pxCore – Portable Framebuffer and Windowing Library

Reference Manual

Framebuffer Support

Class pxBuffer

This class is used to describe a framebuffer that is stored linearly in memory as an array of 32bit integers. The set* APIs can be used to set the pxBuffer to describe how framebuffer in memory is laid out.

Methods

```
void* base() const;
void setBase(void* p);
```

These accessors can be used to get and set the base memory address for the described framebuffer.

```
int width()  const;
void setWidth(int width);
```

These accessors can be used to get and set the width of the framebuffer in pixels.

```
int height() const;
void setHeight(int height);
```

These accessors can be used to get and set the height of the framebuffer in pixels.

```
int stride() const;
void setStride(int stride);
```

These accessors can be used to get and set the width of the framebuffer's scanlines in **bytes**.

```
bool upsideDown() const;
void setUpsideDown(bool upsideDown);
```

These accessors can be used to specify whether scanlines of the framebuffer are "flipped" or not. If this is false then scanline(0) is scanline nearest to the base address; otherwise it is the furthest.

```
inline pxPixel *scanline(int line) const;
```

This method can be used to retrieve a pointer to a scanline which is a packed array of pxPixel(s).

```
inline pxPixel *pixel(int x, int y);
```

This method can be used to randomly access any pixel in the framebuffer based on an (x,y) coordinate.

```
pxRect bounds() const;
```

This method returns the bounds of the framebuffer as a pxRect.

```
void fill(const pxColor& color);
```

This method fills the pxBuffer with the specified pxColor

```
void fillAlpha(unsigned char alpha);
```
This method fills the alpha channel of the pxBuffer with the specified alpha value.

Class pxOffscreen

Methods

```
pxError init(int width, int height);
```
This method will initialize the offscreen for the given width and height but will leave the pxPixel values of the framebuffer untouched. If this method is called again term is called internally to free any used resources.
[Note: One of the init methods must be called prior to using the offscreen.]*

```
pxError initWithColor(int width, int height, const pxColor& color);
```
This method will initialize the offscreen for the given width and height and will initialize all of the pixels to the color value provided. If this method is called again term is called internally to free any used resources.
[Note: One of the init methods must be called prior to using the offscreen.]*

```
pxError term();
```
This method frees any resources used by the offscreen. If this is called then one of the init* methods must be called prior to using the offscreen again.

```
void blit(pxSurfaceNative s, int dstLeft, int dstRight,
              int dstWidth, int dstHeight,
              int srcLeft, int srcRight);
inline void blit(pxSurfaceNative s)
```
These methods can be used to blit (draw) the offscreen to the provided native surface descriptor.

Type pxSurfaceNative
This type is a portable alias to a platform specific drawing handle.
[Note: If you wish to do platform specific drawing within a window you should look at the definition of this type within pxOffscreenNative.h for the given platform.]

Windowing and Event Loop Support

Class pxWindow

Methods

```
pxError init(int left, int top, int width, int height);
```
This method must be invoked prior to calling other methods on this class. This method is the one that actually creates the platform window associated with this pxWindow instance.
[Note: A pxWindow is always created in a hidden state. You can call setVisibility to make the window visible after calling the init method.]

pxError term();
pxError close();
These methods can be used to free the resources associated with this window and have the effect of closing the window.

bool visibility();
void setVisibility(bool visible);
These accessors can be used to get and set the visible state of a pxWindow.

void invalidateRect(pxRect* r = NULL);
This method can be used to mark a rectangular region of the window as being dirty. This will cause the onDraw method to be invoked at some time in the near future so that the windows contents may be redrawn.

void setTitle(char* name);
This method can be used to set the title that typically shows up in the window's titlebar.

pxError setAnimationFPS(long fps);
This method can be used to enable or disable an animation timer event (onAnimationTimer) that is invoke *fps* times per second. A value of zero can be passed in to disable the timer event.

Event Methods
The following methods are meant to be overridden and for the basis of the event handling mechanism of pxWindow. These methods will be called on your subclass in response to the corresponding platform event.

virtual void onCreate();
This event will get fired after the init method has been invoked and after the platform window has been associated with the pxWindow instance.

virtual void onCloseRequest();
This event will get fired when a request to close the window has been received. Typically in response to someone click on the "close box" of the window.

virtual void onClose();
This event will get fired when the underlying platform window is being closed (or destroyed).

virtual void onSize(int w, int h);
This event will be fired when the window has changed its size. The width and height values that are passed in are the dimensions of the new "client area" of the window.

virtual void onMouseDown(int x, int y, unsigned long flags);
This event will be fired whenever the mouse has been pressed within the bounds of the client area of the window. The x, y coordinates are relative to the upper left corner of the window's client area.
[Note: All mouse events are routed to this window instance while the mouse remains down even if the mouse is outside of the windows' client area.

```
virtual void onMouseUp(int x, int y, unsigned long flags) {}
```
This event will be fired whenever a mouse button has been released for this window. The x, y coordinates are relative to the upper left corner or the window's client area.

```
virtual void onMouseMove(int x, int y) {}
```
This event is fired as the mouse moves with respect to this window. The x, y coordinates are relative to the upper left corner of this window's client area.

```
virtual void onMouseLeave() {}
```
This event is fired as the mouse leaves the client area of the window. This is useful to reliably implement hover effects within the display area of the window.

```
virtual void onKeyDown(int keycode, unsigned long flags);
```
This event is fired as the window receives key down events.

```
virtual void onKeyUp(int keycode, unsigned long flags);
```
This event is fired as the window receives key up events.

```
virtual void onDraw(pxSurfaceNative s);
```
This event is fired whenever the window needs to be repainted. The pxOffscreen has a blit method that takes a pxNativeSurface as a parameter.
[Note: If you with to do some platform specific drawing please refer to the pxOffscreenNative class for the definition of the pxSurfaceNative type.]

```
virtual void onAnimationTimer();
```
This event will be fired n times per second in response to a prior call to the setAnimationFPS method. This event can be used to do simple animation or periodic events.

Class pxEventLoop
This class is used to abstract an eventloop it contains two methods that are used to control the lifetime of the eventloop.

Methods

void run(void);
This method runs the eventloop. Event messages will be processed and dispatched appropriately this method will not return until the pxEventLoop::exit method is called.

void exit(void);
This method will cause the currently running eventloop to exit. This will typically be called from within an event handler in response to some user action.

Performance Timer Support
double pxSeconds(void);

This function returns a number that describes a point in time. The value returned from a single call to this function should not be interpreted directly. It should only be used to measure between the point in time that this function was called to the time that another call to this same function is called. You can do this by obtaining the values of two calls to this function and subtracting the value obtained from the first call from the value returned by the value of the second call. The resulting unit of time on this number will be seconds.

double pxMilliseconds(void);
This function returns a number that describes a point in time. The value returned from a single call to this function should not be interpreted directly. It should only be used to measure between the point in time that this function was called to the time that another call to this same function is called. You can do this by obtaining the values of two calls to this function and subtracting the value obtained from the first call from the value returned by the value of the second call. The resulting unit of time on this number will be milliseconds.

double pxMicroseconds(void);
This function returns a number that describes a point in time. The value returned from a single call to this function should not be interpreted directly. It should only be used to measure between the point in time that this function was called to the time that another call to this same function is called. You can do this by obtaining the values of two calls to this function and subtracting the value obtained from the first call from the value returned by the value of the second call. The resulting unit of time on this number sill be microseconds.

void pxSleep(unsigned long msToSleep);
This function specifes the time in milliseonds for the current application to suspend execution. *[Note: The accuracy of this function is not guaranteed and is dependent on the underying OS implementation.]*

License

as being the original software.

3. This notice may not be removed or altered from any source distribution.

John Robinson
pxCore@liquidthought.com
http://www.liquidthought.com