitting just inside the box described by (x1,y1) (x2,y2), using the color and size specified in the selected window's pen structure.

Returns 0 on success and -1 on failure.

```
int yzFillOvalInRect2i2i(int x1, int     y1, int     x2, int     y2);
#define    yzFillOvalInRect  yzFillOvalInRect4i
```

Draws a filled oval fitting just inside the box described by (x1,y1) (x2,y2), using the color specified in the selected window's pen structure.

Returns 0 on success and -1 on failure.

v. Bitmap functions

```
int  yzDisplayBits5i1p(int   x, int      y, int dx, int dy, int  rb, void    *bits);
#define    yzDisplayBits     yzDisplayBits5i1p
```

Copies dx by dy bits of the bitmap image starting at location "bits" with rowbytes parameter "rb" into the selected window at location x, y. Note that no conversion is performed based on the bitdepth. The supplied bitmap is assumed to be in the same bitdepth as the framebuffer device. To find the bitdepth of the framebuffer, use yzGetDepth().

Returns 0 on success and -1 on failure.

```
int  yzGetDepth()
```

Returns the bitdepth of the framebuffer containing the selected window, or -1 on failure.

vi. Text functions

```
int        yzSelectFont1s(char     *font_name);
#define    yzSelectFont      yzSelectFont1s
```

Select the font named in the parameter "font_name" as the active font for the selected window. The format of the name has not been determined yet; however, two aliases are available at the present: "system" selects the system's default font in a 12-point size, and "fixed" selects a mono-spaced font in a 12-point size.

Returns 0 on success and -1 on failure (e.g. font not found).

```
int       yzText2i1s(int    x, int      y, char     *ptr);
#define   yzText      yzText2i1s
```

Renders ASCII characters from the font specified by the selected window's "font" structure from the string "ptr" until a 0 is encountered, starting at horizontal position x with the baseline at y. The characters are drawn in the color in the current window's pen structure.

Returns 0 on success and -1 on failure.

```
int       yzText2i1p1i(int  x, int      y, char     *ptr, int   len);
```

Renders "len" ASCII characters from the font specified by the selected window's "font" structure from the string "ptr," starting at horizontal position x with the baseline at y. The characters are rendered in the color specified by the current window's pen structure.

Returns 0 on success and -1 on failure.

```
int       yzLetterSize(int  letter);
```

Returns the width, in pixels, of the specified ASCII character in the selected font. The return value is unspecified if no window is selected.

```
int       yzStringSize(char *ptr, int   length);
```

Returns the width, in pixels, of the first "length" ASCII characters pointed to by "ptr" in the selected font. The return value is unspecified if no window is selected.

vii. Event functions

The following constants represent the possible values of the "type" field of an event_t:

```
enum
{
     YZ_IDLE     =     0,    // nothing happened
     YZ_KEY_DOWN =     1,    // a key was pressed; "key" is filled out.
     YZ_KEY_UP   =     3,    // a key was released; "key" is filled out.
     // mouse events - each of these fills out "button" and "mouse", too:
     YZ_MOUSE_DOWN =   6,    // the mouse was pushed; "x," and "y" are filled out.
     YZ_MOUSE_DRAG =   7,    // drag while clicked; "x," "x2," "y", "y2" `````
     YZ_MOUSE_UP =     8,    // mouse click ended; "x," and "y" are filled out.
     // focus events
     YZ_WINDOW_ACTIVATE = 20,      // this window has kbd focus (i.e. was raised to top)
     YZ_WINDOW_DEACTIVATE = 21     // this window lost kbd focus (i.e. was sent from top)
};
```

The following constants represent special values of the "key" field of an event_t. If the key in question has an ASCII representation, its ASCII value is placed in the "key" field. Otherwise, if it appears in the following list, the corresponding value is used. Otherwise, YZ_KEYCODE_UNKNOWN is used.:

```
enum
{
     YZ_KEYCODE_LEFT_ARROW   =     255 + 1,
     YZ_KEYCODE_RIGHT_ARROW  =     255 + 2,
     YZ_KEYCODE_DOWN_ARROW   =     255 + 3,
     YZ_KEYCODE_UP_ARROW     =     255 + 4,
```

```
    YZ_KEYCODE_CONTROL      =     255 + 5,
    YZ_KEYCODE_SHIFT        =     255 + 6,
    YZ_KEYCODE_OPTION       =     255 + 7,    // these two are the same
    YZ_KEYCODE_ALT          =     255 + 7,

    YZ_KEYCODE_UNKNOWN = 65536
};


int       yzAreEventsWaiting();
```

Returns the number of events for the selected window that are waiting for handling in the event queue. If no window is selected, returns -1.

```
event_t   yzGetEvent(int    waitfor);
```

Returns the next event off of the selected window's event queue. If the queue is empty, the function waits up to "waitfor" milliseconds before returning a YZ_IDLE event. If at any point during that wait an event becomes available, that event is returned immediately instead.

```
int       yzQueryKey(int    key);
```

Returns -1 if the selected window is not currently focused. Otherwise, returns 1 if the key indicated is pressed, and 0 if it is not pressed. See above for an explanation of the key parameter (it's the same as the "key" field in an event_t).

3. Examples

Initialization

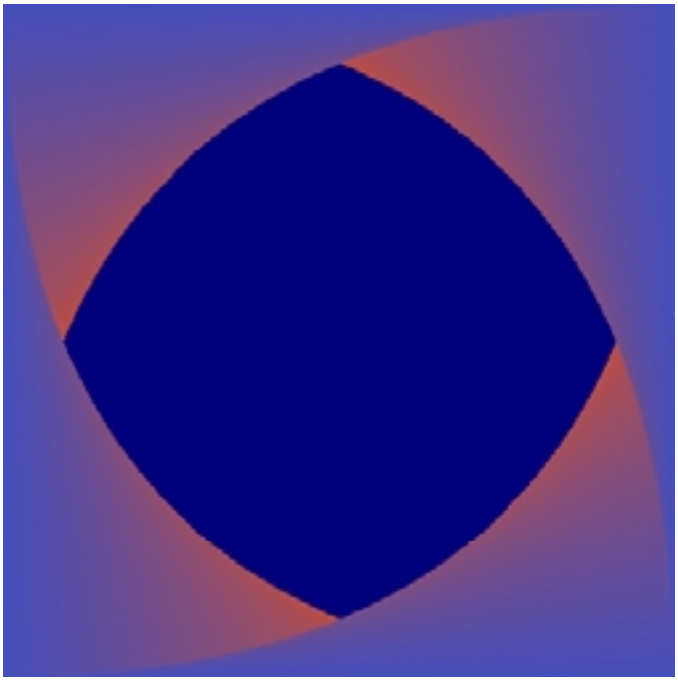
All the examples that follow assume the inclusion of the following few lines of code at the beginning:

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

#include    <yz/window.h>
#include    <yz/draw.h>

window_t    *the_window;

initialize()
{
    the_window = yzNewWindow("example", 400, 400);
    if(!the_window || yzSelectWindow(the_window))
    {
        fprintf(stderr, "unable to create or select a new window. exiting.\n");
        exit(-1);
    }
    yzPenColor(0, 0, 32000);    //paint the window dark blue
    yzRect(0, 0, 400, 400);
}
```



swoosh.c

i. swoosh.c

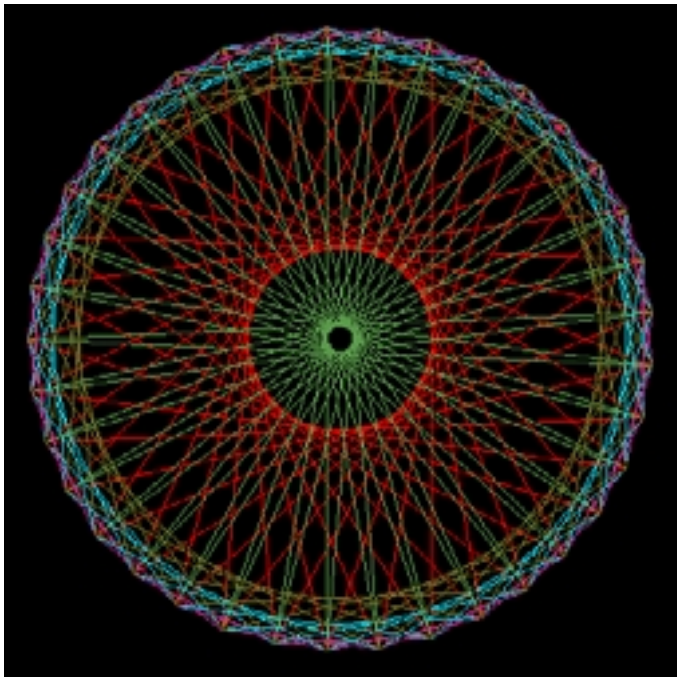
The following program draws a blue and red “swoosh” gradient (left, top):

```
main()
{
	int i;

	initialize();

	for(i=0;i<400;i++)
	{
		yzPenColor(65535 - 120*i,
			20000, 120*i);
		yzLine(i, 0, 0, 400 - i);
		yzLine(400 - i, 400, 400, i);
		yzLine(400 - i, 400, 0, 400 - i);
		yzLine(i, 0, 400, i);
	}

	/* wait until the mouse clicks
		to dismiss */
	while(yzQueryPointer(0, 0) < 2) { }
}
```



stringart.c

ii. stringart.c

This program runs the string-art algorithm to create neat patterns of lines - it takes parameters on the command-line (left, bottom):

```
#define	POINTS	37

main(int	argc, char	*argv[])
{
	int	i, j, k, n = 0;
	int	*inc;
	float	xp[POINTS], yp[POINTS], t;

	initialize();
```

```
        for(i=0;i<POINTS;i++)
        {
                t = ((float)i)/POINTS * (3.1415926 * 2);
                xp[i] = 200 - 180*cos(t);
                yp[i] = 200 - 180*sin(t);
        }
        if(argc <= 1)
        {
                inc = malloc(sizeof(int) * (n = 1));
                inc[0] = 1;
        }
        else
        {
            inc = malloc(sizeof(int) * (n = (argc - 1)));
            for(i=0;i<n;i++)
            {
                 inc[i] = atoi(argv[i+1]);
                 if(inc[i] == 0)
                 {
                     fprintf(stderr, "warning: argument %d is not a valid number\n", i+1);
                     inc[i] = 1;
                 }
            }
        }
        for(k=0;k<n;k++)
        {
                i = 0; j = -1;
                yzPenColor(65535 + 23847 * k, 39482 * k, 18734 * k);
                while(j != 0)
                {
                        j = (i + inc[k]) % POINTS;
                        yzLine(xp[i], yp[i], xp[j], yp[j]);
                        i = j;
                }
        }
        while(yzQueryPointer(0, 0) < 2) { }
}
```

iii. Where to find more

For more examples, see the Applications section of the YZ-Windows website:
http://yz-windows.sourceforge.net/apps/