SDL
Standard Drawing Library
C Library Reference Manual

Rastergraf

Rastergraf, Inc.

1810-J SE First St.
Redmond, OR   97756
(541) 923-5530

web: https://www.rastergraf.com
email: support@rastergraf.com

Release 3.6.2
November 15, 2006

Notices

Trademarks mentioned in this manual are the property of their respective owners.

This manual is based in part on *Xlib - C Language X Interface, Version 11, Release 6* which is copyrighted material. This documentation is used under the terms of its copyright which grants free use as noted below.

SDL is distributed by Rastergraf, Inc under license from:

Curtiss-Wright Controls Embedded Computing Video and Graphics (formerly Peritek)

Preface

This manual is a reference document for programmers using the Standard Drawing Library, SDL. This manual provides a brief overview of SDL features and describes each function in the library.

SDL is licensed in object and source format for use with embedded systems and in other industrial graphics hardware products. Although designed primarily for use with graphics boards from Rastergraf, the source code format allows SDL to be easily ported to the many different products in the marketplace.

The functions described in this manual represent the complete list of SDL library functions. Not all functions may be implemented in all graphics board drivers.

SDL requires a system with an ANSI C compiler and operating system and processor with a linear address space.

Permission is granted to licensed users to reproduce this manual for non-commercial uses. This means that this manual may not be resold to third parties. It does mean that licensed users of SDL may reproduce this manual for use in developing SDL programs, which may be included in a product and subsequently sold.

SDL is the product of over 20 years of experience developing graphics products for industry and we hope you will find it useful.

Please visit our web page at http://www.rastergraf.com for the latest information on our current graphics products.

Contents

Standard Drawing Library (SDL) Overview

System Overview

The *Standard Drawing Library, SDL,* is a scaleable C graphics library designed for use with real-time and non-real-time operating systems. *SDL* is small, compact, ROMable, and offers device independent graphics functions for board level and embedded systems applications.

SDL is easy to use and provides a complete set of graphics primitives. These graphics primitives can be extended by adding **utility functions** for specialized graphics tasks**.**

SDL is written in ANSI C and is supplied in library format, which means that its target code size can be controlled by limiting the number of functions used in a given application. *SDL* has been designed to run on any CPU and operating system that uses linear addressing and that is supported by an ANSI C compiler and linker.

SDL includes a *generic graphics driver* module that provides the hardware specific routines needed to interface to the graphics hardware in a user's system. A **graphics driver specification** is provided to customers licensing SDL in source format, allowing the customer to port *SDL* to the specific graphics hardware in the customer's system. *SDL* can be readily ported to a new system by changing this one module. *SDL*, therefore, does not depend on any specific graphics hardware, and versions of *SDL* can be used with VGA chips, EL panels, LCD displays, and most other graphics devices.

SDL Feature Summary

- All graphics primitives are drawn as single pixel lines. Rectangles, polygons, circles, ellipses, and chords can be filled with a solid color or stipple patterns
- Use with Motorola PowerPC Products: *MVME160x, MVME260x,and MVME360x*
- Use with Rastergraf Graphics Boards under *Linux, VxWorks, OS9, and other real-time OSs*
- Full Featured and Easy to Use
- Written in ANSI C and supplied in Library or Source format
- Scaleable, ROMable, and Minimal RAM usage
- Circles, Ellipses, and Arcs
- Filled Circles, Chords, Sectors, and Polygons
- Solid, wide solid, and dashed lines, polylines, and rectangles
- filled rectangles, polygons, ellipses, circles, sectors, and chords
- pixel blits to/from the display and host memory
- Solid and Pattern Fills – Pixel Processing
- Proportional and Fixed Width Fonts
- Clipping Rectangle and Logical Origin
- Screen to Screen and Host to Screen Image Copy
- Mouse and Keyboard Support
- Video Capture Extensions

Colors

SDL can support most of the popular color configurations, including 16 color, 256 color, and true color systems. The color configuration is determined by the *graphics driver*, which is configured to match the graphics hardware used in the system. *SDL* maintains two colors, a foreground color and background color, stored in variables. The colors are specified with the *setForeground()* and *setBackground()* functions, which load the specified unsigned 32 bit pattern (representing the color) into the corresponding variable.

Fill Style, Drawing Primitives, and Fills

All drawing primitives are affected by the current fill style. *SDL* offers three fill styles: (1) *solid,* (2) *stipple*, (3) *opaque stipple*. For example, if *stipple* is the current fill style, a line would be drawn in the pattern defined by the current *stipple* pattern. See *setFillStyle()* for more information.

Rectangles, circles, chords, sectors, ellipses, and **polygons** can be filled with a **solid color**, **stipple pattern**, or **opaque stipple pattern**.

- **Solid color** fills use the current foreground color to fill the object.
- A binary pattern is used to specify a **stipple** pattern. Each bit in the binary pattern corresponds to a pixel. When an object is filled with a **stipple** pattern, the ones in the pattern are drawn in the current foreground color. The zeros in the pattern are not used and the corresponding screen (or memory) locations are not modified.
- An **opaque stipple** fill uses the same binary pattern used by the stipple fill, except that both the ones and zeros of the pattern are drawn in the corresponding foreground and background colors.
- An offset is provided to allow a stipple pattern to start at any point in the pattern as opposed to always starting at the beginning of the pattern. See *setPatternOrigin().*

Polygon Fill Rule

Polygons are often overlapped or grouped to create more complex graphics objects. How the polygon is filled affects the resulting complex object, especially if pixel processing is used. To provide the user some flexibility in this regard, polygons may be filled according to either the WINDING or EVENODD rule. See *setFillRule().*

Coordinate System

The coordinate system for *SDL* has the
origin in the upper left corner of the
screen. x increases to the right and y
increases downwards on the screen. A
clipping rectangle can be definedon the
display screen to limit where text and
graphics are drawn, and a *logical origin*,
which defaults to the screen origin can
be moved about the screen to make it
easy to position complex objects.

Screen Origin

0,0

+ x

Display Screen

+ y
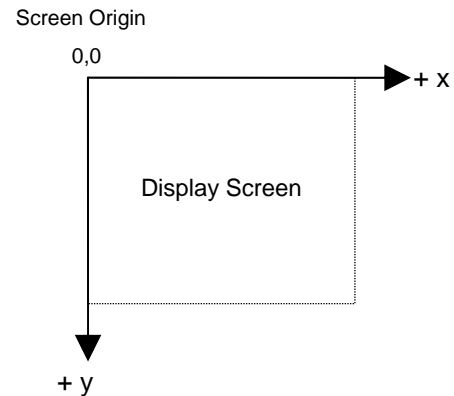
***SDL* Coordinate System**

Logical Origin

The logical origin is a point on the screen that is used as the origin for drawing all graphics
primitives. The default is for the logical origin to be at the screen origin. The logical origin is a
useful feature as it allows a symbol such as a gauge that is drawn relative to zero to be redrawn at
a different screen location by simply changing the logical origin to the desired coordinates before
redrawing the symbol.

Clipping Rectangle

A clipping rectangle is used to limit all graphics to a rectangular area of the display screen
(memory). The default is for the clipping rectangle to be the same size as the entire screen,
thereby limiting memory writes to the range of memory allocated for the display. The clipping
rectangle is programmable in size and location. See *setClipRect()* for more information.

Scanned Images or Bitmaps

Scanned images or bitmaps can be displayed with *SDL.* Two colored images such as logos are
easily drawn with *SDL* as they can be written with the current foreground and background colors.
Colored images such as photographs have many colors and may require processing to match the
scanned colors with those available in the user's system. If the user's system is palette based, the
palette can be updated with the new colors provided with the scanned image, or the scanned
image colors can be modified to use the colors available in the user's system.

Real-Time Applications and Code Size

For real-time applications, *SDL* graphics must be confined to a single task. The *SDL* task is
preemptible by other real-time tasks. Restricting *SDL* to a single task is necessary because *SDL*
does not directly support multi-tasking. Multi-tasking can be achieved by providing a task that
interfaces to *SDL* and that accepts inputs from other tasks. Interprocess communication must be
supported by the real-time operating system to allow the implementation of a multi-tasking *SDL*
application.

Pixel Processing

SDL supports **replace, or, and,** and **xor** pixel processing. For pixel processing to work, the
graphics driver must be able to generate read/modify/write memory cycles to and from the
graphics hardware memory.

Fonts

SDL provides 18 fonts, including both proportional and fixed width fonts. See Appendix B for a description of the fonts included with *SDL*. Fonts can increase the code size significantly, as some of the larger fonts can be 10k bytes or larger in size. For applications with limited memory or ROM, fonts types should be used sparingly to limit *SDL* code size. Additional fonts are available in separate font utility.

The user must specify the fonts to be included in the run time code at compile time. The selected fonts must be included in the file, *userinit.c*. See *initGraphics()* for more information.

Text may be rotated as it is drawn on the display (see *setTextDirection())*.

SDL is Scaleable to Minimize Memory Requirements

SDL is scaleable, which means the user can select only the required graphics functions or group of functions from the library to minimize memory usage.

SDL has been carefully designed so it can run from ROM with a minimum of RAM. For example, for a 68040 processor, *SDL* currently requires about 30k bytes of code if all functions are used, excluding fonts and user code. A practical application for this processor could be implemented with 50k bytes of code.

Some functions, such as polygon fills, do require RAM to construct edge tables, etc. See Appendix C for the code and RAM requirements of *SDL* functions for popular processors.

SDL will run on 8 bit, 16 bit, 32 bit, and 64 bit processors with linear address spaces. The size of the target code produced with *SDL* will vary with the word size and architecture of the processor used.

Graphics Driver

SDL is portable and can be used on a wide range of systems. To be portable, *SDL* uses a simple *graphics driver* to interface *SDL* to the user's graphics hardware, such as VGA display, Flat Panel, etc. The *Graphics Driver* is a module in *SDL* and is the only part of *SDL* that is modified when porting *SDL* to a new platform.

Graphics Driver *YES,* OS Driver *NO*

SDL requires a *graphics driver* to support the user's graphics hardware, but does not require a driver for the real-time operating system, as the *SDL* functions are merely linked with user code. Thus, *SDL* can be used with any operating system using linear addressing and which is supported with an ANSI C compiler and linker.

Performance

SDL runs on the host processor and its performance will vary depending on the processor used. A 68040 class processor gives good graphics performance for industrial control applications. Faster processors such as the 603 PowerPC give outstanding performance.

Multiple Pages of Video Memory, Multiple Output Devices

For output devices that have multiple pages of video memory (such the Rastergraf SVGA-based boards), *SDL* offers functions to switch between pages, copy regions from one page to another, and to have different display and drawing pages. More than one output device (or graphics board) can be configured into the system by the user. The active device is selectable by a function call. For example, on graphics hardware that supports it, this lets you switch between CRT and LCD output. The graphics driver for the Rastergraf graphics boards also supports the overlay plane of its VMEbus graphics boards.

Multiple Video Modes

Most *SDL* drivers include support for multiple video modes, including different resolutions and number of colors (see *initGraphics()*). The minimum resolution/colors for most drivers is 640x480 with 16 colors. The maximum resolution/colors depends on the graphics hardware used. Some graphics cards support resolutions as high as 1600x1200 with 65 million (24-bit) colors.

Mouse and Keyboard Support

Many *SDL* drivers include low level support for an attached mouse and/or keyboard. Library functions are available to check for pending mouse or keyboard events and to read the mouse and keyboard. These functions may not be available with all drivers.

Video Capture Extensions

Available for hardware that supports video capture, the SDL Video Capture Extensions are a set of functions that allow you to setup your hardware to capture (or grab) video fields and frames and to display the video image with overlay or underlay graphics. Functions are available for specifying the area to capture, to change the video input source, to scale or resize the video image and to display a window into the image on the screen. See **Appendix E: Video Capture Extensions** for more detail. The video capture extensions fully support the capture capabilities of the display/capture boards available from Rastergraf.

Long Product Life

SDL offers hardware independent graphics functions, uses a simple *graphics driver*, and provides for increased functionality by allowing the use of additional graphics utilities.

The obvious advantages of this product strategy include:
- Protecting the user's investment in application code.
- Allowing additional functionality to be added when needed.
- Easy port of *SDL* to newer and faster processors.
- Sidestepping the problems associated with obsolete graphics hardware by using a simple *graphics driver* that can be easily updated to work with new graphics devices.

How *SDL* C Library Functions are Used

Graphics applications using *SDL* source code to generate run time code, do so by compiling and linking *SDL* source code and associated application code the same way the rest of the applications code is developed.

For applications using a run time versions of *SDL*, the graphics functions are used the same way as are the resident C functions supported by the user's C compiler and linker. The application code is compiled and *SDL* functions are linked with user code and loaded into memory for execution. For ROMable code, the user's compiler/linker must be capable of generating code that can be placed in ROM.

For systems that support library archives, the compiled object files can be archived together into a single file, which can then be linked with the user's application.

Graphics Programming Example

This example first draws a white bounding rectangle to simulate the edge of the video screen. Next, it shows how to redefine the size and location of the clipping rectangle. It then draws a filled rectangle within the clipping area and draws the text string **Big Box** under the red filled rectangle. The filled rectangle is positioned so that part of it extends outside the clipping rectangle and is therefore clipped as shown below in Figure 1.
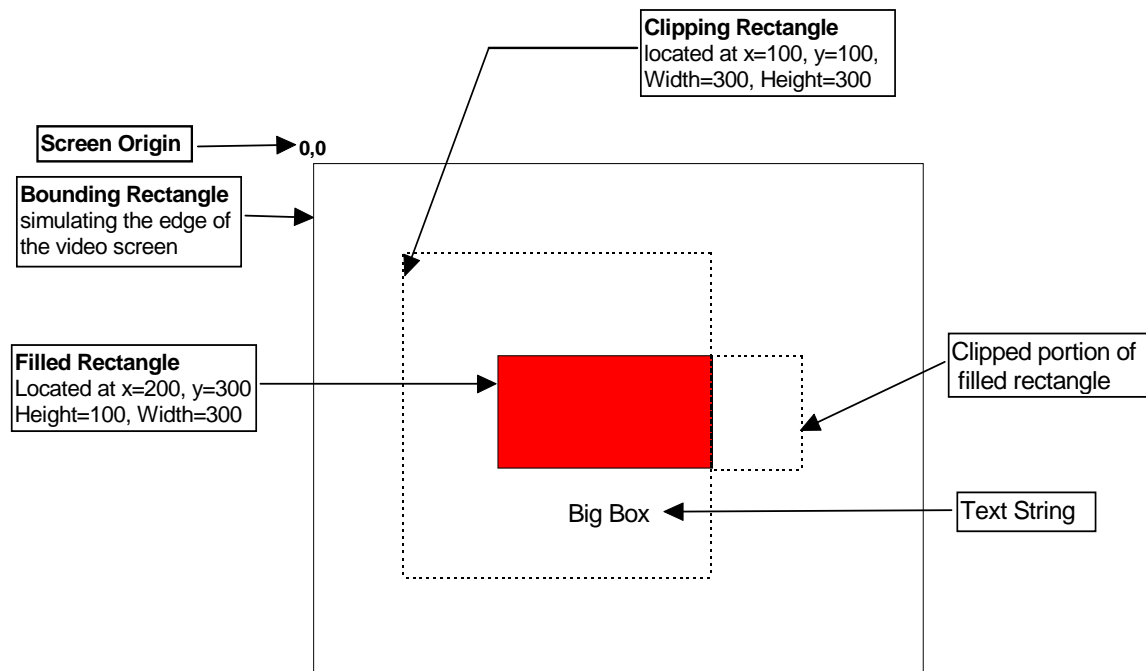
Figure 1: A Clipped Rectangle

This example shows:

1. **How to draw the bounding rectangle.**
2. **How to draw a dashed rectangle simulating the clipping rectangle.**
3. **How to redefine the size and location of the clipping rectangle.**
4. **How to draw a filled rectangle.**
5. **How to draw a text string.**

(1) How to draw the bounding rectangle.

The three include files, *sdl.h*, *extern.h*, and *colors.h* are required by most SDL programs, as they contain constants, structure definitions, defines, and function prototypes. These files are shown in Appendix A. The first thing this program does is initialize SDL and the graphics hardware by calling *initGraphics().* It then sets the foreground color to white with *setForeground().* Then the bounding rectangle is drawn with the *rectangle()* function, outlining the edge of the 640h x 480v video screen.

(2) How to draw a dashed rectangle simulating the clipping rectangle.

Next, the foreground color is changed to green with *setForeground()* and the dashed line style is set to on/off with *setDashStyle().* The on/off dashed line will be drawn with green dashes against the dark screen background. Double dashed lines could also be specified, which would produce a two color dashed line. The dashed rectangle will be drawn using the default dashed line pattern but the dash lengths could be redefined with

setDashPattern() if a different dashed line pattern was desired. The upper left corner of the green dashed rectangle is located at x=100, y=100 and its width = height = 300. The green dashed rectangle is drawn with *dashedRectangle().*

Next, a dashed red rectangle is drawn that extends to the right of the clipping area to show the part of the filled rectangle (not drawn yet) that is clipped (not drawn) because it falls outside the *clipping rectangle*. The foreground color is changed to red, and the red dashed rectangle is drawn with *dashedRectangle().*

(3) Set the clipping rectangle.

The clipping rectangle defaults to the entire screen size, but can be resized and positioned anywhere within the screen area. In this example, the *clipping rectangle* is represented by the green dashed rectangle described above. The actual *clipping rectangle* is defined with the *setClipRec()* function and takes the same parameters as the *dashedRectangle()* function used above to outline the clipping rectangle. After the clipping rectangle has been resized and positioned to coincide with the green dashed rectangle, graphics can only be drawn within the *clipping rectangle*. Note that the smaller red dashed rectangle could not be drawn at its current location once the clipping rectangle has been defined. From this point on, the only drawing routine which can alter pixels outside the clipping rectangle, without redefining the *clipping rectangle*, is the **clearScreen()** routine.

(4) How to draw a filled rectangle.

The filled red rectangle is drawn using *filledRectangle()* and works much the same as the *dashedRectangle()* function. The fill style is set to solid fill, which fills the rectangle with the current foreground color, red. The filled rectangle is located at x=200, y=200 and is 300 pixels wide and 100 pixels high. This filled red rectangle overwrites the portion of the dashed red rectangle within the clipping rectangle, leaving the right 100 pixels of the dashed red rectangle unmodified. This unmodified portion of the dashed red rectangle shows the part of the filled red rectangle that was clipped and did not get written.

(5) How to draw a text string.

The text string 'Big Box' is drawn below the filled rectangle. The text string is located at x=300, y=325 and is drawn using the *drawText()* function. The color of the text is determined by the current foreground color, which can be set with *setForeground().*

The text is drawn at a fixed location in this example, but other options are available. For example, the width of the text string can be calculated by the *getTextWidth()* function, allowing the text to be centered relative to a screen location. Sixteen font types are available with SDL. This example uses the current font. See Appendix B for a description of the SDL fonts.

Listing of clip.c

```
/*****************************************************************************/
/*                        STANDARD DRAWING LIBRARY                        */
/*                                                                         */
/*                            Rastergraf, Inc.                            */
/*         Used under license from CURTISS-WRIGHT CONTROLS, INC.           */
/*           COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.              */
/*                                                                         */
/*   This software is licensed software subject to the terms of the        */
/*   Source Code License Agreement. Refer to the file LICENSE for details. */
/*****************************************************************************/
/* FILE NAME  : clip.c                                                      */
/* DESCRIPTION : Example program for setting clip rectangle                 */
/* AUTHOR       : P. K.                                                     */
/* DATE CREATED : 8/3/95                                                    */
/*****************************************************************************/
/* This program is designed to show how clipping works in SDL. ...........*/
/*****************************************************************************/
#include <sdl.h>      /* SDL root header file                              */
#include <extern.h>   /* SDL header file containing global vars            */
#include <colors.h>   /* SDL color names equated to index values           */



#define GREEN_RECTANGLE_X 100
#define GREEN_RECTANGLE_Y 100
#define GREEN_RECTANGLE_WIDTH 300
#define GREEN_RECTANGLE_HEIGHT 300

#define RED_FREC_X 200
#define RED_FREC_Y 200
#define RED_FREC_WIDTH 300
#define RED_FREC_HEIGHT 100

#define TEXT_X 300
#define TEXT_Y 325


void main(int argc,char **argv)
{
/* initialize the graphics hardware and setup SDL                          */
   initGraphics(argc, argv);

/* at this point in the program the clipping rectangle is the              */
/* same size as the entire video screen.                                   */

/* set the current foreground color                                        */
   setForeground(White);

/* draw a bounding rectangle at the edges of the screen                    */
   rectangle(0,0,640,480);

/* set the current foreground color                                        */
   setForeground(Green);

/* set the dashstyle to On/Off dashes                                      */
   setDashStyle(ONOFF_DASH);

/* draw a dashed rectangle at the boundaries of where the clipping         */
/* rectangle will be located.                                              */
   dashedRectangle(GREEN_RECTANGLE_X, GREEN_RECTANGLE_Y,
                   GREEN_RECTANGLE_WIDTH, GREEN_RECTANGLE_HEIGHT);
```

```
/* set the current foreground color to Red                      */
   setForeground(Red);


/* draw a dashed rectangle the size of the Red Filled Rectangle  */
   dashedRectangle(RED_FREC_X, RED_FREC_Y, RED_FREC_WIDTH,
                   RED_FREC_HEIGHT);


/* set the clipping rectangle to coincide with the green         */
/* dashed rectangle                                              */
   setClipRect(GREEN_RECTANGLE_X, GREEN_RECTANGLE_Y,
               GREEN_RECTANGLE_WIDTH, GREEN_RECTANGLE_HEIGHT);


/* now that clipping rectangle has been redefined, draw the filled */
/* red rectangle and observe that the 100 pixels on the right side, */
/* represented by the dashed red rectangle, does not get drawn.    */
   filledRectangle(RED_FREC_X, RED_FREC_Y, RED_FREC_WIDTH,
                   RED_FREC_HEIGHT);

/* use default font                                              */
   drawText(TEXT_X,TEXT_Y,"Big Box");

/* end graphics task                                             */
   closeGraphics();

} /* end of main */
```

Standard Drawing Library

C FUNCTIONS

SDL C Function Summary

Function Name	Description
void **arc(**int x, int y, int width, int height, int startAngle, int endAngle**)**	Draw an elliptical arc with start and stop
void **arc2(**int x, int y, int width, int height, int startAngle, int angleExtent**)**	Draw an elliptical arc with start and extent
int **boardOK(**void**)**	Provide board status
void **circle(**int x, int y, int radius**)**	Draw a circle
void **clearScreen(**void**)**	Clears the display screen
void **closeGraphics(**void**)**	Ends graphics task
void **copyImage**(int x, int y, int width, int height, int dx, int dy)	Copy image from one region of display to another
void **copyPageImage**(int spage, int x, int y, int width, int height, int dpage, int dx, int dy)	Copy image from region of one display page to another page
void **copyPage**(int spage, int dpage)	Copy contents of one page to another
void **dashedLine(**int x0, int y0, int x1, int y1**)**	Draw a dashed line
void **dashedPolyline(**int numPts, sPoint *ptr**)**	Draw a dashed polyline
void **dashedRectangle(**int x, int y, int width, int height**)**	Draw a dashed rectangle
void **drawPixel(**int x, int y**)**	Write pixel at x,y
void **drawText(**int x, int y, char *string**)**	Draw 8-bit text
void **enableStereo**(int onoff)	Enable/disable stereo output
void **ellipse(**int x, int y, int width, int height**)**	Draw an ellipse
void **filledArc(**int x, int y, int width, int height, int startAngle, int endAngle**)**	Draw a filled sector or chord with start/end
void **filledArc2(**int x, int y, int width, int height, int startAngle, int angleExtent**)**	Filled sector or chord with start/extent
void **filledCircle (**int x, int y, int radius**)**	Draw a filled circle
void **filledEllipse(**int x, int y, int width, int height**)**	Draw a filled ellipse
void **filledPolygon(**int numPts, sPoint *ptr**)**	Draw a filled polygon
void **filledRectangle(**int x, int y, int width, int height**)**	Draw a filled rectangle
void **flushKeyboard(**void**)**	Flush the keyboard queue
void **flushMouse(**void**)**	Flush the mouse queue
void **getColor(**int index, int *red, int *green, int *blue**)**	Read color from palette
int **getDisplayPageStatus**(void)	Get status of h/w address register
void **getFontStruct(**sFontStruct *fs**)**	Get font parameters
unsigned char ***getFrameBufPtr(**void**)**	Get pointer to graphics framebuffer
void **getImage**(int x, int y, int width, int height, unsigned char *buff)	Copy image from display to host memory
void **getMouseXY(**int *mx, int *my**)**	Get the current mouse location
unsigned long **getPixel(**int x, int y**)**	Get the pixel at x,y

SDL C Function Summary (con't)

Function Name	Description
int **getTextWidth(**char *string**)**	Get width of ASCII text string
int **initGraphics(**int argc, char **argv**)**	Initialize graphics hw and global data
void **keyboardRead(**unsigned short *kcode**)**	Read a keycode from the keyboard queue
int **keyboardReady(**void**)**	Check for enqueued keycodes
void **line(**int x0, int y0, int x1, int y1**)**	Draw a line
void **mouseCursorOn(**int state**)**	Turn mouse cursor ON or OFF
void **mouseCursorXY(**int mx, int my**)**	Move mouse cursor to a new position
void **mouseRead(**sMouseEvent *mse**)**	Read a mouse event from mouse queue
int **mouseReady(**void**)**	Check for events in mouse queue
void **mouseRect(**int x, int y, int w, int h**)**	Set window limits for mouse cursor
void **mouseScale(**int xscale, int yscale**)**	Set scale factors for mouse cursor
int **panelType**(void)	Report display panel type
void **polyline(**int numPts, sPoint *ptr**)**	Draw a polyline
void **putImage**(unsigned char *buff, int width, int height, int x, int y)	
	Copy image from host memory to display
void **rectangle(**int x, int y, int width, int height**)**	Draw a rectangle
void **setArcMode(**int arcMode**)**	SECTOR_MODE or CHORD_MODE
void **setBackground(**unsigned long color**)**	Set the background color
void **setClipRect(**int x, int y, int width, int height**)**	Set clipping rectangle
void **setDashOffset(**int dashOffset)	(Re-)set offset into dash line pattern
void **setDashPattern(**int numDashes, unsigned char *dashList, int dashOffset**)**	
	Set dashed line pattern
void **setDashStyle(**int dashStyle)	ONOFF_DASH or DOUBLE_DASH
void **setDisplayPage(**int page**)**	Set video memory page for display
void **setFillRule(**int fillRule**)**	EVENODD or WINDING
void **setFillStyle(**int fillStyle**)**	SOLID_FILL, STIPPLE_FILL, or OPAQUE_FILL
void **setFont(**int fontIndex**)**	Set the current font
void **setForeground(**unsigned long color**)**	Set current foreground color
void **setGraphicsDevice(**int devnum**)**	Set current graphics display device
void **setLineWidth**(int width)	Set line width
int **setMode**(char *args)	Change video mode
void **setMouseCursor(**unsigned long csr_id, unsigned long color1, unsigned long color2**)**	
	Set mouse cursor type and colors
void **setMousePage(**int page**)**	Set video memory page for mouse display
void **setMouseParam(**int reg, int value**)**	Set mouse configuration registers
void **setOrigin(**int x, int y**)**	Set the logical origin
void **setPanStart(**int x, int y**)**	Set display origin within the virtual window
void **setPattern(**sPattern *newPattern**)**	Set stipple fill pattern
void **setPatternOrigin(**int x, int y**)**	Set stipple fill pattern origin
void **setPixelProcessing(**int operation**)**	REPLACE, AND, OR, or XOR

SDL C Function Summary (con't)

Function Name	Description
void **setTextDirection**(int dir)	Set text drawing direction to TXT_DIR_NORM, TXT_DIR_UP, TXT_DIR_RL , or TXT_DIR_DWN
void **setTiming**(float vfreq, float vblank, float vfporch, float vsync, float hblank, float hfporch, float hsync)	Set custom video timing parameters
void **setTransparency(**int transparency**)**	TRANSPARENT or OPAQUE
void **setVirtualSize(**int w, int h**)**	Set size of the virtual graphics window
void **setWritePage(**int page**)**	Set video memory page for drawing
void **storeColor(**int index, int red, int green, int blue**)**	Update color palette
void **syncControl**(int hsync, int vsync)	Override the default sync state to: SYNC_NORMAL, SYNC_LOW, SYNC_HIGH, or SYNC_OFF

SDL C Function Reference

arc

NAME

arc - draws an elliptical arc

SYNOPSIS

```
void arc
        (
        int x,                /*x-coord of bounding rectangle origin */
        int y,                /*y-coord of bounding rectangle origin */
        int width,            /*width of bounding rectangle */
        int height,           /*height of bounding rectangle */
        int start_angle,      /*start angle (in 32nds of a degree) */
        int end_angle         /*end angle (in 32nds of a degree) */
        )
```

DESCRIPTION

This routine draws an elliptical arc (or circular if width=height). Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height of the rectangle. Positive angles indicate counterclockwise direction, and negative angles indicate clockwise motion. If the magnitude of either angle is greater than 360 degrees, it is set to the angle modulo 360 degrees. The arc is drawn relative to the logical origin.

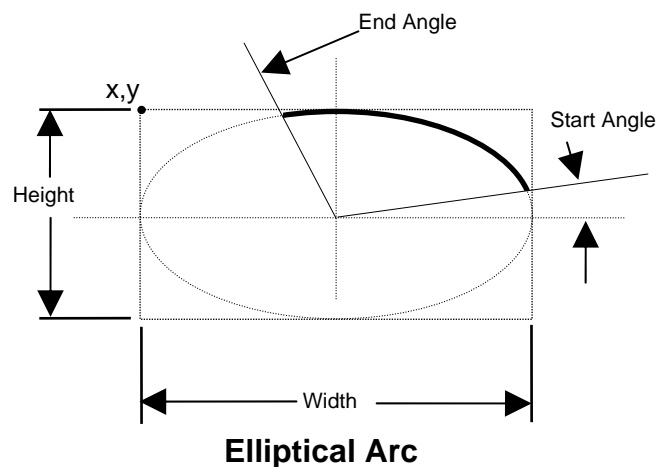*All drawing primitives are affected by the current fill style which is specified with* **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

arc2(), filledArc()

Elliptical Arc

arc2

NAME

arc2 - draws an elliptical arc

SYNOPSIS

```
void arc2
      (
      int x,                /*x-coord of bounding rectangle origin */
      int y,                /*y-coord of bounding rectangle origin */
      int width,            /*width of bounding rectangle */
      int height,           /*height of bounding rectangle */
      int start_angle,      /*start angle (in 32nds of a degree) */
      int angle_extent      /*angle extent (in 32nds of a degree) */
      )
```

DESCRIPTION

This routine draws an elliptical arc (or circular if width=height). Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height of the rectangle. Positive angles indicate counterclockwise direction, and negative angles indicate clockwise motion. If the magnitude of *start_angle* is greater than 360 degrees, it is set to the angle modulo 360 degrees. If the magnitude of *angle_extent* is greater than 360 degrees, it is truncated to 360 degrees. The arc is drawn relative to the logical origin.

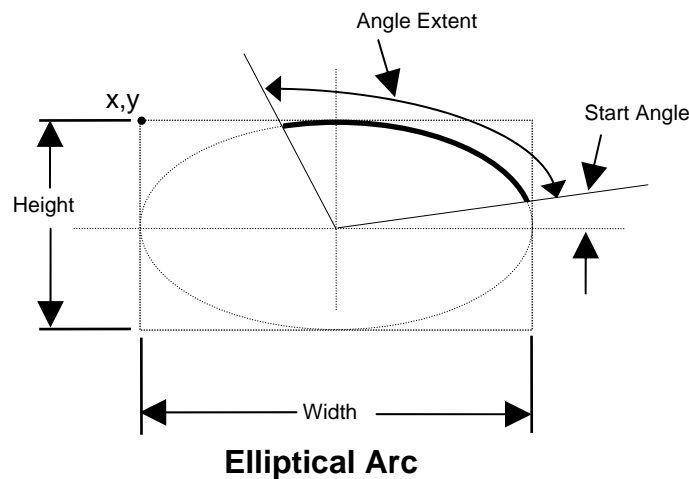*All drawing primitives are affected by the current fill style which is specified with* **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

arc(), filledArc2()

Elliptical Arc

boardOK

NAME

boardOK – perform a simple board test

SYNOPSIS

`int boardOK(void)`

DESCRIPTION

This function performs a few simple board tests to verify the graphics board can be accessed and the framebuffer memory can be read and written to.

RETURNS

int /* 1 on success, 0 on failure */

INCLUDE FILES

sdl.h

SEE ALSO

boardTemp

NAME

boardTemp – report the board temperature

SYNOPSIS

```
int boardTemp(void)
```

DESCRIPTION

This function reports the temperature measured by the onboard LM75 temperature sensor. This function is not available on all graphics boards.

RETURNS

int /* temperature in degrees C */

INCLUDE FILES

sdl.h

SEE ALSO

circle

NAME

circle - draws a circle

SYNOPSIS

```
void circle(int x, int y, int radius)
```

DESCRIPTION

This function draws a single line circle centered about x,y and of the radius specified. x,y is relative to the logical origin.

All drawing primitives are affected by the current fill style which is specified with *setFillStyle().* Use SOLID_FILL for drawing solid lines.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

filledCircle()
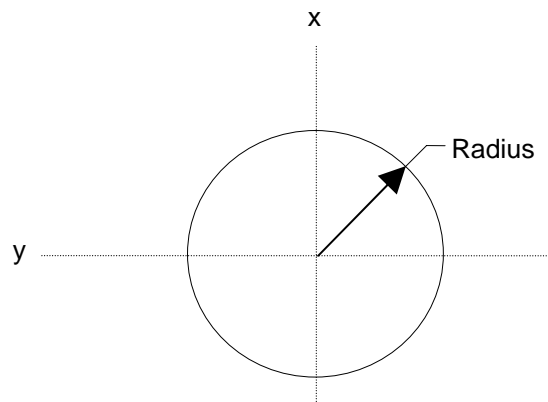
Circle

clearScreen

NAME

clearScreen - clears the screen

SYNOPSIS

```
void clearScreen(void)
```

DESCRIPTION

This function clears the entire screen to black. The color for black must be specified in the *userinit.c* file which is executed by *initGraphics().*

INCLUDE FILES

sdl.h, extern.h

closeGraphics

NAME

closeGraphics - closes the graphics task

SYNOPSIS

```
void closeGraphics(void)
```

DESCRIPTION

This routine is used to terminate graphics processing, and should be the last *SDL* function used before any calls are made to the operating system to terminate the graphics task. This routine is specific to the output graphics hardware and to the operating system being used.

Failure to call *closeGraphics()* when exiting a graphics application may lead to upredictable system operation.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

initGraphics()

copyImage

NAME

copyImage - copy area of display from one location to another

SYNOPSIS

```
void copyImage
    (
    int x,                    /* source upper left x-coord */
    int y,                    /* source upper left y-coord */
    int width,                /* source width */
    int height,               /* source height */
    int destx,                /* destination upper left x-coord */
    int desty                 /* destination upper left y-coord */
    )
```

DESCRIPTION

*CopyImage()* copies a rectangular area of the display to another place on the display.

The image data is copied from and to the current write page (which may be different than the current display page). The image is also clipped to the current clipping rectangle for the source and destination rectangles.

INCLUDE FILES

sdl.h

SEE ALSO

copyPage(), copyPageImage(), getImage(), putImage()
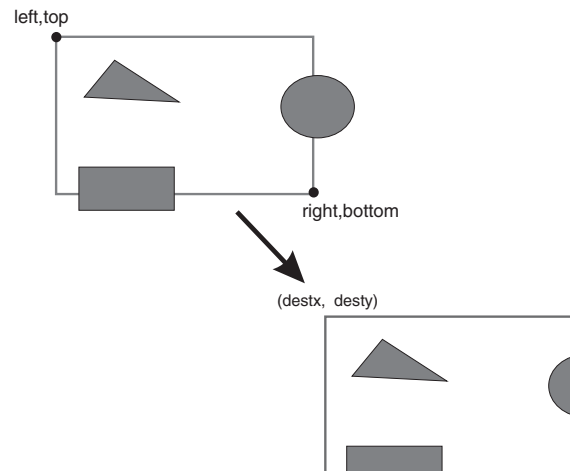
copyPage

NAME

copyPage - copy one page of graphics memory to another page

SYNOPSIS

```
void copyPage
     (
     int src_page,             /* source page number */
     int dst_page              /* destination page number */
     )
```

DESCRIPTION

CopyPage() copies the contents of video RAM page *src_page* to video RAM page *dst_page*. This function is only available for drivers that support mutliple video pages and graphics hardware that physically has multiple pages of video memory.

The entire page is always copied, regardles of the current clipping rectangle.

INCLUDE FILES

sdl.h

SEE ALSO

copyImage(), copyPageImage(), getImage(), putImage()

copyPageImage

NAME

copyPageImage - copy area of display from one page to another page

SYNOPSIS

```
void copyPageImage
    (
    int src,                /* source video page */
    int x,                  /* source upper left x-coord */
    int y,                  /* source upper left y-coord */
    int width,              /* source width */
    int height,             /* source height */
    int dst,                /* destination video page */
    int destx,              /* destination upper left x-coord */
    int desty               /* destination upper left y-coord */
    )
```

DESCRIPTION

CopyPageImage() copies a rectangular area of the source video page to location (*destx*, *desty*) in a destination video page. *Src* and *dst* can refer to the same or different pages in video memory.For Rastergraf VME graphics boards, both pages must be in the same graphics channel.This function is only available for drivers that support mutliple video pages and graphics hardware that physically has multiple pages of video memory.

The image is also clipped to the current clipping rectangle for the source and destination rectangles.

INCLUDE FILES

sdl.h

SEE ALSO

copyImage(), copyPage(), getImage(), putImage()

dashedLine

NAME

dashedLine - draws a dashed line

SYNOPSIS

```
void dashedLine
    (
    int x0,                      /* x-coord of first endpoint */
    int y0,                      /* y-coord of first endpoint */
    int x1,                      /* x-coord of second endpoint */
    int y1                       /* y-coord of second endpoint */
    )
```

DESCRIPTION

This routine draws a dashed line from (*x0,y0*) to (*x1,y1*). The coordinates are relative to the logical origin.   It uses the current dashed line pattern. The dashed line pattern is set with the *setDashPattern()* function, which initializes an array of unsigned chars with the pixel lengths of the two sections of the dashed line. Two styles of dashed lines are possible, *on/off* and *double dash*, selected with the *setDashStyle()* function.

On/off dashed lines write the even array index values ([0], [2], [4], etc. that define the dashed line pattern) in the current foreground color.   The odd array index values ([1], [3], [5], etc.) represent the number of pixels to skip. The result is that *on/off* dashed lines are drawn in the foreground color and the gap part of the dashed line is not drawn (it is skipped).

Double dashed lines draw both parts of the dashed line using the current foreground and background colors. The odd array values [1], [3], [5] are drawn in the background color. The result is a two color dashed line.

*All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

dashedPolyline(),setDashPattern(), setDashStyle(), setLineWIdth()
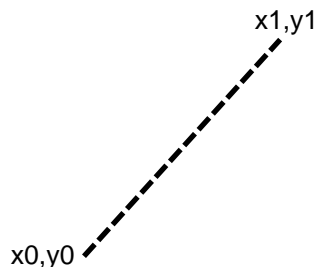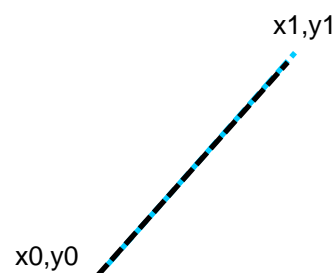
On/Off Dashed Line   **Double Dashed Line**

dashedPolyline

NAME

dashedPolyline - draws a dashed polyline

SYNOPSIS

```
void dashedPolyline
     (
     int num_pts,                /*number of points in the array */
     sPoint *ptr_to_coord_list /*pointer to array of points */
     )
```

DESCRIPTION

This routine draws dashed lines between each pair of points in the array of *sPoint* structures. It draws the lines in the order listed in the array. All coordinates are relative to the logical origin. It uses the current dashed line pattern. The dashed line pattern is set with the *setDashPattern()* function. Two styles of dashed lines are possible: *on/off* and *double dash*. *On/off* dashed lines write the even array index values ([0], [2], [4], etc. that define the dashed line pattern) in the current foreground color. The odd array index values ([1], [3], [5], etc.) represent the number of pixels to skip. The result is that *on/off* dashed lines are drawn in the foreground color and the gap part of the dashed line is not drawn (it is skipped). *Double dashed* lines draw both parts of the dashed line using the current foreground and background colors. The odd array values [1], [3], [5] are drawn in the background color. The result is a two color dashed line.

All drawing primitives are affected by the current fill style which is specified with setFillStyle(). Use SOLID_FILL for drawing solid lines.

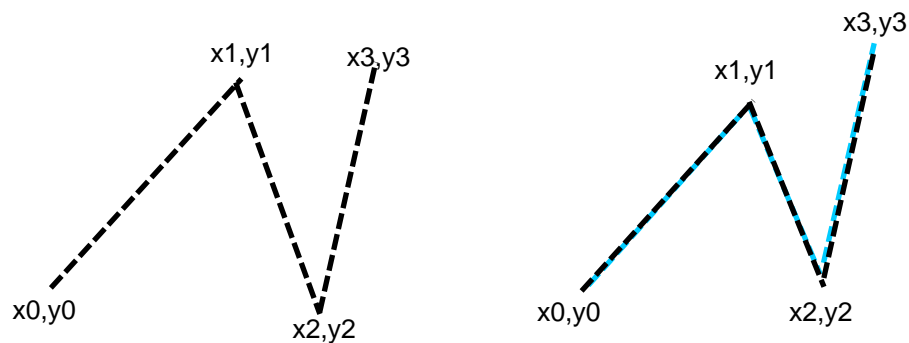The **sPoint** structure is defined as follows:

```
typedef struct tagPoint
{
short x;
short y;
} sPoint;
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

dashedLine(), setDashStyle(), setDashOffset(), setDashPattern(), setLineWidth()

On/Off Dashed Polyline   **Double Dashed Polyline**

dashedRectangle

NAME

dashedRectangle - draws a dashed rectangle

SYNOPSIS

```
void dashedRectangle
    (
    int x,                 /* x-coord of upper left corner */
    int y,                 /* y-coord of upper left corner */
    int width,             /* width  */
    int height             /* height */
    )
```

DESCRIPTION

This routine draws a dashed rectangle. The *x* and *y* coordinates are relative to the logical origin. Depending on the current dash style, rectangles can be drawn with *on/off* or *double dashed* lines. The dashed line pattern is set with the *setDashPattern()* function, which initializes an array of unsigned chars with the pixel lengths of the two sections of the dashed line. Two styles of dashed lines are possible, *on/off* and *double dashed*, selected with the *setDashStyle()* function.

On/off dashed lines write the even array index values ([0], [2], [4], etc. that defines the dashed line pattern) in the current foreground color. The odd array index values ([1], [3], [5], etc.) represent the number of pixels to skip. The result is that *on/off* dashed lines are drawn in the foreground color and the gap part of the dashed line is not drawn (it is skipped). *Double dashed* lines draw both parts of the dashed line using the current foreground and background colors. The odd array values [1], [3], [5] are drawn in the background color. The result is a two color dashed line.

Double dashed lines draw both parts of the dashed line using the current foreground and background colors. The result is a two color dashed line.

All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

rectangle(), filledRectangle(), setDashOffset(), setDashPattern(), setDashStyle()

On/Off Dashed Rectangle   **Double Dashed Rectangle**

drawPixel

NAME

drawPixel - draws a pixel at x,y

SYNOPSIS

```
void drawPixel
    (
    int x,                      /* x-coord of pixel */
    int y                       /* y-coord of pixel */
    )
```

DESCRIPTION

This routine draws a pixel at the display coordinates x,y, relative to the logical origin.

This routine draws a single pixel using the current fill style (set by *setFillStyle()*) and uses the foreground color when the fill style is solid color or stipple, and both the foreground and background colors when the fill style is opaque stipple.
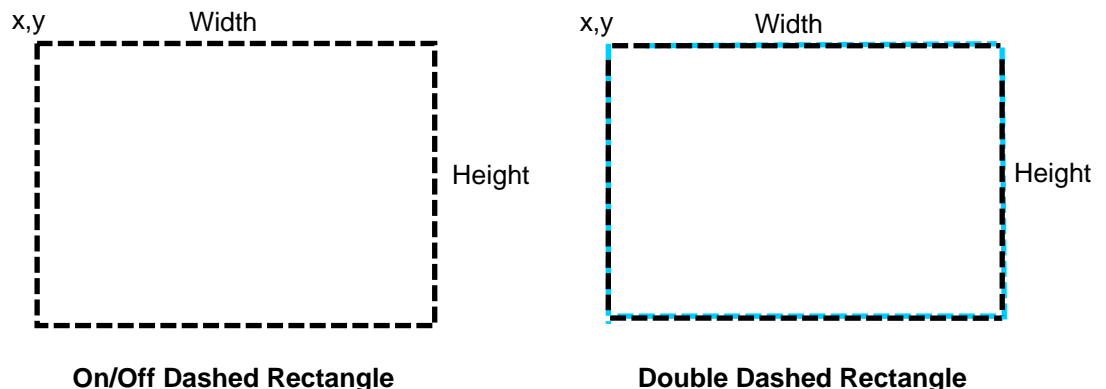
All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO
getPixel()

drawText

NAME

drawText - draws a text string

SYNOPSIS

```
void drawText
    (
    int x,                          /* x-coord of text origin */
    int y,                          /* y-coord of text origin */
    char *string                    /* string to draw */
    )
```

DESCRIPTION

This routine draws a string of ASCII coded characters starting at the specified location, x,y.   x,y is located relative to the logical origin and defines the location of the first character. The character is positioned so that the left side of its bounding box is located at x, and the character is positioned vertically so that its baseline sits on a horizontal line located at y. *string* contains the ASCII string of text to be printed to the screen or graphics output device.

Font glyphs are defined with a binary pattern. Ones in the character glyph are drawn in the current foreground color. If transparency is on, only the ones are drawn and the zeros in the glyph pattern are not drawn. If transparency is off, the ones are drawn in the foreground color and the zeros in the glyph pattern are drawn in the background color.

The default is transparency on.

In adition to normal left to right text drawing, a text string may be drawn right to left, top to bottom, or bottom to top. If text rotated by an arbitrary angle is required, use the optional vector font or polygon font library.

All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setFont(), getTextWidth(), getFontStruct(), setTextDirection()

Baseline ⟶ Text String   Ascent
                            Descent

x,y - First character is located at x,y

ellipse

NAME

ellipse - draws an ellipse

SYNOPSIS

```
void ellipse
     (
     int x,              /*x-coord of bounding rectangle origin */
     int y,              /*y-coord of bounding rectangle origin */
     int width,          /*width of bounding rectangle   */
     int height          /*height of bounding rectangle */
     )
```

DESCRIPTION

This routine draws a single line ellipse. The ellipse is located relative to the logical origin.
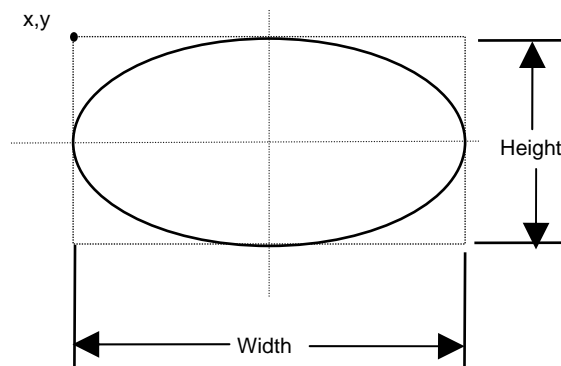
All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

filledEllipse()

Ellipse

enableStereo

NAME

enableStereo – enable or disable stereo output

SYNOPSIS

```
void enableStereo
    (
    int state                  /* 1 = on, 0 = off */
    )
```

DESCRIPTION

EnableStereo() enables or disables the special stereo output mode of the Borealis 3 graphics chip. The output format is alternating line. A control line toggles to indicate the active line (left or right). The stereo image is comprised of a left image in the normal frame buffer (page 0) and a right image in page 1. The graphics controller alternates outputting a line from page 0, then a line from page 1, then the second line from page 0, and the second line from page 1, etc.

Typically, this stereo format is usable only on head mounted displays (HMDs) with two LCDs, or with an external line blanking device.

INCLUDE FILES

sdl.h

SEE ALSO

setWritePage ()

filledArc

NAME

filledArc - draws a filled sector or chord

SYNOPSIS

```
void filledArc
     (
     int x,                /* x-coord of bounding rectangle origin */
     int y,                /* y-coord of bounding rectangle origin */
     int width,            /* width of bounding rectangle */
     int height,           /* height of bounding rectangle */
     int start_angle,      /* start angle (in 32nds of a degree) */
     int end_angle         /* end angle (in 32nds of a degree) */
     )
```

DESCRIPTION

This routine draws a filled sector or filled chord depending on the current filled arc mode, set by *setArcMode().* The arc portion is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. Positive angles indicate counterclockwise motion, and negative angles indicate clockwise motion. If the magnitude of either angle is greater than 360 degrees, it is set to the angle modulo 360 degrees.

All drawing primitives are affected by the current fill style which is specified with ***setFillStyle().*** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

arc(), filledArc2(), setArcMode(), setFillStyle()
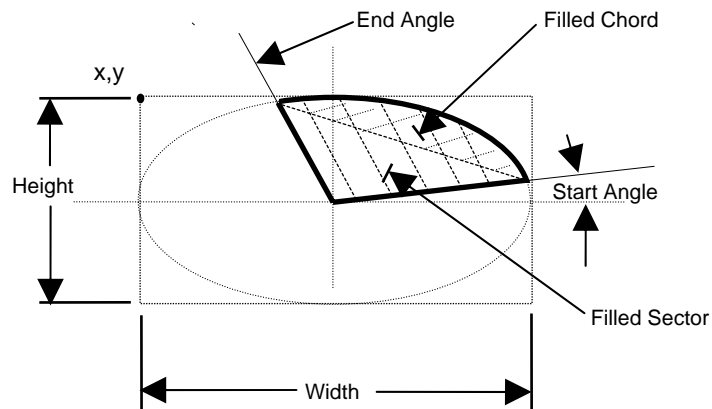
Filled Sector or Filled Chord

filledArc2

NAME

filledArc2 - draws a filled sector or chord

SYNOPSIS

```
void filledArc2
        (
        int x,              /* x-coord of bounding rectangle origin */
        int y,              /* y-coord of bounding rectangle origin */
        int width,          /* width of bounding rectangle */
        int height,         /* height of bounding rectangle */
        int start_angle,    /* start angle (in 32nds of a degree) */
        int angle_extent    /* angle extent (in 32nds of a degree) */
        )
```

DESCRIPTION

This routine draws a filled sector or filled chord depending on the current filled arc mode, set by *setArcMode().* The arc portion is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. Positive angles indicate counterclockwise motion, and negative angles indicate clockwise motion. If the magnitude of *start_angle* is greater than 360 degrees, it is set to the angle modulo 360 degrees. If the magnitude of *angle_extent* is greater than 360 degrees, it is truncated to 360 degrees.

All drawing primitives are affected by the current fill style which is specified with ***setFillStyle().*** *Use SOLID_FILL for drawing solid lines.*

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

arc2(), filledArc(), setArcMode(), setFillStyle()
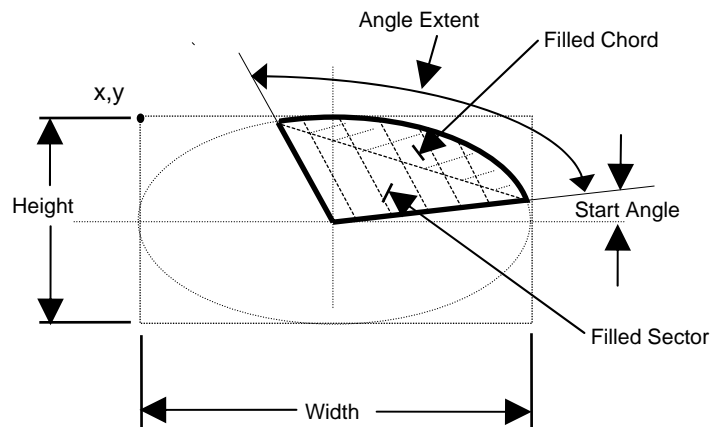
Filled Sector or Filled Chord

filledCircle

NAME

      filledCircle - draws a filled circle

SYNOPSIS

      `void filledCircle(int x, int y, int radius)`

DESCRIPTION

      This function draws a filled circle centered about x,y, and of the radius value specified. x,y is relative to the logical origin. The circle can be filled with a solid color, a stipple fill, or an opaque stipple fill. See *setFillStyle().*

INCLUDE FILES

      sdl.h, extern.h

SEE ALSO

      circle(), setFillStyle()

Filled Circle

filledEllipse

NAME

filledEllipse - draws a filled ellipse
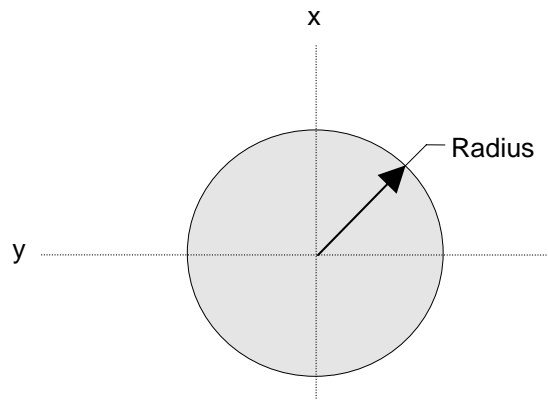
SYNOPSIS

```
void filledEllipse
      (
      int x,              /* x-coord of bounding rectangle origin */
      int y,              /* y-coord of bounding rectangle origin */
      int width,          /* width of bounding rectangle  */
      int height          /* height of bounding rectangle */
      )
```

DESCRIPTION

This routine draws a filled ellipse. The ellipse is specified by a bounding rectangle with an origin at x,y, relative to the logical origin. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. The ellipse is filled using the current fill style: solid, stipple, or opaque stipple.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO
filledArc(), setFillStyle()
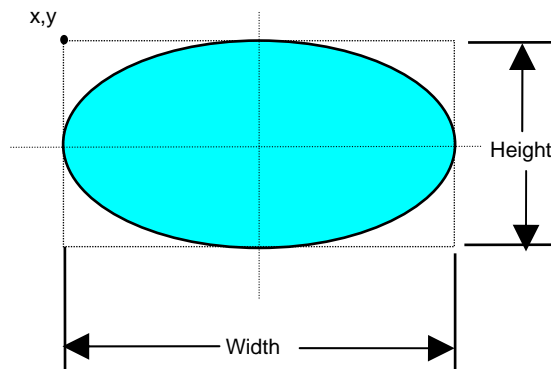
Filled Ellipse

filledPolygon

NAME

filledPolygon - draws a filled polygon

SYNOPSIS

```
void filledPolygon
      (
      int ptscount,    /* number of points in the array          */
      sPoint *points   /* pointer to array of points in the path */
      )
```

DESCRIPTION

This routine fills the polygon formed by the connected lines forming a closed path. If the path is not closed, (ie the last line does not end where the first line started) the path is closed automatically. All coordinates are relative to the logical origin.

The polygon is filled using the current fill style (set by *setFillStyle()*) and according to the rule specified by the *setFillRule()* function. The foreground color is used for solid color and stipple fills, and both the foreground and background colors are used for opaque stipple fills.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

polyline(), setFillRule(), setFillstyle(), setPattern()
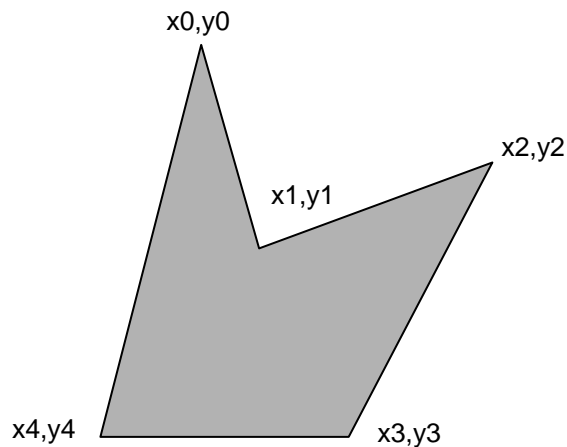
Filled Polygon

filledRectangle

NAME

filledRectangle - draws a filled rectangle

SYNOPSIS

```
void filledRectangle
     (
     int x,                    /* x-coord of upper left corner */
     int y,                    /* y-coord of upper left corner */
     int width,                /* width  */
     int height                /* height */
     )
```

DESCRIPTION

This routine draws a filled rectangle. The *x* and *y* coordinates are relative to the logical origin, and specify the coordinates for the upper left corner of the rectangle.

The rectangle is filled using the current fill style (set by *setFillStyle()*) and uses the foreground color for solid color and stipple fills, and both the foreground and background colors for opaque stipple fills.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

rectangle(), setFillStyle()
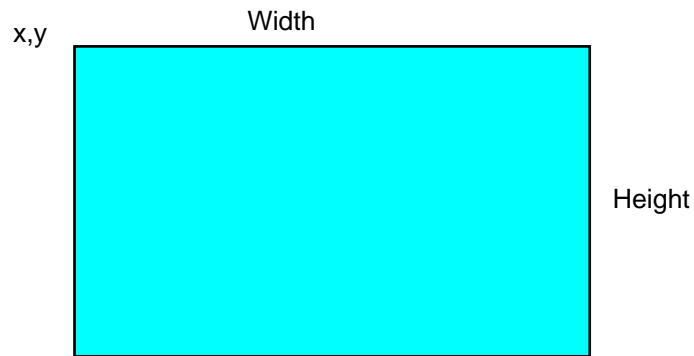
Filled Rectangle

flushKeyboard

NAME

flushKeyboard - flush the keyboard queue

SYNOPSIS

```
void flushKeyboard(void)
```

DESCRIPTION

FlushKeyboard() clears the keyboard queue on the graphics board of any key pushes.

INCLUDE FILES

drv/rgkeybd.h

SEE ALSO

keyboardRead(), keyboardReady()

flushMouse

NAME

flushMouse - flush the mouse queue

SYNOPSIS

```
void flushMouse(void)
```

DESCRIPTION

FlushMouse() clears the mouse queue on the graphics board of any mouse events.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

mouseRead(), mouseReady()

getColor

NAME

getColor - reads an rgb color from color palette

SYNOPSIS

```
void getColor
      (
      int index,          /* palette index */
      int *red,           /* points to save location for red */
      int *green,         /* points to save location for green */
      int *blue           /* points to save location for blue */
      )
```

DESCRIPTION

This routine reads the rgb color values from the specified index value of the systems color palette and stores the values at the addresses, red, green, and blue.

INCLUDE FILES

sdl.h, extern.h, colors.h

SEE ALSO

storeColor()

getFontStruct

NAME

getFontStruct - gets font parameters

SYNOPSIS

```
void getFontStruct
    (
    sFontStruct  *fs          /* pointer to structure with */
                              /* font parameters */
    )
```

DESCRIPTION

This routine fills in the structure, *fs,* with the parameters for the current font.

Proportionally spaced fonts typically fill in the *fontAscent* and *fontDescent* structure members, while constant space fonts typically use *minbounds.ascent* and *minbounds.descent* for the overall font ascent and descent values.

The *FontStruct* structure is defined as follows:

```
typedef  struct  tagFontInfo
{
short width,        /* width of character in pixels    */
     ascent,        /* number of pixels above baseline  */
     descent;       /* number of pixels below baseline  */
}sFontInfo, *spFontInfo;


typedef  struct  tagFontStruct
{
char           fontName[12];/* name of font              */
unsigned long  fontID;       /* unique font identifier    */
sFontInfo      minbounds,    /* smallest char dimensions */
               maxbounds;    /* largest char dimensions   */
unsigned short  fontAscent;   /* maximum font ascent      */
unsigned short  fontDescent;  /* maximum font descent     */
}sFontStruct,  *spFontStruct;
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

drawText(), getTextWidth(), setFont()

getFramebufPtr

NAME

getFramebufPtr – gets current value of the framebuffer pointer

SYNOPSIS

unsigned char *getFramebufPtr(void**)**

DESCRIPTION

This routine looks up the current value of the framebuffer pointer. This pointer is only valid for the current device context. If the active graphics device is changed, or a different write page is selected, the pointer may become invalid, so this function should be called again to obtain the updated value.

RETURNS

unsigned char * /* pointer to framebuffer memory */

INCLUDE FILES

sdl.h

SEE ALSO

setGraphicsDevice(), setWritePage()

getImage

NAME

getImage - copy area of display to host memory

SYNOPSIS

```
void getImage
    (
    int x,                    /* source upper left x-coord */
    int y,                    /* source upper left y-coord */
    int width,                /* source width */
    int height,               /* source height */
    unsigned char *buff       /* host memory buffer */
    )
```

DESCRIPTION

GetImage() copies a rectangular area of the display to a host memory buffer pointed to by *buff*. The user must allocate enough memory to contain the image. The amount of memory, in bytes, can be calculated as:

size = width * height;

In 256 color video modes, each pixel is stored as a single byte with a value from 0 to 255. In 16 color video modes, each pixel is stored as a single byte with a value from 0 to 15.

The image source data is taken from the current write page and is clipped to the current clipping rectangle.

INCLUDE FILES

sdl.h

SEE ALSO

copyImage(), copyPageImage(), putImage()

Video   Screen
(VRAM)
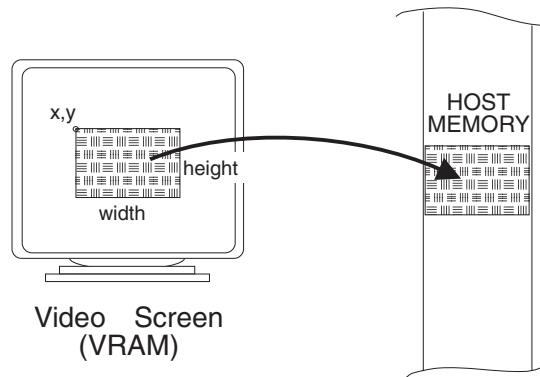
getMouseXY

NAME

getMouseXY - get the current mouse location

SYNOPSIS

```
void getMouseXY
     (
     int *mx,            /* pointer to returned mouse X position */
     int *my             /* pointer to returned mouse Y position */
     )
```

DESCRIPTION

GetMouseXY() gets the current location of the mouse cursor.

The returned position is the location of the cursor "hot-spot" on the screen, based on absolute screen coordinates.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

mouseRead(), mouseReady(), mouseCursorXY(), mouseRect(), mouseScale(), setMouseParam()

getPixel

NAME

getPixel - gets pixel at x,y

SYNOPSIS

```
unsigned long getPixel
        (
        int x,                    /* pixel's x coordinate*/
        int y                     /* pixel's y coordinate */
        )
```

DESCRIPTION

This routine returns the value of the pixel at x,y. The x,y coordinates are relative to the logical origin.

INCLUDE FILES

sdl.h, extern.h

RETURNS

unsigned long   /* 32 bit value of pixel at x,y */

SEE ALSO

drawPixel()

getTextWidth

NAME

getTextWidth - gets width of text string

SYNOPSIS

```
int getTextWidth
    (
    char *string                /* string to measure */
    )
```

DESCRIPTION

This routine returns the width in pixels of the null terminated text string pointed to by *string.* Regardless of the current text drawing direction, *getTextWidth()* always returns the width as if the string is drawn horizontally left to right. If passed a null string, it returns zero for the length.

INCLUDE FILES

sdl.h, extern.h

RETURNS

int   /* string_width, -1 on error */

SEE ALSO

drawText(), getFontStruct()

initGraphics

NAME

initGraphics - initializes *SDL* and the graphics hardware

SYNOPSIS

```
int initGraphics(int argc, char **argv)
```

DESCRIPTION

This routine initializes *SDL* and the graphics hardware by executing the code contained in *userinit.c*. This file must be modified by the user to initialize the user's system before graphics processing can occur. Typically, *userinit.c* installs the selected fonts, initializes the graphics hardware, and performs power up initialization required by the user's system to be able to run *SDL.* The *argv* parameters may be used to pass command line arguments to *initGraphics().*

To modify the default parameters, the user must edit the file *userinit.c*, recompile it, and link it.

Arguments passed to *initGraphics()* are driver specific. Currently, the only argument used by all drivers is **-v**, which overides the video mode selection made in *userinit.c*. See the include file *sdl.h* for a list of video modes. Using a value of *-1*, with the **-v** switch will print a list of supported video modes. Using a flag of **-h** will print a help message showing available options.

Since VxWorks does not generally support argc/argv style arguments, all options must be passed as a single string as the first argument to *initGraphics()*. For example:

```
initGraphics("-v 3 -C 1", 0);
```

Some PowerPC boards use host bridges that support two independent PCI buses. In this case, when using VxWorks, you must sometimes tell *SDL* when the graphics board is installed on the second channel by using the "**-C 1**" option.

INCLUDE FILES

sdl.h, extern.h

RETURNS

int /* 0 on success, -1 on error */

SEE ALSO

closeGraphics()

keyboardRead

NAME

keyboardRead - read keyboard character

SYNOPSIS

```
void keyboardRead(unsigned short *kdata)
```

DESCRIPTION

KeyboardRead() is a blocking call to read a character from the keyboard buffer. The keycode is returned in the least significant byte of the short word pointed to by kdata. For Rastergraf VME graphics boards, extended keycodes have the most significant byte of the short word set to 0xff; standard keycodes have the most significant byte set to zero.

INCLUDE FILES

drv/rgkeybd.h

SEE ALSO

flushKeyboard(), keyboardReady()

keyboardReady

NAME

keyboardReady - check for keycodes in the keyboard queue

SYNOPSIS

`int keyboardReady(void)`

DESCRIPTION

KeyboardReady() is a non-blocking call to check if any characters are available in the keyboard buffer.

INCLUDE FILES

drv/rgkeybd.h

SEE ALSO

flushKeyboard(), keyboardRead()

line

NAME

line - draws a line

SYNOPSIS

```
void line
    (
    int x0,                 /* x-coord of first endpoint  */
    int y0,                 /* y-coord of first endpoint  */
    int x1,                 /* x-coord of second endpoint */
    int y1                  /* y-coord of second endpoint */
    )
```

DESCRIPTION

This routine draws a single pixel line from (*x0,y0*) to (*x1,y1*). The coordinates are relative to the logical origin. The line is drawn using the current fill style (set by *setFillStyle()*) and and line width (set by *setLineWidth()*) uses the foreground color when the fill style is solid color or stipple, and both the foreground and background colors when the fill style is opaque stipple.
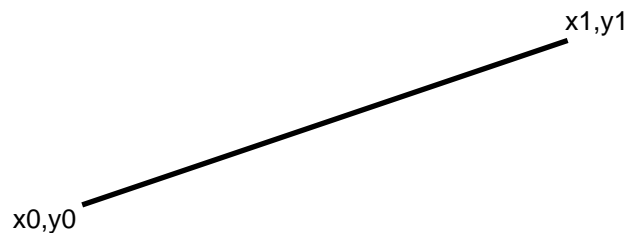
Wide lines are only available when the fill style is SOLID. A line width of 1 forces use of the wide line function, while a line width of zero uses the optimized zero-width line drawing functions.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

polyline(), dashedLine(), setLineWidth()

x1,y1

x0,y0

Line

mouseCursorOn

NAME

mouseCursorOn - turn the mouse cursor on or off

SYNOPSIS

```
void mouseCursorOn
    (
    int state                /* 1 = on, 0 = off */
    )
```

DESCRIPTION

MouseCursorOn() turns the mouse cursor on and off. The cursor type can be changed by using *setMouseCursor()*. The cursor position can be moved by using *mouseCursorXY()*. The screen contents under the cursor are automatically saved when the cursor is enabled (*state* = 1) and restored when the cursor is disabled (*state* = 0).

When the mouse is enabled in the driver, the default mouse cursor is automatically displayed. This function can be used to turn it off, then on again as required when changing the mouse setup.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

getMouseXY(), mouseCursorXY(), mouseRead(), mouseReady(), mouseRect(), mouseScale(), setMouseCursor(), setMouseParam()

mouseCursorXY

NAME

mouseCursorXY - move the mouse cursor to a new position

SYNOPSIS

```
void mouseCursorXY
    (
    int x,                        /* new x-coord of position */
    int y                         /* new y-coord of position */
    )
```

DESCRIPTION

MouseCursorXY() changes the current x-y mouse address and specifies the starting point for the mouse cursor on the screen. If the mouse cursor already being displayed, it is repositioned to the x-y location specified.

The position is the location of the cursor "hot-spot" on the screen, based on absolute screen coordinates.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

mouseCursorOn(), mouseRect(), mouseScale(),setMouseCursor(), setMouseParam()

mouseRead

NAME

mouseRead - read a mouse event

SYNOPSIS

```
void mouseRead
    (
    sMouseEvent *mse        /* pointer to returned mouse event */
    )
```

DESCRIPTION

MouseRead() reads a single mouse event from the mouse queue into the event structure *mse*. This is a blocking call that does not return until a mouse event has been read. Use the non-blocking *mouseReady()* call to query if any events are in the queue. The **sMouseEvent** structure is defined as:

```
typedef struct tagMouseEvent {
    short report_x;         /* 'x' location on mouse input event */
    short report_y;         /* 'y' location on mouse input event */
    unsigned short mouse_sw;/* switch image on mouse input event */
    unsigned short timer;   /* free running 16 bit 'timer'
                               when event occurred */
    short cursor_x;         /* 'x' location (continuosly updated) */
    short cursor_y;         /* 'y' location (continuosly updated) */
} sMouseEvent, *spMouseEvent;
```

The returned positions are the location of the cursor "hot-spot" on the screen, based on absolute screen coordinates.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

flushMouse(), mouseReady(), getMouseXY()

mouseReady

NAME

mouseReady - check for mouse events in the mouse queue

SYNOPSIS

`int mouseReady(void)`

DESCRIPTION

MouseReady() is a non-blocking call to check if any events available in the mouse queue. It is used to check if an event is available before calling the blocking *mouseRead()* function.

INCLUDE FILES

drv/rgmouse.h

RETURNS

Zero on success, non-zero on failure or when the mouse is not enabled in the driver.

SEE ALSO

flushMouse(), mouseRead()

mouseRect

NAME

mouseRect - set window limits for the mouse cursor

SYNOPSIS

```
void mouseRect
    (
    int x,                      /* upper left x-coord */
    int y,                      /* upper left y-coord */
    int width,                  /* width of bounding area */
    int height                  /* height of bounding area */
    )
```

DESCRIPTION

MouseRect() sets a boundary window for the mouse cursor. The mouse cursor movement is pinned at the new window edges. The bounding rectangle is based on absolute screen coordinates.

MouseRect() does not automatically set the `MSECSR_WINDOW` bit in mouse parameter zero. This must be done using the *setMouseParam()* function when the boundary window is smaller than the screen size.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

getMouseXY(), mouseCursorOn(), mouseCursorXY(), mouseScale(), setMouseCursor(), setMouseParam()

mouseScale

NAME

mouseScale - set scale factors for the mouse cursor

SYNOPSIS

```
void mouseScale
     (
     int xscale,              /* scaling in x direction */
     int yscale               /* scaling in y direction */
     )
```

DESCRIPTION

MouseScale() sets the scale factors used for the mouse cursor. Incoming mouse movement is multiplied by the scale factors *factors* in calculating the updated mouse cursor position. The scale factors are in a 16-bit fixed point format with 8-bit integer and 8-bit fraction. The default scaling is +1.000 (0x0100).

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

getMouseXY(), mouseCursorOn(), mouseCursorXY(), mouseRect(), setMouseCursor(), setMouseParam()

panelType

NAME

      panelType – report hardware configured display panel type

SYNOPSIS

      `int panelType (void)`

DESCRIPTION

      This function returns the display panel type as reported by the graphics hardware configuration bits. The return value is driver specific.

INCLUDE FILES

      sdl.h

RETURNS

      int /* up to 32-bits of driver/hardware specific info */

polyline

NAME

polyline - draws a polyline

SYNOPSIS

```
void polyline
     (
     int num_pts,              /*number of points in the array */
     sPoint *ptr_to_coord_list /*pointer to array of points    */
     )
```

DESCRIPTION

This routine draws lines connecting each pair of points in the array of *sPoint* structures. It draws the lines, connecting the points in the order listed in the array. If the lines intersect, the intersecting pixels are drawn again and pixel processing will behave accordingly. All coordinates are relative to the logical origin.

The lines are drawn using the current fill style (set by *setFillStyle()*)and line width (set by *setLineWidth()*) and use the foreground color when the fill style is solid color or stipple, and both the foreground and background colors when the fill style is opaque stipple.

Wide lines are only available when the fill style is SOLID. A line width of 1 forces use of the wide line function, while a line width of zero uses the optimized zero-width line drawing functions.

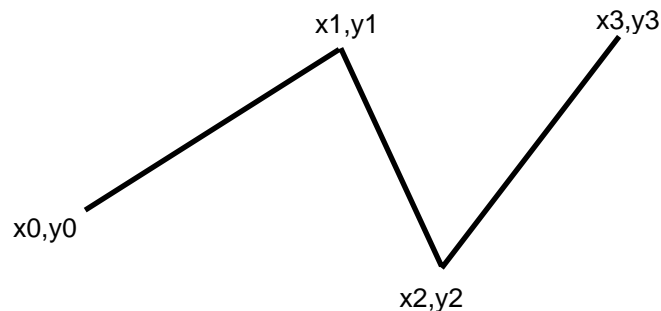The **sPoint** structure is defined as follows:

```
typedef struct tagPoint
{
short x;
short y;
} sPoint;
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

line(), dashedPolyline(), setLineWidth()

Polyline

putImage

NAME

putImage - copy image in host memory to the display

SYNOPSIS

```
void putImage
    (
    unsigned char *buff,     /* host memory buffer */
    int width,               /* source width */
    int height,              /* source height */
    int x,                   /* destination upper left x-coord */
    int y                    /* destination upper left y-coord */
    )
```

DESCRIPTION

PutImage() copies an image in host memory to a rectangular area of the display. The host memory buffer is a raw pixmap with one byte per pixel and pitch equal to the width of the image.

Each byte in the source image represents a pixel color in the current color palette. In 256 color video modes, each pixel is a single byte with a value from 0 to 255. In 16 color video modes, each pixel is a single byte with a value from 0 to 15.

The image data is drawn to the current write page and is clipped to the current clipping rectangle.

INCLUDE FILES

sdl.h

SEE ALSO

copyImage(), copyPageImage(), getImage()
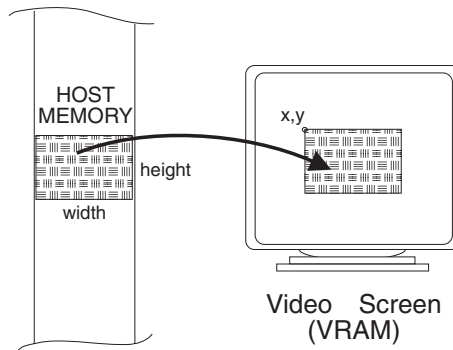
rectangle

NAME

rectangle - draws a rectangle

SYNOPSIS

```
void rectangle
    (
    int x,                    /* x-coord of upper left corner */
    int y,                    /* y-coord of upper left corner */
    int width,                /* width  */
    int height                /* height */
    )
```

DESCRIPTION

This routine draws a rectangle. The *x* and *y* coordinates are relative to the logical origin of the screen.

The rectangle edges are drawn using the current fill style (set by *setFillStyle()*)and line width (set by *setLineWidth()*).

Wide lines for edges are only available when the fill style is SOLID. A line width of 1 forces use of the wide line function, while a line width of zero uses the optimized zero-width line drawing functions.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO
filledRectangle(), dashedRectangle()
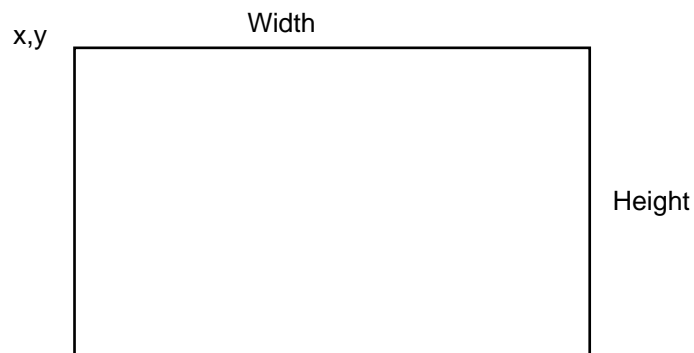
Rectangle

setArcMode

NAME

setArcMode - specifies a filled arc to be a filled sector or filled chord

SYNOPSIS

```
void setArcMode
        (
        int fillMode          /* 0==filled sector; 1==filled chord */
        )
```

DESCRIPTION

This routine specifies what the *filledArc()* function will draw: a filled sector or filled chord.
The sector or chord can be filled with a solid color, stipple pattern, or an opaque stipple pattern.
The arc modes are defined in *sdl.h* as follows:

\# define SECTOR_MODE 0
\# define CHORD_MODE   1

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

filledArc(), setFillStyle()
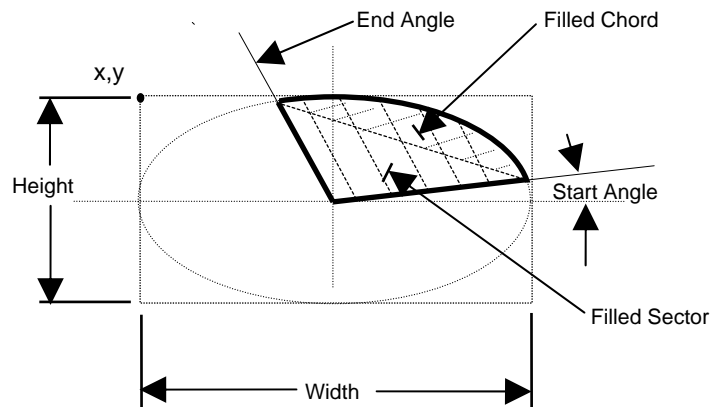
Filled Sector or Filled Chord

setBackground

NAME

setBackground - sets the background color

SYNOPSIS

```
void setBackground
     (
     unsigned long color      /* new background color */
     )
```

DESCRIPTION

This routine sets the current background color by loading the global variable _bcolor_ with the unsigned 32 bit value passed to it. The background color is used for the character cell background color when transparency is off, for a stipple pattern background color, and for dashed lines when double dash is specified. The value loaded into the global variable is device dependent, and will be converted as necessary by the *graphics driver* to be compatible with the graphics hardware.

Most graphics drivers support 8, 16, and 24 or 32 bit color values.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setForeground()

setClipRect

NAME

setClipRect - sets the clipping rectangle.

SYNOPSIS

```
void setClipRect
    (
    int x,                  /* x coordinate of clip origin */
    int y,                  /* y coordinate of clip origin */
    int width,              /* width of clipping rectangle */
    int height              /* height of clipping rectangle */
    )
```

DESCRIPTION

This routine sets the parameters of the clipping rectangle on the screen. All primitives are clipped to this area. Height and width must be positive or zero. A width or height of zero will prevent any graphics from appearing on the screen. The default clip rectangle is the entire visible screen.   The clipping rectangle is positioned relative to the screen coordinates.

INCLUDE FILES

sdl.h, extern.h
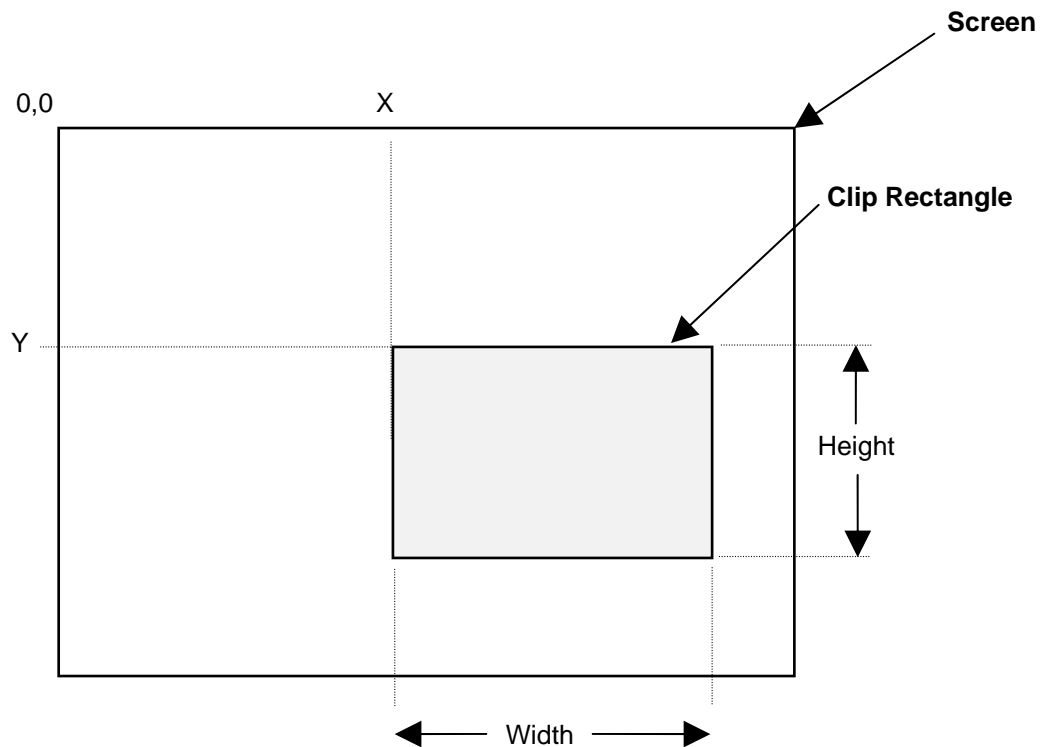
Clip Rectangle

setDashOffset

NAME

setDashOffset – sets a new pattern offset for dashed lines

SYNOPSIS

```
void setDashOffset
      (
      int dashOffset              /* pixel offset for pattern start*/
      )
```

DESCRIPTION

The *dashOffset* specifies the starting point from the beginning of the pattern in pixels. This feature allows the user to specify where to start in the pattern.   The dashed line pattern does not restart each time it is used, allowing dashed lines to continue around the corner of a rectangle, or polyline. The user has the option of restarting the pattern by specifying a *dashOffset* value of zero.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setDashPattern(), setDashStyle()

setDashPattern

NAME

setDashPattern - specifies the dash line pattern for dashed lines

SYNOPSIS

```
void setDashPattern
     (
     int numDashes,      /* number of entries in the dash list   */
     unsigned char *dashList, /* dash pattern list               */
     int dashOffset          /* pixel offset for pattern start*/
     )
```

DESCRIPTION

This routine specifies the pattern to be used for dashed lines. The first parameter, *numDashe*s, specifies the number of entries in the *dashList.*   The *dashList* tells the line-drawing routine the sequence of pixels to write and skip. If, for example, the *dashList* contained [2,3,1,4], the line drawing routine would draw two pixels in the foreground color, skip three, draw one, skip four, and then repeat the pattern until the line was completed, assuming *on/off* was the active dashed line style. If *double dashed* was the active dashed line style, the values in the even index locations ([0], [2], etc.) would be written in the foreground color as before, and the values in the odd index locations ([1], [3], etc.) would be written in the background color.

There must be at least one element in the specified *dash list.* All of the elements must be nonzero.

The *dashOffset* specifies the starting point from the beginning of the pattern in pixels. This feature allows the user to specify where to start in the pattern.   The dashed line pattern does not restart each time it is used, allowing dashed lines to continue around the corner of a rectangle, or polyline. The user has the option of restarting the pattern by specifying a *dashOffset* value of zero.

The number of entries in the list is limited to 255.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setDashOffset(), setDashStyle()

━ ━━ ━━ ━━━ ━━ ━━━

On/Off Dashed Pattern

━ ━━━ ━━━ ━━━ ━━━

Double Dashed Pattern

setDashStyle

NAME

setDashStyle - specifies the dash line style, On/Off or Double Dash

SYNOPSIS

```
void setDashStyle
    (
    int dashStyle            /* 0 == on/off;  1 == double dash */
    )
```

DESCRIPTION

This routine specifies the dashed line style to be used for dashed lines.

On/Off dashed lines draw one color dashed lines.   The foreground color is used.
Double Dashed lines are two color dashed lines.   Both the foreground and background colors are used.

The dash style is defined in *sdl.h* as:

#define ONOFF_DASH   0
#define DOUBLE_DASH 1

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setDashOffset(), setDashPattern()

‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑

On/Off Dashed Pattern

‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑ ‑

Double Dashed Pattern

setDisplayPage

NAME

setDisplayPage - set the current display page

SYNOPSIS

```
void setDisplayPage
     (
     int pagenum                    /* graphics memory page number */
     )
```

DESCRIPTION

SetDisplayPage() selects the page number (for graphics hardware with more than one page of video memory) that is to be displayed. Page numbering starts at zero.

INCLUDE FILES

sdl.h

SEE ALSO

setWritePage()

setFillRule

NAME

setFillRule - specifies the fill rule for polygon fills

SYNOPSIS

```
void setFillRule
    (
    int fillrule            /* rule for filledPolygon */
    )
```

DESCRIPTION

This routine specifies the fill rule to be used for filling polygons.

Specify 0 for EVENODD or 1 for WINDING.   The default is EVENODD.

The fill-rule determines how self-intersecting polygons are filled, by defining which pixels are inside (drawn). For EVENODD, a point is inside if an infinite ray originating at that point as crosses the path an odd number of times. For WINDING, a point is inside if an infinite ray originating at that point crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the point. A counterclockwise segment is one that crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can simply choose a different ray that is not coincident with a segment.

For both EVENODD and WINDING, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside if the center point of the pixel is inside and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).

The fill rule is defined in *sdl.h,* as:

```
#define EVENODD   0
#define WINDING    1
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

filledPolygon()

setFillStyle

NAME

setFillStyle - specifies a fill style of solid, stipple, or opaque stipple

SYNOPSIS

```
void setFillStyle
      (
      int fillStyle                /* 0== solid,  1== stipple */
                                   /* 2==opaque stipple */
      )
```

DESCRIPTION

This routine specifies the fill style for circles, ellipses, arcs (sectors and chords) polygons, and rectangles. **The fill style affects all other drawing primitives**. For example, a line will be drawn with the stipple pattern if the fill style is set to stipple.

Only the foreground color is used for solid or stipple fills. Both the foreground and background colors are used for opaque stipple fills.

The fill style is defined in *sdl.h* as:

```
#define SOLID_FILL     0
#define STIPPLE_FILL   1
#define OPAQUE_FILL  2
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

filledPolygon(), filledRectangle(), filledArc()

setFont

NAME

setFont - sets the current font

SYNOPSIS

```
void setFont
    (
    int fontIndex            /* selects the current font */
    )
```

DESCRIPTION

This routine selects the current font. This function takes as a parameter, the font name as defined in *fonts.h*, or the corresponding index value for the font.

The user installs fonts by adding the font names to the global array *_fontTable* specified in the file *fonts.c*.

The default fonts for *SDL* are shown below. Also see Appendix B.

```
#define HELVR12      0    /* Helvetica 12pt Normal Prop. Spaced      */
#define HELVR08      1    /* Helvetica  8pt Normal Prop. Spaced      */
#define HELVR10      2    /* Helvetica 10pt Normal Prop. Spaced      */
#define HELVR14      3    /* Helvetica 14pt Normal Prop. Spaced      */
#define HELVR18      4    /* Helvetica 18pt Normal Prop. Spaced      */
#define HELVR24      5    /* Helvetica 24pt Normal Prop. Spaced      */
#define CLR6X6       6    /* Clear 6x6 Normal Fixed Width            */
#define CLR8X8       7    /* Clear 8x8 Normal Fixed Width            */
#define CLR8X16      8    /* Clear 8x16 Normal Fixed Width           */
#define FIX12X24RK   9    /* 12x24 Normal Fixed Width Roman-Kana     */
#define HELVB14      10   /* Helvetica 14pt Bold Prop. Spaced        */
#define HELVB18      11   /* Helvetica 18pt Bold Prop. Spaced        */
#define HELVB24      12   /* Helvetica 24pt Bold Prop. Spaced        */
#define HELVBO14     13   /* Helvetica 14pt BoldOblique Prop. Spaced */
#define HELVBO18     14   /* Helvetica 18pt BoldOblique Prop. Spaced */
#define HELVBO24     15   /* Helvetica 24pt BoldOblique Prop. Spaced */
#define RGBOLD36     16   /* Peritek Bold Fixed 36x78                */
#define RGSWISS44    17   /* Peritek Swiss Fixed 44x70               */
```

INCLUDE FILES

sdl.h, extern.h, fonts.h

SEE ALSO

drawText()

setForeground

NAME

setForeground - sets the current foreground color

SYNOPSIS

```
void setForeground
      (
      unsigned long color      /* new foreground color */
      )
```

DESCRIPTION

This routine sets the current foreground color. All primitives are drawn using the foreground color. The actual value used for the color is hardware dependent. This functions loads the 32 bit color parameter passed to it into the global variable _fcolor.

The *graphics driver* must convert the 32 bit color parameter to the appropriate value for the display hardware. Most graphics drivers support 8, 16, and 24 or 32 bit color values.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setBackground()

setGraphicsDevice

NAME

setGraphicsDevice - set the current graphics display device

SYNOPSIS

```
void setGraphicsDevice
    (
    int devnum                  /* graphics device number */
    )
```

DESCRIPTION

SetGraphicsDevice() selects the graphics device for graphics hardware with more than one channel, for systems with more than one graphics device installed or for graphics hardware with multiple outputs formats, such as SVGA chips with CRT, LCD and/or video outputs. Device numbering starts at zero and is driver specific.

All drawing primitives, keyboard operations and mouse operations apply to the currently selected graphics device. When used to switch between multiple graphics devices, the current display page, write page, clipping rectangle and mouse status is saved for each graphics device to make the switching between multiple graphics devices as transparent as possible.

Depending on the particular driver and graphics hardware used, the **userinit.c** file may need to be edited to set the device addresses and other features that are unique for each device.

The device ids used with typical PMC graphics boards defined in *sdl.h* as:

```
#define CRT_DEVICE        0x000001   /* default */
#define LCD_DEVICE        0x000002
#define NTSC_TV_DEVICE    0x000004
#define PAL_TV_DEVICE     0x000008
```

As of SDL version 3.1, additional support was added to support multiple graphics controllers on a single graphics board, and multiple graphics boards in a system. The following defines can be or'd together to select a specific board and controller combination:

```
#define GDEV_0            0x000000   /* first (or only) graphics controller */
#define GDEV_1            0x000100   /* second graphics controller */
#define GDEV_2            0x000200   /* third graphics controller */
#define GDEV_3            0x000300   /* fourth graphics controller */

#define GBRD_0            0x000000   /* first (or only) graphics board */
#define GBRD_1            0x010000   /* second graphics board */
#define GBRD_2            0x020000   /* third graphics board */
#define GBRD_3            0x030000   /* fourth graphics board */
```

As of SDL version 3.2, additional flags were added to support the dual channel display engines in the SM731 and M9-based graphics boards. These flags are:

```
#define VP2CRTDAC        0x000010   /* VP engine feeds CRT DAC */
#define VP2DIGOUT        0x000020   /* VP engine feeds flat panel output */
#define VP2LVDS2         0x000040   /* VP engine feeds LVDS2 output */
#define NULL_DEVICE      0x000080   /* keep previous output settings */
```

INCLUDE FILES

sdl.h

SEE ALSO

setMode()

setLineWidth

NAME

setLineWidth - sets the current line width

SYNOPSIS

```
void setLineWidth
    (
    int width                    /* new line width */
    )
```

DESCRIPTION

This routine sets the current line width. All dashed and solid line primitives (line, polyline, rectangle) use the line width when drawing lines. A line width of 1 may not be the same as a line width of zero for diagonal lines as they use different low level functions to draw the lines.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

dashedLine(), dashedPolyline(), line(), polyline()

setMode

NAME

setMode - set displayed video mode and attributes

SYNOPSIS

```
int setMode
    (
    char *mode_args          /* init args in string form */
    )
```

DESCRIPTION

SetMode() is used to change the video mode (resolution and bits per pixel) of the currently active graphics device. Other attributes, such as virtual width and height, refresh clock frequency, etc., can also be set, depending on the specific graphics board and driver. This function can be used (where supported) in place of calling closeGraphics() followed by initGraphics() to reinitialize the board. The mode_args string is the list of initialization arguments in string form.

SetMode() is usually used with the sm731 driver (Colos/Duros/Stratus/Tropos boards) to set a different resolution on the second display channel. When the second channel is initialized (via *setGraphicsDevice()*), it acquires the same video mode (resolution, bits per pixel, etc.) as the primary channel, so by using *setMode()* the primary (flat panel) channel can remain at its original resolution (say 1024x768) and the secondary CRT channel could be configured to display a different image at say 640x480 resolution.

EXAMPLE

```
int myprog(int argc, char **argv)
{
    /* initialize main (FP) channel */
    if (initGraphics(argc, argv)) {
        printf("initialization failed!\n");
        return -1;
    }
    /* initialize VP channel */
    setWritePage(4);  /* different memory region for channel 2 */
    setGraphicsDevice(GDEV_1|VP2CRTDAC);
    /* change VP video mode to 640x480x8 at 60 Hz */
    if (setMode("-v 1 -c 60"))
        return -1; /* failed to change video mode */
    /* select FP channel again */
    setGraphicsDevice(GDEV_0|NULL_DEVICE);
    setWritePage(0);  /* original memory region for channel 1 */
        :
}
```

INCLUDE FILES

sdl.h

RETURNS

int /* 0 on success, -1 on error */

SEE ALSO

closeGraphics(), initGraphics(), setGraphicsDevice()

setMouseCursor

NAME

setMouseCursor - set mouse cursor type and colors

SYNOPSIS

```
void setMouseCursor
     (
     unsigned long csr_id,    /* predefined cursor num. or addr */
     unsigned long color1,    /* bottom color of the cursor */
     unsigned long color2     /* top color of the cursor */
     )
```

DESCRIPTION

SetMouseCursor() selects one of the predefined mouse cursors, or a user defined cursor symbol. If the value of *csr_id* is less than 128, it is assumed to be an index into the table of default cursors; otherwise *csr_id* is assumed to be a pointer to a user defined cursor structure which has been previously downloaded to the graphics board. A list of predefined cursors is listed in Appendix D. The default cursor is a left arrow.

Only one mouse cursor is available per graphics device. If a mouse cursor is already on, it must be turned off (using *mouseCursorOn()*) before calling this function, then turned on again.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

getMouseXY(), mouseCursorOn(), mouseCursorXY()

setMousePage

NAME

setMousePage - set the mouse display page

SYNOPSIS

```
int setMousePage
    (
    int page                    /* graphics memory page */
    )
```

DESCRIPTION

SetMousePage() sets graphics memory page that is used for displaying the mouse cursor. The page always refers to the currently selected graphics device (if there is more than one device in the system) and currently selected channel (underlay or overlay, if supported).

If the mouse cursor is already on, it must be turned off (using *mouseCursorOn()*) before calling this function, then turned on again.

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

mouseCursorOn(), mouseCursorXY(), mouseRead(), mouseReady(), mouseRect(), mouseScale(), setGraphicsDevice(), setMouseCursor()

setMouseParam

NAME

setMouseParam - set the mouse configuration parameters

SYNOPSIS

```
void setMouseParam
    (
    int pid,                    /* parameter id */
    int value                   /* parameter value */
    )
```

DESCRIPTION

SetMouseParam() sets mouse parameter *pid* to value *value*. The mouse parameters and special defines for the bits in certain parameters are listed below. Use *mouseScale()* to set the scale parameters and *mouseRect()* to set the window parameters.

If the mouse cursor is already on, it must be turned off (using *mouseCursorOn()*) before calling this function, then turned on again.

```
/*******************************************************/
/* MOUSE MODE PARAMETERS                             */
/*******************************************************/
#define MSE_TRACKMODE  0  /* mouse position tracking mode */
#define MSE_REPORTMODE 1  /* mouse reporting mode */

/*********************************************/
/* bit fields and values of selected parameters */
/*********************************************/

/* mouse parameter 0 (tracking mode) */
/* default is (MSECSR_LOCALTRACK | MSECSR_LOCALSAVE) */
#define MSECSR_HOSTTRACK  0x00  /* cursor pos controlled by the host */
#define MSECSR_LOCALTRACK 0x01  /* cursor pos controlled by graphics hw */
#define MSECSR_PIN        0x00  /* cursor sticks at boundary */
#define NSECSR_WRAP       0x02 /* cursor wraps to other side of boundary */
#define MSECSR_SCREEN     0x00 /* cursor confined to screen boundaries */
#define MSECSR_WINDOW     0x04 /* cursor confined to window boundaries */
#define MSECSR_NOSWAPXY   0x00  /* no swap of mouse X, Y coordinates */
#define MSECSR_SWAPXY     0x08  /* swap X, Y mouse cursor movement */
#define MSECSR_HOSTSAVE   0x00  /* cursor save/res done by host */
#define MSECSR_LOCALSAVE  0x10  /* cursor save/res done by graphics hw */

/* mouse parameter 1 (report mode) */
/* these can be OR'd together - the default is MSE_SWITCHON */
#define MSE_NOEVENTS      0  /* no reports */
#define MSE_SWITCHON      1  /* report on switch closure */
#define MSE_SWITCHOFF     2  /* report on switch release */
#define MSE_SWITCHONOFF   3  /* report on switch closure or release */
#define MSE_MOVE          4  /* report all movement */
#define MSE_MOVE_SWON     8  /* report all movement while any switch closed */
```

INCLUDE FILES

drv/rgmouse.h

SEE ALSO

mouseCursorOn(), mouseCursorXY(), mouseRead(), mouseReady(), mouseRect(), mouseScale(), setMouseCursor(), setMousePage()

setOrigin

NAME

setOrigin() - sets the logical origin

SYNOPSIS

```
void setOrigin
    (
    int x,                    /* x coordinate */
    int y                     /* y coordinate */
    )
```
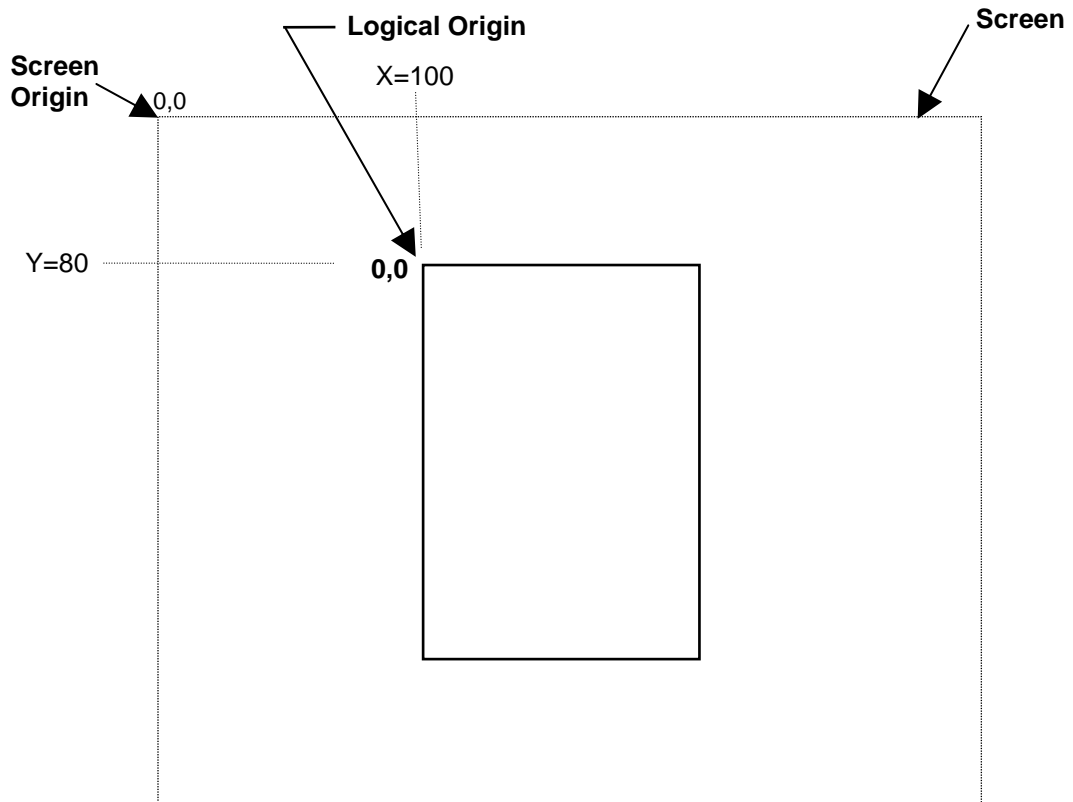
DESCRIPTION

This routine sets the logical origin.   All graphics primitives are drawn relative to the logical origin.

The default value is for the logical origin to be located at the screen origin.

A graphics object, such as a gauge, can be drawn relative to 0,0 and can subsequently be redrawn at a different screen location by simply changing the logical origin to the new location before redrawing the gauge.

INCLUDE FILES

sdl.h, extern.h

Logical Origin at x=100, y=80

setPanStart

NAME

setPanStart() - sets the display origin within the virtual window

SYNOPSIS

```
void setPanStart
    (
    int x,                        /* x coordinate */
    int y                         /* y coordinate */
    )
```

DESCRIPTION

This routine sets the origin of the displayed window, within a larger virtual window. The default value is for the display origin to be located at the upper left corner of the virtual window.
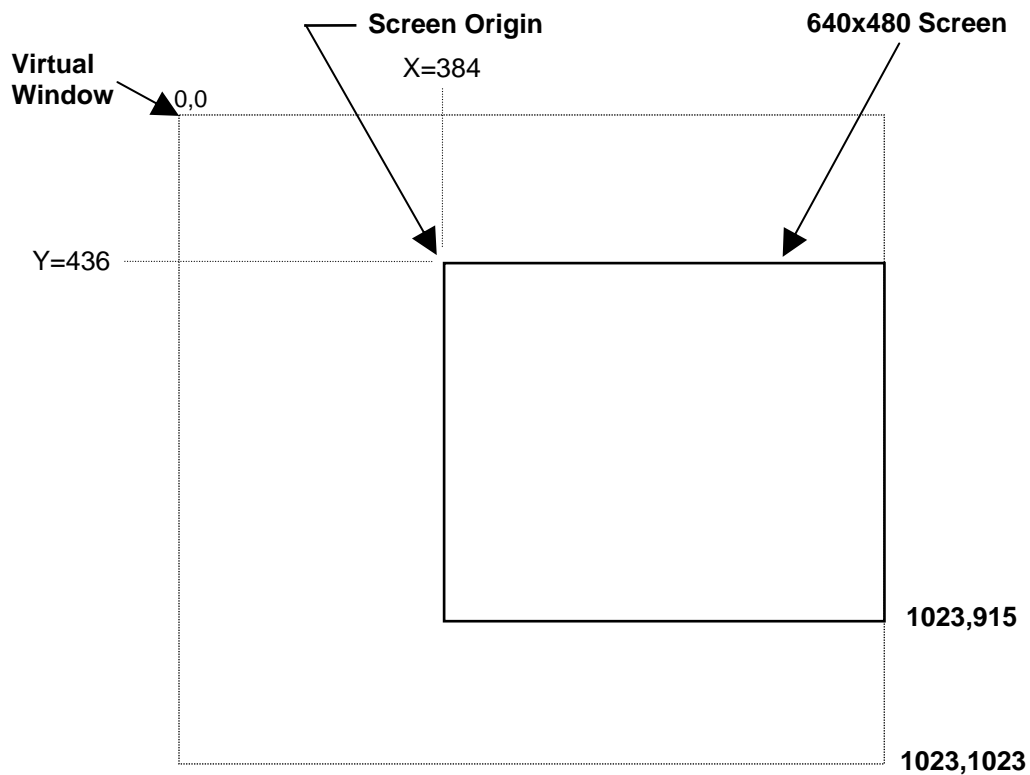
A large graphics object, or multiple graphics objects, can be drawn beyond the physical display size with *setPanStart()* called to pan the display to make different portions of the virtual window visible.

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setVirtualSize()

Pan Start at x=384, y=436

setPattern

NAME

setPattern() - specifies the current stipple fill pattern

SYNOPSIS

```
void setPattern
     (
     sPattern *newpattern  /* Pointer to stipple pattern */
     )
```

DESCRIPTION

Use this function to specify the current stipple pattern for use with both *stipple* and *opaque stipple* fills. The stipple pattern must be created and placed at the corresponding memory location by the user.

The following can be filled with a stipple pattern: filled polygons, filled circles, filled ellipses, filled arcs, and filled rectangles.

Text and lines **are also drawn** with the stipple pattern if *stipple* or *opaque stipple* is selected as the current fill style.

The stipple fill pattern is limited to 16 pixels wide maximum, but the width can be any value from 1 to 16.   The height of the pattern can be any value from 1 to 255. The stipple pattern is a binary pattern. The ones in the pattern are expanded to the foreground color. For *opaque stipple* fills, the zeros in the pattern are expanded to the background color.

All drawing primitives are affected by the current fill style which is specified with **setFillStyle().** *Use SOLID_FILL for drawing solid lines.*

The **sPattern** structure is defined as follows:

```
typedef struct tagPattern
{
  int width;
  int height;
  unsigned short *data;
}sPattern, *spPattern;
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setPatternOrigin(), setFillStyle()

setPatternOrigin

NAME

setPatternOrigin - sets stipple fill pattern origin

SYNOPSIS

```
void setPatternOrigin
      (
      int x,     /* start location in x from pattern edge */
      int y      /* start location in y from pattern edge */
      )
```

DESCRIPTION

Use this function to set the pattern origin in x and y from the pattern edge.

The default coordinates for the pattern origin is x=0, y=0. As a pattern is drawn the coordinates of the pattern continue to increment, modulo whatever the pattern dimensions (in pixels) are.

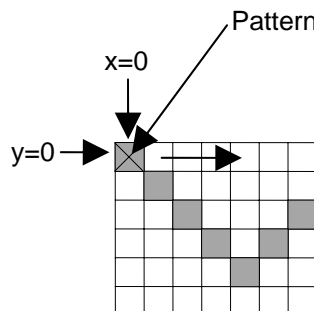Use the *setPatternOrigin()* to reset the pattern coordinates to zero, thereby causing the stipple pattern to be drawn starting at the beginning of the pattern. A stipple pattern is shown twice below, each with a different origin, indicating the starting point of the pattern.
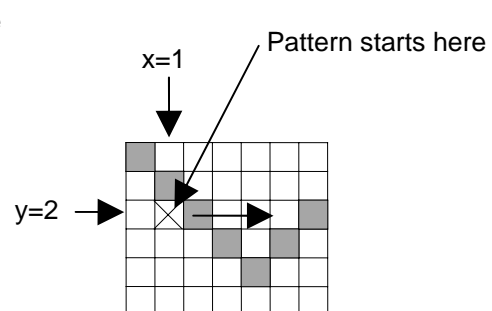
INCLUDE FILES

sdl.h, extern.h

SEE ALSO

setPattern()

Pattern Origin = 0,0

Default Coordinates

Pattern Origin = 1,2

setPixelProcessing

NAME

setPixelProcessing - sets pixel processing, *replace, and, or, xor*

SYNOPSIS

```
void setPixelProcessing
      (
      int operation                /* 0 == replace*/
      )                            /* 1 == and */
                                   /* 2 == or */
                                   /* 3 == xor */
```

DESCRIPTION

This function specifies the boolean operation for pixel processing. Pixel processing operates on the color index values, not on the color data stored in the color lookup table. If the color mode is true color, then pixel processing operates on the actual pixel values, because they are not index values. For a palette based system, the pixel color value is an index into a color lookup table.

The pixel processing types are defined in *sdl.h* as follows:

#define REPLACE 0
#define AND   1
#define OR   2
#define XOR   3

Invalid processing values are set to REPLACE.

INCLUDE FILES

sdl.h, extern.h

setTextDirection

NAME

setTextDirection - sets the current drawing direction for text

SYNOPSIS

```
void setTextDirection
    (
    int dir                  /* new direction */
    )
```

DESCRIPTION

This routine sets the current direction for drawing text on the display. Text may be drawn left to right, right to left, top to bottom, or bottom to top on a character basis. The default is left to right.

Proportionally spaced text is left justified on the bounding rectangle of the glyph even when drawing vertically. This may cause the text to be non-centered. A fixed width font is generally better for this application.

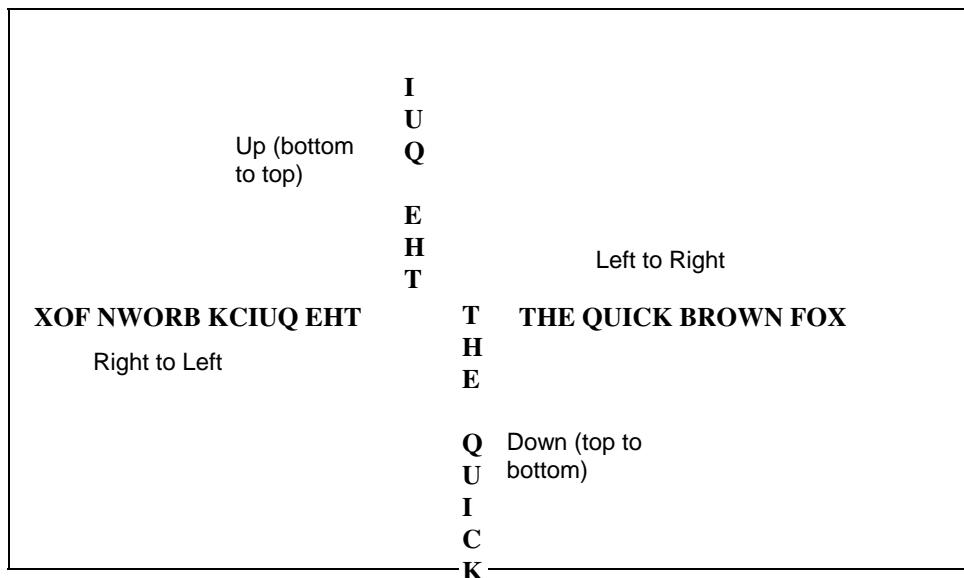The text direction is defined in *sdl.h* as:

```
#define TXT_DIR_NORM    0
#define TXT_DIR_UP      1
#define TXT_DIR_RL      2
#define TXT_DIR_DWN     3
```

INCLUDE FILES

sdl.h, extern.h

SEE ALSO

drawText()

```
                              I
                              U
            Up (bottom        Q
            to top)
                              E
                              H
                              T
                                                    Left to Right

XOF NWORB KCIUQ EHT           T        THE QUICK BROWN FOX
                              H
   Right to Left              E

                              Q      Down (top to
                              U      bottom)
                              I
                              C
                              K
```

setTiming

NAME

setTiming - sets custom video timing

SYNOPSIS

```
int setTiming
    (
    float vfreq,            /* vertical frequency (in Hz) */
    float vblank,           /* vertical blanking width */
    float vfporch,          /* vertical front porch width */
    float vsync,            /* vertical sync width */
    float hblank,           /* horizontal blanking width */
    float hfporch,          /* horizontal front porch width */
    float hsync             /* horizontal sync width */
    )
```

DESCRIPTION

This function provides means to adjust the video timing to match a particular display monitor. This function may be called at any time after a successful *initGraphics()* call.

The vertical timing values, other than the vertical frequency, are specified in milliseconds. Horizontal timing values are specified in microseconds. This function is not available with the RG-101 driver.

INCLUDE FILES

sdl.h, extern.h

RETURNS

int /* 0 on success, -1 on error */

SEE ALSO

initGraphics()

setTransparency

NAME

setTransparency - sets the text transparency mode

SYNOPSIS

```
void setTransparency
       (
       int transparency          /* 0 == transparency off */
       )                         /* 1 == transparency on  */
```

DESCRIPTION

This function turns transparency on or off.

Transparency applies only to text. If on, only the ones of the character glyph are drawn. They are drawn in the foreground color. If transparency is off, both the ones and zeros of the character glyph are drawn. The ones are drawn in the foreground color, and the zeros are drawn in the background color.

Transparency should be used for proportionally spaced fonts, because the background cell varies from character to character.   To place a background behind proportionally spaced fonts, first draw a filled rectangle with the desired color and then write the text into the rectangular area. Transparency enabled is the default mode.

The transparency modes are defined in *sdl.h* as:

#define OPAQUE   0   /* Transparency off */
#define TRANSPARENT 1   /* Transparency on */

INCLUDE FILES

sdl.h, extern.h

setVirtualSize

NAME

setVirtualSize() - sets the display origin within the virtual window

SYNOPSIS

```
void setVirtualSize
     (
     int width,                  /* window width */
     int height                  /* window height */
     )
```

DESCRIPTION

SetVirtualSize() provides a means to specify a virtual window larger then the physcial display or screen size. For instance, a 1024x1024 virtual window can be created in the framebuffer memory using a 640x480 physical display size. The entire virtual window is available for rendering graphics objects, with the physical display providing a smaller visible window into the virtual window. *SetPanStart()* is used to move the origin of the visible displayed window.

*Note: if a virtual window is desired, this function must be called **prior** to the call to **initGraphics().** Once set, the size of the virtual window can not be changed during the graphics session.*

INCLUDE FILES

sdl.h

SEE ALSO

setPanStart()

setWritePage

NAME

setWritePage - set the current write page

SYNOPSIS

```
void setWritePage
    (
    int pagenum              /* graphics memory page number */
    )
```

DESCRIPTION

SetWritePage() selects the page number (for graphics hardware with more than one page of video memory) that is to be written to. Page numbering starts at zero.

INCLUDE FILES

sdl.h

SEE ALSO

setDisplayPage()

storeColor

NAME

storeColor - writes an rgb value to the color palette

SYNOPSIS

```
void storeColor
        (
        int index,
        int red,
        int green,
        int blue
        )
```

DESCRIPTION

This function writes the three rgb values specified to the system's color palette at the index specified. Use this function to update a single color entry in the palette, or to update the entire palette with repeated calls to this function.

INCLUDE FILES

sdl.h

SEE ALSO

getColor()

syncControl

NAME

syncControl – overrides the default horizontal and vertical sync state

SYNOPSIS

```
void syncControl
      (
      int hsync,                    /* horiz sync state */
      int vsync                     /* vert sync state */
      )
```

DESCRIPTION

This routine provides a method to set the video sync signals to a fixed state. This could be used to manually implement Display Power Management (DPMS) to force the display monitor into a standby or off state. Not all drivers may implement this function or all possible sync states.

The sync state is defined in *sdl.h* as:

```
#define SYNC_NORMAL    0
#define SYNC_LOW       1
#define SYNC_HIGH      2
#define SYNC_OFF       3
#define SYNC_INVERT    4
```

VESA Display Power Management States:

DPMS State	Vsync	Hsync
Normal	SYNC_NORMAL	SYNC_NORMAL
Standby	SYNC_NORMAL	SYNC_OFF
Suspend	SYNC_OFF	SYNC_NORMAL
Off	SYNC_OFF	SYNC_OFF

INCLUDE FILES

sdl.h

SEE ALSO

Appendix A: SDL Header Files

sdl.h

```
/* ===== $Id: sdl.h,v 1.104 2005/01/03 16:09:50 billr Exp $ ===== */
/****************************************************************************/
/*                     STANDARD DRAWING LIBRARY                          */
/*                                                                       */
/*                         Rastergraf, Inc.                             */
/*      Used under license from CURTISS-WRIGHT CONTROLS, INC.           */
/*        COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.              */
/*                                                                       */
/*  This software is licensed software subject to the terms of the      */
/*  Source Code License Agreement. Refer to the file LICENSE for details. */
/****************************************************************************/
/* FILE NAME   :sdl.h                                                    */
/* DATE CREATED:6/8/95                                                   */
/* PROJECT NAME:SDL                                                      */
/* DESCRIPTION :contains global variables, prototypes and structure defs */
/* AUTHOR      :BR                                                       */
/* REVISIONS   :                                                         */
/* NOTES       :                                                         */
/* STRUCTURES  :Point                                                    */
/*              Rectangle                                                */
/*              Pattern                                                  */
/*              FontInfo                                                 */
/*              FontStruct                                               */
/*              geCharInfo                                               */
/*              RGFontInfo                                               */
/*              RGFont                                                   */
/****************************************************************************/

#ifndef SDL_H
# define SDL_H

#ifdef __cplusplus
extern "C" {
#endif

#ifndef FALSE
# define FALSE 0
# define TRUE  !FALSE
#endif

#ifndef NULL
# define NULL (void*)0
#endif

/* HANDY MACROS */
#ifndef MAX
# define MAX(_a,_b) (((_a) > (_b)) ? (_a) : (_b))
#endif
#ifndef MIN
# define MIN(_a,_b) (((_a) < (_b)) ? (_a) :(_b))
#endif
```

SDL Header Files

sdl.h, Continued

```
/* pixel processing codes */
#define REPLACE 0
#define AND       1
#define OR        2
#define XOR       3


/* FILL RULES FOR POLYGONS */
#define EVENODD 0
#define WINDING 1

/* TRANSPARENCY FLAG (used only for text) */
#define OPAQUE        0
#define TRANSPARENT 1

/* DASH LINE STYLES */
#define ONOFF_DASH  0
#define DOUBLE_DASH 1

/* FILL STYLES */
#define SOLID_FILL     0
#define STIPPLE_FILL  1
#define OPAQUE_FILL    2   /* not available on two color systems! */

/* ARC MODES */
#define SECTOR_MODE 0
#define CHORD_MODE  1

/* TEXT DRAWING DIRECTION */
#define TXT_DIR_NORM 0
#define TXT_DIR_UP   1
#define TXT_DIR_RL   2
#define TXT_DIR_DWN  3

/* VIDEO MODES - not all modes are supported by all drivers */
#define MODE_MASK    0x0ff      /* allows for 256 video modes */
#define M640X480X4       0      /*  640h x   480v x   4 bpp */
#define M640X480X8       1      /*  640h x   480v x   8 bpp */
#define M800X600X4       2      /*  800h x   600v x   4 bpp */
#define M800X600X8       3      /*  800h x   600v x   8 bpp */
#define M1024X768X4      4      /* 1024h x   768v x   4 bpp */
#define M1024X768X8      5      /* 1024h x   768v x   8 bpp */
#define EL_VIDEO         6      /* RGI EL/LCD boards only */
#define M1152X900X8      7      /* 1152h x   900v x   8 bpp */
#define M1280X1024X8     8      /* 1280h x 1024v x   8 bpp */
#define M1600X1200X8     9      /* 1600h x 1200v x   8 bpp */
#define M640X480X15      10     /*  640h x   480v x   15/16 bpp */
#define M640X480X16      10     /*  640h x   480v x   15/16 bpp */
#define M800X600X15      11     /*  800h x   600v x   15/16 bpp */
#define M800X600X16      11     /*  800h x   600v x   15/16 bpp */
#define M1024X768X15     12     /* 1024h x   768v x   15/16 bpp */
#define M1024X768X16     12     /* 1024h x   768v x   15/16 bpp */
#define M1152X900X15     13     /* 1152h x   900v x   15/16 bpp */
#define M1152X900X16     13     /* 1152h x   900v x   15/16 bpp */
#define M1280X1024X15    14     /* 1280h x 1024v x   15/16 bpp */
```

SDL Header Files

sdl.h, Continued

```
#define M1280X1024X16    14       /* 1280h x 1024v x  15/16 bpp */
#define M1600X1200X15    15       /* 1600h x 1200v x  15/16 bpp */
#define M1600X1200X16    15       /* 1600h x 1200v x  15/16 bpp */
#define M640X480X24      16       /*  640h x  480v x  24 bpp */
#define M800X600X24      17       /*  800h x  600v x  24 bpp */
#define M1024X768X24     18       /* 1024h x  768v x  24 bpp */
#define M1152X900X24     19       /* 1152h x  900v x  24 bpp */
#define M1280X1024X24    20       /* 1280h x 1024v x  24 bpp */
#define M1600X1200X24    21       /* 1600h x 1200v x  24 bpp */
#define M640X480X32      22       /*  640h x  480v x  32 bpp */
#define M800X600X32      23       /*  800h x  600v x  32 bpp */
#define M1024X768X32     24       /* 1024h x  768v x  32 bpp */
#define M1152X900X32     25       /* 1152h x  900v x  32 bpp */
#define M1280X1024X32    26       /* 1280h x 1024v x  32 bpp */
#define M1600X1200X32    27       /* 1600h x 1200v x  32 bpp */
#define MTEXT            255      /* 80 column text mode */


/* Custom Video Modes */
#define M512X256X4       128      /*  512h x  256v x  4 bpp */
#define M512X256X8       129      /*  512h x  256v x  8 bpp */
#define M512X384X4       130      /*  512h x  384v x  4 bpp */
#define M512X384X8       131      /*  512h x  384v x  8 bpp */
#define M320X240X4       132      /*  320h x  240v x  4 bpp */
#define M752X582X8       133      /*  752h x  582v x  8 bpp */
#define M768X576X8       134      /*  768h x  576v x  8 bpp */
#define MSTANAG_AX8      (135|P_SYNC_ON_GREEN)  /*  672h x  809v x  8 bpp */
#define MVISTAX8         (136|P_SYNC_ON_GREEN)  /*  672h x  672v x  8 bpp */
#define MSTANAG_AX16     (137|P_SYNC_ON_GREEN)  /*  672h x  809v x  16 bpp */
#define MVISTAX16        (138|P_SYNC_ON_GREEN)  /*  672h x  672v x  16 bpp */
#define MSTANAG_AX32     (139|P_SYNC_ON_GREEN)  /*  672h x  809v x  32 bpp */
#define MVISTAX32        (140|P_SYNC_ON_GREEN)  /*  672h x  672v x  32 bpp */
#define M1024X768X8_XGA  141           /* 1024h x  768v x  8 bpp @ 60 Hz */
#define MSTANAG_BX8      (142|P_SYNC_ON_GREEN)  /*  768h x  574v x  8 bpp */
#define MSTANAG_BX16     (143|P_SYNC_ON_GREEN)  /*  768h x  574v x 16 bpp */
#define MSTANAG_BX32     (144|P_SYNC_ON_GREEN)  /*  768h x  574v x 32 bpp */
#define MSTANAG_CX8      (145|P_SYNC_ON_GREEN)  /*  640h x  484v x  8 bpp */
#define MSTANAG_CX16     (146|P_SYNC_ON_GREEN)  /*  640h x  484v x 16 bpp */
#define MSTANAG_CX32     (147|P_SYNC_ON_GREEN)  /*  640h x  484v x 32 bpp */


/* FLAT PANEL TYPES - not all panels are supported by all drivers */
#define P_NEC_NL6448AC33_18       0x0000
#define P_SHARP_640X480X18_TFT    0x0100
#define P_VT_LCD70X_640X480X18_TFT 0x0200
#define P_GENERIC_18BIT_TFT       0x0E00
#define P_GENERIC_24BIT_TFT       0x0F00
#define P_SYNC_ON_GREEN           0x1000
#define P_COMPOSITE_SYNC          0x2000
#define P_DVI                     0x3000
#define P_SEC_DAC                 0x08000  /* enable secondary DAC */
#define P_SEC_DAC_SOG             0x10000  /* SOG on secondary RGB DAC */
#define P_BLOCK_SYNC_ON_GREEN     0x20000  /* block mode SOG on main */
#define P_SEC_BLOCK_SOG           0x40000  /* block mode SOG on secondary */
#define P_LVDS1                   0x80000  /* enable LVDS1 output */
#define P_LVDS2                   0x100000 /* enable LVDS2 output */
#define P_NO_WFIFO                0x200000 /* do not use WFIFO with tvout */
/* SOG types for Duros */
#define P_SOG_XOR                 0x001000
#define P_SOG_STANAG              0x020000
#define P_SOG_AND                 0x021000
#define PANEL_MASK                0x00f00 /* allows for 256 flat panel types */
```

SDL Header Files

sdl.h, Continued

```
/* DEVICE SPECIFIERS for setGraphicsDevice() */
/* these can be or'd together, except for the two TV modes */
#define CRT_DEVICE      0x01 /* default */
#define LCD_DEVICE      0x02
#define NTSC_TV_DEVICE 0x04
#define PAL_TV_DEVICE   0x08
#define VP2CRTDAC       0x10 /* VP engine feeds CRT DAC */
#define VP2DIGOUT       0x20 /* VP engine feeds flat panel output */
#define VP2LVDS2        0x40 /* VP engine feeds LVDS2 output */
#define NULL_DEVICE     0x80 /* keep previous output settings */

/* (GRAPHICS CHIP) CHIP/BOARD SELECT for setGraphicsDevice() */
/* one of these can be or'd with the device specifiers above */
    /* graphics chip number on a board */
#define GDEV_0  0x000000
#define GDEV_1  0x000100
#define GDEV_2  0x000200
#define GDEV_3  0x000300


/* board number in system */
#define GBRD_0  0x000000
#define GBRD_1  0x010000
#define GBRD_2  0x020000
#define GBRD_3  0x030000


/* DEVICE SPECIFIERS for setVideoDevice() and setVideoSource(dev, port) */
#define VDEV_HOST       0
#define VDEV_DECODER1   1
#define VDEV_DECODER2   2

/* (GRAPHICS CHIP) PORT SPECIFIERS for setVideoSource(dev, port) */
#define VPORT_VPORT    0 /* hardware video port */
#define VPORT_PCI      1 /* PCI bus */

/* input port(s) on video decoder chip */
#define VDEC_ANALOG    0 /* Standard analog input */
#define VDEC_DIGITAL   1 /* CCIR 656 digital video input */
#define VDEC_DVI       2 /* DVI digital video input */

/* output port(s) on video decoder chip */
#define VDEC_SPI       1 /* Streaming Pixel Interface (V-Port) */
#define VDEC_PCI       2 /* PCI bus */

/* (DECODER) VIDEO SOURCES for videoSelect() *
/* VIDEO SOURCES - for RG101 */
#define CVIDEO1   0     /* composite video 1 */
#define CVIDEO2   1     /* composite video 2 */
#define CVIDEO3   3     /* composite video 3 */
#define CVIDEO4   4     /* composite video 4 */
#define SVIDEO    2     /* S-video 1 */
#define GPIO_DIG  16    /* digital video on GPIO port */
#define RGBHV     17    /* analog RGBHsVs video */
#define TEST_PATN 64    /* internal test pattern (e.g. color bars) */
#define LOOPBACK  65    /* loopback from composite video output */
```

SDL Header Files

sdl.h, Continued

```
/* (DECODER) SOURCE VIDEO MODES for initCapture() */
#define NTSC            0
#define PAL             1
#define CCIR_NTSC       2
#define CCIR_PAL        3
#define NTSC_2_1        4
#define PAL_2_1         5
#define CCIR_NTSC_2_1   6
#define CCIR_PAL_2_1    7
#define SECAM           8
#define CCIR_656        16  /* CCIR 656 digital video */
#define SMPTE_125       17  /* Modified SMPTE-125 digital video */
#define VGA_RGB         18  /* RGBHV input to AD9882 on Stratus */
#define VGA_MONO        19  /* monochrome RGB input to AD9882 on Stratus */
#define VGA_YC          19  /* old name */
#define VGA_RGB_SOG     20  /* RGB+SOG input to AD9882 on Stratus */
#define VGA_MONO_SOG    21  /* monochrome RGB+SOG input to AD9882 */
#define VGA_DVI         22  /* DVI input to AD9882 on Stratus */
#define STANAG_A        23  /* STANAG-A input to AD9882 */
#define STANAG_B        24  /* STANAG-B input to AD9882 */
#define STANAG_C        25  /* STANAG-C input to AD9882 */
#define SONY_DXC990     26  /* Sony DXC-990 camcorder RGB+SOG to AD9882 */
#define NUM_VIDMODES    9   /* number of Bt835 capture formats */


/* Custom source video modes (or'd into base source mode above) */
#define CCIR_PAL_CCD (0x01<<8) /* 752h x 582v PAL CCIR */


/* Video capture pixel formats */
#define VID_YUV422      0   /* default YCrCb 4:2:2 packed */
#define VID_RGB16       1   /* 5-6-5 RGB */
#define VID_RGB24       2   /* 8-8-8 RGB (packed) */
#define VID_RGB32       3   /* 8-8-8-8 RGB */
#define VID_Y8          4   /* 8-bit luminance only (for monochrome) */
#define VID_RGB8        5   /* 8-bit RGB (direct or indexed) */


/* Video capture field selection */
#define CAPTURE_EVEN  1 /* capture/display odd fields */
#define CAPTURE_ODD   2 /* capture/display even fields */
#define CAPTURE_BOTH  3 /* capture/display both even and odd fields */


/* DMA source flags (support varies by board type) */
#define DMA_NONE        0
#define DMA_GDEV0_GRMEM 1 /* graphics device 0 - graphics memory */
#define DMA_GDEV0_VIMEM 2 /* graphics device 0 - video capture  memory */
#define DMA_GDEV1_GRMEM 3 /* graphics device 1 - graphics memory */
#define DMA_GDEV1_VIMEM 4 /* graphics device 1 - video capture  memory */
#define DMA_VDEC0       5 /* video decoder 0 */
#define DMA_VDEC1       6 /* video decoder 1 */
#define DMA_ADEC0       7 /* audio decoder 0 */
#define DMA_ADEC1       8 /* audio decoder 1 */
#define DMA_NSRCS       8 /* number of possible sources for DMA */


/* DMA transfer flags (support varies by board type) */
#define DMA_WAIT         0 /* polled wait for entire transfer */
#define DMA_NOWAIT       1 /* interrupt driven transfer */
#define DMA_CONTINUOUS   2 /* copy on each Vblank or ZV port interrupt */
#define DMA_INTERLACE    4 /* transfer even fields only */
#define DMA_GRAPHICS_MEM 8 /* copy graphics mem instead of video mem */
```

SDL Header Files

sdl.h, Continued

```
/* syncControl() flags */
#define SYNC_NORMAL      0
#define SYNC_LOW         1
#define SYNC_HIGH        2
#define SYNC_OFF         3

/* BOARD TYPES - for drivers that support multiple board types */
#define BOARD_NONE       0
#define BOARD_RG100      1
#define BOARD_RG101      2
#define BOARD_RG103      3
#define BOARD_RG750      4
#define BOARD_VFX        5
#define BOARD_VCQM       6
#define BOARD_VFG        7
#define BOARD_VGL        8
#define BOARD_VQP        9
#define BOARD_VFR        10
#define BOARD_ARGUS      11 /* 2x Borealis 3 + 2x Bt878a */
#define BOARD_GEMINI     12 /* 2x Borealis 3 */
#define BOARD_STRATUS    13 /* SM731 + Bt835 + AD9882 */
#define BOARD_TROPOS     14 /* SM731 */
#define BOARD_VISTA      15 /* SM731 (custom mod Tropos) */
#define BOARD_DUROS      16 /* SM731 + CY22150 */
#define BOARD_GARNET     17 /* SM731 + CY22150 + Bt835 + AD9882 */
#define BOARD_ARGUSR2    18 /* 2x Borealis 3 + 2x cx23880 + usb audio */
/* board families */
#define BOARD_FAM_RG10x   129 /* RG-100, RG-101 */
#define BOARD_FAM_VCQM    130 /* VCQ, VFG, VQP */
#define BOARD_FAM_VFX     131 /* VFX, VFR */
#define BOARD_FAM_GEMINI  132 /* GEMINI, ARGUS, ARGUSV2 */
#define BOARD_FAM_STRATUS 133 /* STRATUS, TROPOS, COLOS, DUROS */
```

SDL Header Files

sdl.h, Continued

```c
typedef int Bool;
typedef void (*PIXEL_OUTPUT)(int x,int y);

/* x, y designate the origin, usually in the upper left hand */
/* corner.                                                    */
typedef struct tagPoint
{
  short x;
  short y;
}sPoint,*spPoint;


typedef struct tagRectangle
{
  short x,
        y,
        width,  /* relative to x,y */
        height; /* relative to x,y */
}sRectangle,*spRectangle;

/* PATTERNS: patterns are 16 columns by 16 rows.
*  The width of a pattern can be from [1..16] but must always be filled
*  out as unsigned shorts with bit 0 being x0 in the pattern.
*/
typedef struct tagPattern
{
  unsigned int  width,
                height;
  unsigned short *data;  /* pointer to the pattern data */
}sPattern,*spPattern;

/* NEEDED FOR STRUCT DEFINITION BELOW */
typedef struct tagFontInfo
{
  short width,    /* width of character in pixels    */
        ascent,   /* number of pixels above baseline */
        descent;  /* number of pixels below baseline */
}sFontInfo, *spFontInfo;


/* FONT INFO STRUCTURE */
typedef struct tagFontStruct
{
  char            fontName[12]; /* name of font                 */
  unsigned long   fontId;       /* unique font identifier       */
  sFontInfo       minbounds,    /* smallest char dimensions     */
                  maxbounds;    /* largest  char dimensions     */
  unsigned short  fontAscent;   /* overall font ascent          */
  unsigned short  fontDescent;  /* overall font descent         */
}sFontStruct, *spFontStruct;
```

SDL Header Files

sdl.h, Continued

```
/*
 * Base structures used by the Raster Graphics Font format (rgf)
 * from X11R6.
 */

/* $XConsortium: fontstruct.h,v 1.16 94/04/17 20:11:08 gildea Exp $ */
/*************************************************************
Copyright 1987 by Digital Equipment Corporation, Maynard, Massachusetts.

                         All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Digital not be
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL
DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.

*************************************************************/



typedef struct taggeCharInfo
{
    short leftSideBearing,
          rightSideBearing,
          characterWidth,
          ascent,
          descent,
          attributes;
}sgeCharInfo, *spgeCharInfo;
```

SDL Header Files

sdl.h, Continued

```c
typedef struct tagRGFontInfo
{
    unsigned short firstCol;        /* range of glyphs for this font */
    unsigned short lastCol;
    unsigned short firstRow;
    unsigned short lastRow;
    unsigned short defaultCh;       /* default character index      */
    char           noOverlap;       /* no combina of glyphs overlap */
    char           terminalFont;    /* Character cell font          */
    char           constantMetrics; /* all metrics are the same     */
    char           constantWidth;   /* all char widths are the same */
    char           allExist;        /* no missing chars in range    */
    char           drawDirection;   /* left-to-right/right-to-left   */
    short          maxOverlap;      /* maximum overlap amount        */
    short          pad;             /* unused                        */
    sgeCharInfo    minbounds;       /* glyph metrics minimums        */
    sgeCharInfo    maxbounds;       /* glyph metrics maximums        */
    short          fontAscent;      /* font ascent amount            */
    short          fontDescent;     /* font descent amount           */

}sRGFontInfo,*spRGFontInfo;


typedef struct tagRGFont
{
    int            refcnt;
    int            type;            /* 0=scalable, 2=bitmap */
    char           name[12];        /* font name            */
    sRGFontInfo    info;
    unsigned long format;
 /* format is composed of (starting at the LSB [see macros above]):
  glyphpad: 2      glyph pad: 1, 2, 4 or 8
  byteorder:1      byte order: LSBFirst/MSBFirst
  bitorder: 1      bit order: LSBFirst/MSBFirst
  scanunit: 2      glyph scan unit: 1, 2 or 4
  encodingunit:2   encoding unit: 1, 2 or 4
  reserved:25
  */
    void          *fontPrivate;   /* private to font             */
                                  /* e.g. offset to sRGBitmapFont */
}sRGFont,*spRGFont;


typedef struct tagDmaInfo
{
    unsigned int *startAddr;    /* start of data to transfer (in PCI space) */
    unsigned int *dmaStart;     /* start of DMA buffer (in CPU space) */
    unsigned int *dmaPciStart;  /* start of DMA buffer (in PCI space) */
    int totalSize;              /* total size of transfer */
    int remainSize;             /* remaining transfer size */
    int dmaFlags;               /* flags for DMA operations */
    int unused01;
} sDmaInfo, *spDmaInfo;
```

SDL Header Files

sdl.h, Continued

```
/* Function prototypes */
```

```
#ifdef __cplusplus
};
#endif
#endif
```

SDL Header Files

extern.h

```
/* ===== $Id: extern.h,v 1.33 2004/02/28 20:34:38 billr Exp $ ===== */
/*************************************************************************/
/*                       STANDARD DRAWING LIBRARY                      */
/*                                                                     */
/*                          Rastergraf, Inc.                           */
/*       Used under license from CURTISS-WRIGHT CONTROLS, INC.         */
/*       COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.              */
/*                                                                     */
/*  This software is licensed software subject to the terms of the     */
/*  Source Code License Agreement. Refer to the file LICENSE for details. */
/*************************************************************************/
/* FILE NAME   : extern.h                                              */
/* DATE CREATED: 7/29/95                                               */
/* PROJECT NAME: SDL                                                   */
/* DESCRIPTION : header file for SDL global variables                  */
/* AUTHOR      : PK                                                    */
/* REVISIONS   : 7/12/97 br - add _sdl_fd for Lynx                     */
/* NOTES       :                                                       */
/* FUNCTIONS   :                                                       */
/*************************************************************************/

#ifndef EXTERN_H
# define EXTERN_H

extern int          _arcMode;
extern int          _fillStyle;
extern int          _fillRule;
extern int          _pproc;

extern unsigned char *_dashList;
extern int          _dashStyle;
extern int          _dashOffset;
extern int          _numDash;

extern int          _currentFont;
extern int          _numFonts;
extern int          _transparency;

extern sPoint       _origin;
extern sPoint       _patrnOrigin;
extern sRectangle   _clipRect;
extern sRectangle   _realClipRect;
extern spPattern    _currentPattern;

extern unsigned long _fgColor;
extern unsigned long _bgColor;

extern int          _maxX;
extern int          _maxY;
extern int          _bitsPerPixel;
extern long         _screenPitch;
extern unsigned char *_graphicsBase;
extern int          _videoMode;
extern int          _grayscale;
extern unsigned char *_paletteTable;
```

SDL Header Files

extern.h, Continued

```
extern unsigned char *_videoPages[];
extern int           _numVideoPages;
extern int           _displayPage;
extern int           _writePage;
extern int           _isOverlay;
extern int           _numDevices;
extern int           _currentDevice;
extern int           _maxVideoMemory;
extern int           _useKeyboard;
extern int           _useMouse;
extern int           _mouseType;
#ifdef __Lynx__
extern int           _sdl_fd;
#endif
extern int           _pciMemBase;
extern int           _pciIoBase;
extern int           _pciConfigAddr;
extern int           _pciConfigData;
extern int           _pciPciIoOffset;
extern int           _pciIsaIoOffset;
extern int           _pciPciMemOffset;
extern int           _pciSwap;
extern int           _pciChannel;
#if defined(_solaris_) || defined(__Lynx__)
extern char *        _devName;
#endif
extern int           _intNumber;
extern int           _ioBase;
extern int           _pciBus;
extern int           _pciDev;
extern int           _pciBus2;
extern int           _forcePciConfig;
extern int           _boardType;
extern int           _virtualWidth;
extern int           _virtualHeight;
extern int           _numBoards;
extern int           _currentBoard;
extern int           _numVideoDevs;
extern int           _videoInPort;
extern int           _videoInDev;
extern int           _cxOffset;
extern int           _cyOffset;

extern void (*fatLineFunc)(int x1, int y1, int x2, int y2);
extern void (*fatDashLineFunc)(spPoint pt1, spPoint pt2,
            int tindex, int toffset, RGBool swapped);
extern int (*sdl_numPciBussesFunc)(void);
extern int (*sdl_pciConfigInLongFunc)(int busNo, int deviceNo,
            int funcNo, int address, unsigned int *pData);
extern int (*sdl_pciConfigOutLongFunc)(int busNo, int deviceNo,
            int funcNo, int address, unsigned int data);


#endif
```

SDL Header Files

sdltimer.h

```
/* ===== $Id: sdltimer.h,v 1.17 2003/08/30 23:53:08 billr Exp $ ===== */
/**********************************************************************/
/*                     STANDARD DRAWING LIBRARY                     */
/*                                                                  */
/*                         Rastergraf, Inc.                         */
/*      Used under license from CURTISS-WRIGHT CONTROLS, INC.       */
/*      COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.            */
/*                                                                  */
/*  This software is licensed software subject to the terms of the  */
/*  Source Code License Agreement. Refer to the file LICENSE for details. */
/**********************************************************************/
/* FILE NAME   : sdltimer.h                                         */
/* DATE CREATED: 8/26/95                                            */
/* PROJECT NAME: SGL                                                */
/* DESCRIPTION : defines for time delays                            */
/* AUTHOR      : BR                                                 */
/* FUNCTIONS   :                                                    */
/**********************************************************************/
#ifndef SDLTIMER_H
# define SDLTIMER_H

#ifdef __cplusplus
extern "C" {
#endif
/*
 * This file defines macros for delays in increments of clock ticks
 * (typically 1/60th of a second) and in increments of seconds.
 * One and only one of these OS types must be defined in the Makefile
 * or make.include file.
 */
#ifdef VXWORKS
#include <taskLib.h>
#define TICK_DELAY(_n)  taskDelay(_n)
#define SLEEP(_n)       taskDelay(60*(_n))
#endif

#ifdef OS9_OS
#define TICK_DELAY(_n)  tsleep(_n)
#define SLEEP(_n)       sleep(_n)
#endif

#ifdef _OS9000
/* close-enough conversion from 1/60 second to 1/256 second */
#define TICK_DELAY(_n) {int t; unsigned long int s;\
    t = ((_n)*5)|0x80000000;\
    _os_sleep(&t, &s);\
    }
#define SLEEP(_n){int t; unsigned long int s;\
    t = ((_n)*256)|0x80000000;\
    _os_sleep(&t, &s);\
    }
#endif
```

SDL Header Files

sdltimer.h, Continued

```
#ifdef VMEX
#include <unistd.h>
#include "vmex/vmexflgs.h"
#define TICK_DELAY(_n)  tm_wkafter(_n)
#define SLEEP(_n)       sleep(_n)
#endif

#ifdef AIX
#include <X11/Xlib.h>
#include <unistd.h>
#include <sys/select.h>
#define TICK_DELAY(_n)  {struct timeval delay;\
                    extern Display *dpy;\
                    XSync(dpy,0);\
                    delay.tv_sec = (_n)/1000000;\
                    delay.tv_usec = (((_n)*1000000)/60)%1000000;\
                    select(0, 0, 0, 0, &delay);\
                  }
#define SLEEP(_n)  {extern Display *dpy;\
                     XSync(dpy,0);\
                     sleep(_n);\
                    }
#endif

#ifdef UNIX  /* generic X11/Unix */
#ifdef SVGA
#include <unistd.h>
#include <sys/time.h>
#define TICK_DELAY(_n) {struct timeval delay;\
     delay.tv_sec = (_n)/1000000;\
     delay.tv_usec = (((_n)*1000000)/60)%1000000;\
     select(0, (fd_set*)0, (fd_set*)0, (fd_set*)0, &delay);\
     }
#define SLEEP(_n) sleep(_n)
# else
#include <unistd.h>
#include <sys/time.h>
#define TICK_DELAY(_n) {struct timeval delay;\
     sync__peritek_com();\
     delay.tv_sec = (_n)/1000000;\
     delay.tv_usec = (((_n)*1000000)/60)%1000000;\
     select(0, (fd_set*)0, (fd_set*)0, (fd_set*)0, &delay);\
     }
#define SLEEP(_n){sync__peritek_com();\
     sleep(_n);\
     }
# endif
#endif
```

SDL Header Files

sdltimer.h, Continued

```
#ifdef __Lynx__
#include <unistd.h>
#include <sys/time.h>
#define TICK_DELAY(_n) {struct timeval delay;\
    delay.tv_sec = (_n)/1000000;\
    delay.tv_usec = (((_n)*1000000)/60)%1000000;\
    select(0, (fd_set*)0, (fd_set*)0, (fd_set*)0, &delay);\
    }
#define SLEEP(_n)sleep(_n)
#endif

#ifdef __cplusplus
};
#endif

#endif
```

SDL Header Files

colors.h

```
/* ===== $Id: colors.h,v 1.12 1997/08/02 23:25:15 billr Exp $ ===== */
/***************************************************************************/
/*                    STANDARD DRAWING LIBRARY                         */
/*                                                                     */
/*                       Rastergraf, Inc.                              */
/*      Used under license from CURTISS-WRIGHT CONTROLS, INC.          */
/*      COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.               */
/*                                                                     */
/*  This software is licensed software subject to the terms of the     */
/*  Source Code License Agreement. Refer to the file LICENSE for details. */
/***************************************************************************/
/* FILE NAME    :colors.h                                              */
/* DATE CREATED:6/30/95                                                */
/* PROJECT NAME:SDL                                                    */
/* DESCRIPTION :Color names for SDL colors for palette RGI_1           */
/* AUTHOR       :Herbie                                                */
/* REVISIONS   12/25/95 br – add RG-752 color mapping                  */
/***************************************************************************/
/* This file defines color names and their corresponding index values */
/* allowing SDL colors to be referenced by name instead of by number. */

#ifndef COLORS_H
# define COLORS_H

#ifdef RG752
/*
 * The RG-752 is a direct color board (i.e no lookup palette) with
 * 3 bits of color information for each gun, mapped into a 16-bit
 * word: o000rrr0ggg0bbb0. The leading 'o' bit, when set, makes the
 * color opaque when used in the overlay plane.
 * The defines below are an attempt to map the palettized colors
 * used in 8-bit systems into the RG-752 color space. This provides
 * a common color name space for demo programs designed to run on
 * different hardware/boards.
 */
/* Color Name           Color Value  */
/* CGA Colors */
#define XBrown               0x8026
#define Black           0x8000          /*  0 */
#define Blue            0x8006          /*  1 */
#define Green           0x8060          /*  2 */
#define Cyan            0x8066          /*  3 */
#define Red             0x8600          /*  4 */
#define Magenta         0x8606          /*  5 */
#define Brown           0x8660          /*  6 */
#define LightGray       0x8888          /*  7 */
#define DarkGray        0x8444          /*  8 */
#define LightBlue       0x800e          /*  9 */
#define LightGreen      0x80e0          /* 10 */
#define LightCyan       0x80ee          /* 11 */
#define LightRed        0x8e00          /* 12 */
#define LightMagenta    0x8e0e          /* 13 */
#define Yellow          0x8ee0          /* 14 */
#define White           0x8eee          /* 15 */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Color Value  */
/* 8 Shades of Gray */
#define Gray0           0x8000          /*  16 */       /* Darkest Shade */
#define Gray1           0x8000          /*  17 */
#define Gray2           0x8000          /*  18 */
#define Gray3           0x8222          /*  19 */
#define Gray4           0x8222          /*  20 */
#define Gray5           0x8222          /*  21 */
#define Gray6           0x8444          /*  22 */
#define Gray7           0x8444          /*  23 */
#define Gray8           0x8666          /*  24 */
#define Gray9           0x8666          /*  25 */
#define Gray10          0x8666          /*  26 */
#define Gray11          0x8888          /*  27 */
#define Gray12          0x8888          /*  28 */
#define Gray13          0x8aaa          /*  29 */
#define Gray14          0x8ccc          /*  30 */
#define Gray15          0x8eee          /*  31 */       /* Lightest Shade */
/* Eight shades of various colors */
#define DarkRed0        0x8600          /*  32 */       /* Darkest Shade */
#define DarkRed1        0x8600          /*  33 */
#define DarkRed2        0x8600          /*  34 */
#define DarkRed3        0x8600          /*  35 */
#define DarkRed4        0x8600          /*  36 */
#define DarkRed5        0x8800          /*  37 */
#define DarkRed6        0x8800          /*  38 */
#define DarkRed7        0x8800          /*  39 */       /* Lightest Shade */

#define DarkBrown0      0x8400          /*  40 */       /* Darkest Shade*/
#define DarkBrown1      0x8420          /*  41 */
#define DarkBrown2      0x8620          /*  42 */
#define DarkBrown3      0x8620          /*  43 */
#define DarkBrown4      0x8620          /*  44 */
#define DarkBrown5      0x8620          /*  45 */
#define DarkBrown6      0x8620          /*  46 */
#define DarkBrown7      0x8820          /*  47 */       /* Lightest Shade */

#define Red0            0x8800          /*  48 */       /* Darkest Shade */
#define Red1            0x8a00          /*  49 */
#define Red2            0x8a00          /*  50 */
#define Red3            0x8c00          /*  51 */
#define Red4            0x8c00          /*  52 */
#define Red5            0x8c00          /*  53 */
#define Red6            0x8c00          /*  54 */
#define Red7            0x8e00          /*  55 */       /* Lightest Shade */

#define Brown0          0x8620          /*  56 */       /* Darkest Shade*/
#define Brown1          0x8820          /*  57 */
#define Brown2          0x8820          /*  58 */
#define Brown3          0x8840          /*  59 */
#define Brown4          0x8840          /*  60 */
#define Brown5          0x8a40          /*  61 */
#define Brown6          0x8a40          /*  62 */
#define Brown7          0x8c40          /*  63 */       /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Color Value   */
#define RedBrown0       0x8a20         /*  64 */        /* Darkest Shade */
#define RedBrown1       0x8a20         /*  65 */
#define RedBrown2       0x8a20         /*  66 */
#define RedBrown3       0x8a20         /*  67 */
#define RedBrown4       0x8a40         /*  68 */
#define RedBrown5       0x8a40         /*  69 */
#define RedBrown6       0x8a40         /*  70 */
#define RedBrown7       0x8a42         /*  71 */        /* Lightest Shade */

#define LightBrown0     0x8a60         /*  72 */        /* Darkest Shade */
#define LightBrown1     0x8a60         /*  73 */
#define LightBrown2     0x8a60         /*  74 */
#define LightBrown3     0x8a60         /*  75 */
#define LightBrown4     0x8c60         /*  76 */
#define LightBrown5     0x8c80         /*  77 */
#define LightBrown6     0x8c80         /*  78 */
#define LightBrown7     0x8c80         /*  79 */        /* Lightest Shade */

#define Orange0         0x8e00         /*  80 */        /* Darkest Shade*/
#define Orange1         0x8e20         /*  81 */
#define Orange2         0x8e20         /*  82 */
#define Orange3         0x8e40         /*  83 */
#define Orange4         0x8e40         /*  84 */
#define Orange5         0x8e40         /*  85 */
#define Orange6         0x8e60         /*  86 */
#define Orange7         0x8e60         /*  87 */        /* Lightest Shade */

#define Tan0            0x8e80         /*  88 */        /* Darkest Shade */
#define Tan1            0x8e80         /*  89 */
#define Tan2            0x8e80         /*  90 */
#define Tan3            0x8ea0         /*  91 */
#define Tan4            0x8ea0         /*  92 */
#define Tan5            0x8ea0         /*  93 */
#define Tan6            0x8ea0         /*  94 */
#define Tan7            0x8ec2         /*  95 */        /* Lightest Shade */

#define LightOrange0    0x8c42         /*  96 */        /* Darkest Shade*/
#define LightOrange1    0x8c42         /*  97 */
#define LightOrange2    0x8c42         /*  98 */
#define LightOrange3    0x8c42         /*  99 */
#define LightOrange4    0x8c42         /* 100 */
#define LightOrange5    0x8c40         /* 101 */
#define LightOrange6    0x8c40         /* 102 */
#define LightOrange7    0x8c40         /* 103 */        /* Lightest Shade */

#define Yellow0         0x8ec2         /* 104 */        /* Darkest Shade */
#define Yellow1         0x8ec2         /* 105 */
#define Yellow2         0x8ec2         /* 106 */
#define Yellow3         0x8ec0         /* 107 */
#define Yellow4         0x8ee0         /* 108 */
#define Yellow5         0x8ee0         /* 109 */
#define Yellow6         0x8ee0         /* 110 */
#define Yellow7         0x8ee0         /* 111 */        /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Color Value   */
#define Pink0           0x8e00         /* 112 */        /* Darkest Shade */
#define Pink1           0x8e22         /* 113 */
#define Pink2           0x8e22         /* 114 */
#define Pink3           0x8e42         /* 115 */
#define Pink4           0x8e42         /* 116 */
#define Pink5           0x8e44         /* 117 */
#define Pink6           0x8e64         /* 118 */
#define Pink7           0x8e64         /* 119 */        /* Lightest Shade */

#define YellowGreen0    0x8ae0         /* 120 */        /* Darkest Shade*/
#define YellowGreen1    0x8ae0         /* 121 */
#define YellowGreen2    0x8ae2         /* 122 */
#define YellowGreen3    0x8ae2         /* 123 */
#define YellowGreen4    0x8ae4         /* 124 */
#define YellowGreen5    0x8ae6         /* 125 */
#define YellowGreen6    0x8ae6         /* 126 */
#define YellowGreen7    0x8ae8         /* 127 */        /* Lightest Shade */

#define Raspberry0      0x8a02         /* 128 */        /* Darkest Shade */
#define Raspberry1      0x8a04         /* 129 */
#define Raspberry2      0x8a04         /* 130 */
#define Raspberry3      0x8c04         /* 131 */
#define Raspberry4      0x8c06         /* 132 */
#define Raspberry5      0x8c06         /* 133 */
#define Raspberry6      0x8e06         /* 134 */
#define Raspberry7      0x8e06         /* 135 */        /* Lightest Shade */

#define LightGreen0     0x88e0         /* 136 */        /* Darkest Shade*/
#define LightGreen1     0x88e0         /* 137 */
#define LightGreen2     0x88e2         /* 138 */
#define LightGreen3     0x88e2         /* 139 */
#define LightGreen4     0x88e4         /* 140 */
#define LightGreen5     0x88e6         /* 141 */
#define LightGreen6     0x88e8         /* 142 */
#define LightGreen7     0x88e6         /* 143 */        /* Lightest Shade */

#define Purple0         0x8e0a         /* 144 */        /* Darkest Shade */
#define Purple1         0x8e0a         /* 145 */
#define Purple2         0x8e0c         /* 146 */
#define Purple3         0x8e2c         /* 147 */
#define Purple4         0x8e2c         /* 148 */
#define Purple5         0x8e2c         /* 149 */
#define Purple6         0x8e4c         /* 150 */
#define Purple7         0x8e6c         /* 151 */        /* Lightest Shade */

#define Green0          0x80c0         /* 152 */        /* Darkest Shade */
#define Green1          0x80c0         /* 153 */
#define Green2          0x82c2         /* 154 */
#define Green3          0x82c2         /* 155 */
#define Green4          0x84c4         /* 156 */
#define Green5          0x86c6         /* 157 */
#define Green6          0x86c6         /* 158 */
#define Green7          0x88c8         /* 159 */        /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Color Value   */
#define DarkPurple0     0x8004          /* 160 */        /* Darkest Shade*/
#define DarkPurple1     0x8006          /* 161 */
#define DarkPurple2     0x8208          /* 162 */
#define DarkPurple3     0x8408          /* 163 */
#define DarkPurple4     0x840a          /* 164 */
#define DarkPurple5     0x860a          /* 165 */
#define DarkPurple6     0x860a          /* 166 */
#define DarkPurple7     0x880c          /* 167 */        /* Lightest Shade */

#define DarkGreen0      0x8060          /* 168 */        /* Darkest Shade */
#define DarkGreen1      0x8080          /* 169 */
#define DarkGreen2      0x8080          /* 170 */
#define DarkGreen3      0x80a0          /* 171 */
#define DarkGreen4      0x80a0          /* 172 */
#define DarkGreen5      0x80c0          /* 173 */
#define DarkGreen6      0x82c0          /* 174 */
#define DarkGreen7      0x84c0          /* 175 */        /* Lightest Shade */

#define Slate0          0x8662          /* 176 */        /* Darkest Shade*/
#define Slate1          0x8664          /* 177 */
#define Slate2          0x8666          /* 178 */
#define Slate3          0x8666          /* 179 */
#define Slate4          0x8668          /* 180 */
#define Slate5          0x866a          /* 181 */
#define Slate6          0x866a          /* 182 */
#define Slate7          0x866c          /* 183 */        /* Lightest Shade */

#define ForestGreen0    0x8040          /* 184 */        /* Darkest Shade */
#define ForestGreen1    0x8040          /* 185 */
#define ForestGreen2    0x8040          /* 186 */
#define ForestGreen3    0x8040          /* 187 */
#define ForestGreen4    0x8240          /* 188 */
#define ForestGreen5    0x8240          /* 189 */
#define ForestGreen6    0x8240          /* 190 */
#define ForestGreen7    0x8442          /* 191 */        /* Lightest Shade */

#define Blue0           0x800e          /* 192 */        /* Darkest Shade */
#define Blue1           0x800e          /* 193 */
#define Blue2           0x802e          /* 194 */
#define Blue3           0x802e          /* 195 */
#define Blue4           0x804e          /* 196 */
#define Blue5           0x824e          /* 197 */
#define Blue6           0x824e          /* 198 */
#define Blue7           0x844e          /* 199 */        /* Lightest Shade */

#define Mustard0        0x8660          /* 200 */        /* Darkest Shade*/
#define Mustard1        0x8660          /* 201 */
#define Mustard2        0x8680          /* 202 */
#define Mustard3        0x8880          /* 203 */
#define Mustard4        0x88a0          /* 204 */
#define Mustard5        0x88a0          /* 205 */
#define Mustard6        0x88a0          /* 206 */
#define Mustard7        0x8ac0          /* 207 */        /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Color Value   */
#define LightBlue0       0x806e         /* 208 */          /* Darkest Shade */
#define LightBlue1       0x808e         /* 209 */
#define LightBlue2       0x80ae         /* 210 */
#define LightBlue3       0x80ae         /* 211 */
#define LightBlue4       0x82ce         /* 212 */
#define LightBlue5       0x84ee         /* 213 */
#define LightBlue6       0x86ee         /* 214 */
#define LightBlue7       0x8aee         /* 215 */          /* Lightest Shade */

#define DesertGreen0     0x8682         /* 216 */          /* Darkest Shade*/
#define DesertGreen1     0x8882         /* 217 */
#define DesertGreen2     0x8882         /* 218 */
#define DesertGreen3     0x8882         /* 219 */
#define DesertGreen4     0x8882         /* 220 */
#define DesertGreen5     0x88a2         /* 221 */
#define DesertGreen6     0x88a4         /* 222 */
#define DesertGreen7     0x8aa4         /* 223 */          /* Lightest Shade */

#define Turquoise0       0x80a8         /* 224 */          /* Darkest Shade */
#define Turquoise1       0x80a8         /* 225 */
#define Turquoise2       0x82c8         /* 226 */
#define Turquoise3       0x82e8         /* 227 */
#define Turquoise4       0x84e8         /* 228 */
#define Turquoise5       0x86e8         /* 229 */
#define Turquoise6       0x86e8         /* 230 */
#define Turquoise7       0x88e8         /* 231 */          /* Lightest Shade */

#define DarkTurquoise0   0x8042         /* 232 */          /* Darkest Shade */
#define DarkTurquoise1   0x8062         /* 233 */
#define DarkTurquoise2   0x8062         /* 234 */
#define DarkTurquoise3   0x8082         /* 235 */
#define DarkTurquoise4   0x80a2         /* 236 */
#define DarkTurquoise5   0x80a2         /* 237 */
#define DarkTurquoise6   0x80c2         /* 238 */
#define DarkTurquoise7   0x80e2         /* 239 */          /* Lightest Shade */


#else
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

/* CGA Colors */
#define Black            0
#define Blue             1
#define Green            2
#define Cyan             3
#define Red              4
#define Magenta          5
#define Brown            6
#define Gray             7
#define DarkGray         8
#define LightBlue        9
#define LightGreen      10
#define LightCyan       11
#define LightRed        12
#define BrightMagenta   13
#define Yellow          14
#define White           15

/*Shades of Gray */
#define Gray0           16     /* Darkest Shade */
#define Gray1           17
#define Gray2           18
#define Gray3           19
#define Gray4           20
#define Gray5           21
#define Gray6           22
#define Gray7           23
#define Gray8           24
#define Gray9           25
#define Gray10          26
#define Gray11          27
#define Gray12          28
#define Gray13          29
#define Gray14          30
#define Gray15          31     /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define DarkRed0            32      /* Darkest Shade */
#define DarkRed1            33
#define DarkRed2            34
#define DarkRed3            35
#define DarkRed4            36
#define DarkRed5            37
#define DarkRed6            38
#define DarkRed7            39      /* Lightest Shade */


#define DarkBrown0          40      /* Darkest Shade */
#define DarkBrown1          41
#define DarkBrown2          42
#define DarkBrown3          43
#define DarkBrown4          44
#define DarkBrown5          45
#define DarkBrown6          46
#define DarkBrown7          47      /* Lightest Shade */


#define Red0                48      /* Darkest Shade */
#define Red1                49
#define Red2                50
#define Red3                51
#define Red4                52
#define Red5                53
#define Red6                54
#define Red7                55      /* Lightest Shade */


#define Brown0              56      /* Darkest Shade */
#define Brown1              57
#define Brown2              58
#define Brown3              59
#define Brown4              60
#define Brown5              61
#define Brown6              62
#define Brown7              63      /* Lightest Shade */


#define RedBrown0           64      /* Darkest Shade */
#define RedBrown1           65
#define RedBrown2           66
#define RedBrown3           67
#define RedBrown4           68
#define RedBrown5           69
#define RedBrown6           70
#define RedBrown7           71      /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define LightBrown0          72      /* Darkest Shade */
#define LightBrown1          73
#define LightBrown2          74
#define LightBrown3          75
#define LightBrown4          76
#define LightBrown5          77
#define LightBrown6          78
#define LightBrown7          79      /* Lightest Shade */


#define Orange0              80      /* Darkest Shade */
#define Orange1              81
#define Orange2              82
#define Orange3              83
#define Orange4              84
#define Orange5              85
#define Orange6              86
#define Orange7              87      /* Lightest Shade */


#define Tan0                 88      /* Darkest Shade */
#define Tan1                 89
#define Tan2                 90
#define Tan3                 91
#define Tan4                 92
#define Tan5                 93
#define Tan6                 94
#define Tan7                 95      /* Lightest Shade */


#define LightOrange0         96      /* Darkest Shade */
#define LightOrange1         97
#define LightOrange2         98
#define LightOrange3         99
#define LightOrange4        100
#define LightOrange5        101
#define LightOrange6        102
#define LightOrange7        103      /* Lightest Shade */


#define Yellow0             104      /* Darkest Shade */
#define Yellow1             105
#define Yellow2             106
#define Yellow3             107
#define Yellow4             108
#define Yellow5             109
#define Yellow6             110
#define Yellow7             111      /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define Pink0                 112     /* Darkest Shade */
#define Pink1                 113
#define Pink2                 114
#define Pink3                 115
#define Pink4                 116
#define Pink5                 117
#define Pink6                 118
#define Pink7                 119     /* Lightest Shade */


#define YellowGreen0          120     /* Darkest Shade */
#define YellowGreen1          121
#define YellowGreen2          122
#define YellowGreen3          123
#define YellowGreen4          124
#define YellowGreen5          125
#define YellowGreen6          126
#define YellowGreen7          127     /* Lightest Shade */


#define Raspberry0            128     /* Darkest Shade */
#define Raspberry1            129
#define Raspberry2            130
#define Raspberry3            131
#define Raspberry4            132
#define Raspberry5            133
#define Raspberry6            134
#define Raspberry7            135     /* Lightest Shade */


#define LightGreen0           136     /* Darkest Shade */
#define LightGreen1           137
#define LightGreen2           138
#define LightGreen3           139
#define LightGreen4           140
#define LightGreen5           141
#define LightGreen6           142
#define LightGreen7           143     /* Lightest Shade */


#define Purple0               144     /* Darkest Shade */
#define Purple1               145
#define Purple2               146
#define Purple3               147
#define Purple4               148
#define Purple5               149
#define Purple6               150
#define Purple7               151     /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define Green0              152      /* Darkest Shade */
#define Green1              153
#define Green2              154
#define Green3              155
#define Green4              156
#define Green5              157
#define Green6              158
#define Green7              159      /* Lightest Shade */


#define DarkPurple0         160      /* Darkest Shade */
#define DarkPurple1         161
#define DarkPurple2         162
#define DarkPurple3         163
#define DarkPurple4         164
#define DarkPurple5         165
#define DarkPurple6         166
#define DarkPurple7         167      /* Lightest Shade */


#define DarkGreen0          168      /* Darkest Shade */
#define DarkGreen1          169
#define DarkGreen2          170
#define DarkGreen3          171
#define DarkGreen4          172
#define DarkGreen5          173
#define DarkGreen6          174
#define DarkGreen7          175       /* Lightest Shade */


#define Slate0              176       /* Darkest Shade */
#define Slate1              177
#define Slate2              178
#define Slate3              179
#define Slate4              180
#define Slate5              181
#define Slate6              182
#define Slate7              183      /* Lightest Shade */


#define ForestGreen0        184      /* Darkest Shade */
#define ForestGreen1        185
#define ForestGreen2        186
#define ForestGreen3        187
#define ForestGreen4        188
#define ForestGreen5        189
#define ForestGreen6        190
#define ForestGreen7        191       /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define Blue0              192     /* Darkest Shade */
#define Blue1              193
#define Blue2              194
#define Blue3              195
#define Blue4              196
#define Blue5              197
#define Blue6              198
#define Blue7              199     /* Lightest Shade */


#define Mustard0           200     /* Darkest Shade */
#define Mustard1           201
#define Mustard2           202
#define Mustard3           203
#define Mustard4           204
#define Mustard5           205
#define Mustard6           206
#define Mustard7           207     /* Lightest Shade */


#define LightBlue0         208     /* Darkest Shade */
#define LightBlue1         209
#define LightBlue2         210
#define LightBlue3         211
#define LightBlue4         212
#define LightBlue5         213
#define LightBlue6         214
#define LightBlue7         215     /* Lightest Shade */


#define DesertGreen0       216     /* Darkest Shade */
#define DesertGreen1       217
#define DesertGreen2       218
#define DesertGreen3       219
#define DesertGreen4       220
#define DesertGreen5       221
#define DesertGreen6       222
#define DesertGreen7       223     /* Lightest Shade */


#define Turquoise0         224     /* Darkest Shade */
#define Turquoise1         225
#define Turquoise2         226
#define Turquoise3         227
#define Turquoise4         228
#define Turquoise5         229
#define Turquoise6         230
#define Turquoise7         231     /* Lightest Shade */
```

SDL Header Files

colors.h, Continued

```
/* Color Name          Index Value  */

#define DarkTurquoise0      232      /* Darkest Shade */
#define DarkTurquoise1      233
#define DarkTurquoise2      234
#define DarkTurquoise3      235
#define DarkTurquoise4      236
#define DarkTurquoise5      237
#define DarkTurquoise6      238
#define DarkTurquoise7      239      /* Lightest Shade */


#define UserDefined0        240
#define UserDefined1        241
#define UserDefined1        242
#define UserDefined3        243
#define UserDefined4        244
#define UserDefined5        245
#define UserDefined6        246
#define UserDefined7        247


#define UserDefined8        248
#define UserDefined9        249
#define UserDefined10       250
#define UserDefined11       251

#endif

/* colors reserved for the palette editor */
#define ReservedRed         252      /* Used by palette editor */
#define ReservedGreen       253      /* Used by palette editor */
#define ReservedBlue        254      /* Used by palette editor */
#define ReservedWhite       255      /* Used by palette editor */

#endif
```

Appendix B: SDL Fonts

The following fonts are included with the *Standard Drawing Library*.

```
/* ===== $Id: fonts.h,v 1.9 1997/08/02 23:25:15 billr Exp $ ===== */
/***************************************************************************/
/*                        STANDARD DRAWING LIBRARY                        */
/*                                                                        */
/*                          Rastergraf, Inc.                              */
/*       Used under license from CURTISS-WRIGHT CONTROLS, INC.            */
/*       COPYRIGHT (C) 2001 CURTISS-WRIGHT CONTROLS, INC.                 */
/*                                                                        */
/*  This software is licensed software subject to the terms of the        */
/*  Source Code License Agreement. Refer to the file LICENSE for details. */
/***************************************************************************/
/* FILE NAME   : fonts.h                                                  */
/* DATE CREATED: 8/7/95                                                   */
/* PROJECT NAME: SDL                                                      */
/* DESCRIPTION : Define macros for 16 basic fonts for SGL                 */
/* AUTHOR      : BR                                                       */
/* REVISIONS   :                                                          */
/* NOTES       :                                                          */
/* FUNCTIONS   :                                                          */
/***************************************************************************/

#ifndef FONTS_H
#define FONTS_H

#define HELVR12       0      /* Helvetica 12pt Normal Prop. Spaced */
#define HELVR08       1      /* Helvetica  8pt Normal Prop. Spaced */
#define HELVR10       2      /* Helvetica 10pt Normal Prop. Spaced */
#define HELVR14       3      /* Helvetica 14pt Normal Prop. Spaced */
#define HELVR18       4      /* Helvetica 18pt Normal Prop. Spaced */
#define HELVR24       5      /* Helvetica 24pt Normal Prop. Spaced */
#define CLR6X6        6      /* Clear 6x6 Normal Fixed Width */
#define CLR8X8        7      /* Clear 8x8 Normal Fixed Width */
#define CLR8X16       8      /* Clear 8x16 Normal Fixed Width */
#define FIX12X24RK    9      /* 12x24 Normal Fixed Width Roman-Kana */
#define HELVB14       10     /* Helvetica 14pt Bold Prop. Spaced */
#define HELVB18       11     /* Helvetica 18pt Bold Prop. Spaced */
#define HELVB24       12     /* Helvetica 24pt Bold Prop. Spaced */
#define HELVBO14      13     /* Helvetica 14pt BoldOblique Prop. Spaced */
#define HELVBO18      14     /* Helvetica 18pt BoldOblique Prop. Spaced */
#define HELVBO24      15     /* Helvetica 24pt BoldOblique Prop. Spaced */
#define RGBOLD36      16     /* RGI Bold Fixed 36x78 */
#define RGSWISS44     17     /* RGI Swiss Fixed 44x70 */
#define CURSOR1       18     /* RGI Cursor Font #1 (16x16) */
#define CURSOR2       19     /* RGI Cursor Font #2 (32x32) */

#endif
```

Fonts Included with SDL

The following are samples of the 16 fonts supplied with SDL and the number of bytes of memory required for the font.   The complete character sets for these fonts are shown on the following pages.

Font Style	Size
helvR12	6216
helvR08	5600
helvR10	5912
helvR14	6816
helvR18	8372
helvR24	11124
c1R6x6	1124
c1R8x8	1380
c1R8x16	2404
fix12x24rk	8880
helvB14	6988
helvB18	8924
helvB24	11392
helvBO14	7540
helvBO18	9232
helvBO24	12096
rgbold36	37924
rgswiss44	40804

ACKNOWLEDGEMENTS:

Appendix C: Function Code Size and Dependency Chart

Some of the SDL functions use common code with different entry points. This means that the scaleable capabilities of SDL are, in some cases, limited to groups of functions. The chart below shows the SDL functions that must be used as a group. The x's in the column below indicate the functions that must be used as a group. Each column represents a separate group of functions. The Code and RAM requirements in bytes are for the processor listed.

See Appendix B for SDL font sizes.

Code Size for 68040 Processor

Function	Groups of Functions										Code	RAM
arc()											3076	0
circle()											416	0
clearScreen()										x	52	0
closeGraphics()										x		0
dashedLine()		x									60	0
dashedPolyline()	x	x	x								2256	0
dashedRectangle()			x								116	0
drawPixel()											40	0
drawText()											1976	16 x no. of chars
ellipse()											684	
fatLine()											2604	
filledArc()					x						572	4 x height of rectangle
filledCircle()											352	
filledEllipse()					x							
filledPolygon()											2052	
filledRectangle()											188	
getFontStruct()											116	
getPixel()										x	76	
getTextWidth()											292	16 x no. of chars
initGraphics()										x	628	112+ 3 x screen width
line()						x					60	0
polyline()						x	x	x			844	0
rectangle()								x			116	0
setArcMode()											28	0
setBackground()											16	0
setClipRect()											196	0
setDashPattern()									x		112	0
setDashStyle()									x			
setFillRule()											36	0
setFillStyle()											36	0
setFont()											36	0
setForeground()											16	0
setLineWidth()											20	
setOrigin()											28	0
setPattern()											68	0
setPatternOrigin()											68	0
setPixelProcessing()											44	0
setTextDirection()											28	
setTransparency()											16	0
userinit()											800	4 x no. of fonts

SDL Functions Code Size and Dependency Chart

Some of the SDL functions use common code with different entry points. This means that the scaleable capabilities of SDL are, in some cases, limited to groups of functions. The chart below shows the SDL functions that must be used as a group. The x's in the column below indicate the functions that must be used as a group. Each column represents a separate group of functions. The Code and RAM requirements in bytes are for the processor listed.

See Appendix B for SDL font sizes.

Code Size for 603 PowerPC Processor

| Function	Groups of Functions											Code	RAM
arc()												3868	68
circle()												460	
clearScreen()											x	84	0
closeGraphics()											x		
dashedLine()		x										64	0
dashedPolyline()		x	x	x								3868	0
dashedRectangle()			x									168	0
drawPixel()												60	0
drawText()												3124	16 x no. of chars
ellipse()												794	
fatLine()													
filledArc()					x							784	168 + 4 x rect. height
filledCircle()												454	
filledEllipse()					x								
filledPolygon()												3504	254
filledRectangle()												256	0
getFontStruct()												168	0
getPixel()											x	108	0
getTextWidth()												568	16 x no. chars
initGraphics()											x	796	156 + 3 x screen width
line()						x						64	0
polyline()						x	x	x				1508	0
rectangle()							x					168	0
setArcMode()												32	0
setBackground()												16	0
setClipRect()												316	0
setDashPattern()									x			140	0
setDashStyle()									x				
setFillRule()												48	0
setFillStyle()												48	0
setFont()												56	0
setForeground()												16	0
setLineWidth()													
setOrigin()												24	0
setPattern()												88	0
setPatternOrigin()													
setPixelProcessing()												72	0
setTextDirection()													
setTransparency()												16	0
userinit()												812	4 x no. of fonts

Appendix D: Cursors and Cursor Bitmaps

The bitmap, size and name of the predefined cursors supported by SDL are shown in the table below. Cursors 40-49 are only available on VMEbus graphics boards.

Cursor Number	Cursor Size	Cursor Name	Cursor Bitmap
0	16 x16	MSC_left_arrow	↖
1	16 x16	MSC_arrow	↗
2	16 x16	MSC_center_ptr	↑
3	16 x16	MSC_down_center_ptr	↓
4	16 x16	MSC_double_arrow	↕
5	16 x16	MSC_lr_double_arrow	↔
6	16 x16	MSC_fleur	✛
7	16 x16	MSC_exchange	⇄
8	16 x16	MSC_left_side	⊢
9	16 x16	MSC_right_side	⊣
10	16 x16	MSC_top_side	⊤
11	16 x16	MSC_bottom_side	⊥
12	16 x16	MSC_top_left_corner	◸
13	16 x16	MSC_top_right_corner	◹
14	16 x16	MSC_bottom_left_corner	◺
15	16 x16	MSC_bottom_right_corner	◿
16	16 x16	MSC_sb_left_arrow	⇐
17	16 x16	MSC_sb_right_arrow	⇒
18	16 x16	MSC_sb_up_arrow	⇑
19	16 x16	MSC_sb_down_arrow	⇓
20	16 x16	MSC_sb_h_double_arrow	↔
21	16 x16	MSC_sb_v_double_arrow	↕
22	16 x16	MSC_circle	○
23	16 x16	MSC_target	◈
24	16 x16	MSC_cross	╪
25	16 x16	MSC_crosshair	┼
26	16 x16	MSC_plus	✚
27	16 x16	MSC_tcross	┼

Cursor Number	Cursor Size	Cursor Name	Cursor Bitmap
28	16 x16	MSC_left_hand1	
29	16 x16	MSC_hand1	
30	16 x16	MSC_hand2	
31	16 x16	MSC_right_hand2	
32	16 x16	MSC_leftbutton	
33	16 x16	MSC_middlebutton	
34	16 x16	MSC_rightbutton	
35	16 x16	MSC_xterm	
36	16 x16	MSC_watch	
37	16 x16	MSC_pencil	
38	16 x16	MSC_gumby	
39	16 x21	MSC_hour_glass16	
40	32x32	MSC_NW_arrow32	
41	32x32	MSC_NE_arrow32	
42	32x32	MSC_NW_hand32	
43	32x32	MSC_NE_hand32	
44	32x32	MSC_xhair32a	
45	32x32	MSC_xhair32b	
46	32x32	MSC_xhair32c	
47	32x32	MSC_xhair32d	
48	32x32	MSC_watch32	
49	32x32	MSC_hour_glass32	

Appendix E: Video Capture Extensions

This appendix describes the SDL extensions for capturing and displaying video signals (NTSC or PAL) as used with graphics cards with onboard video decoders. The decoder digitizes the incoming video stream, converts to YUV 4:2:2 or RGB16 format and feeds it either to the VPORT port on the graphics controller or DMA's it direct to frame buffer memory. The graphics controller supports cropping and scaling of the input image and panning and zooming of the image as it is displayed on screen.

The size and position of the image on the screen is controlled by three rectangles: vidSrcRect, vidCapRect and vidDstRect and the pan start point vidPanStart. Their relationship is roughly this:

The functions *setVIdeoSourceRect()*, *setCaptureSize()* and *setVideoDisplayRect()* are used to set the size and position of these rectangles. *SetVideoPanStart()* sets the starting location in the capture rectangle that maps to the upper left corner of the display rectangle. The scale and zoom factors are automatically computed and set based on the three basic rectangles.
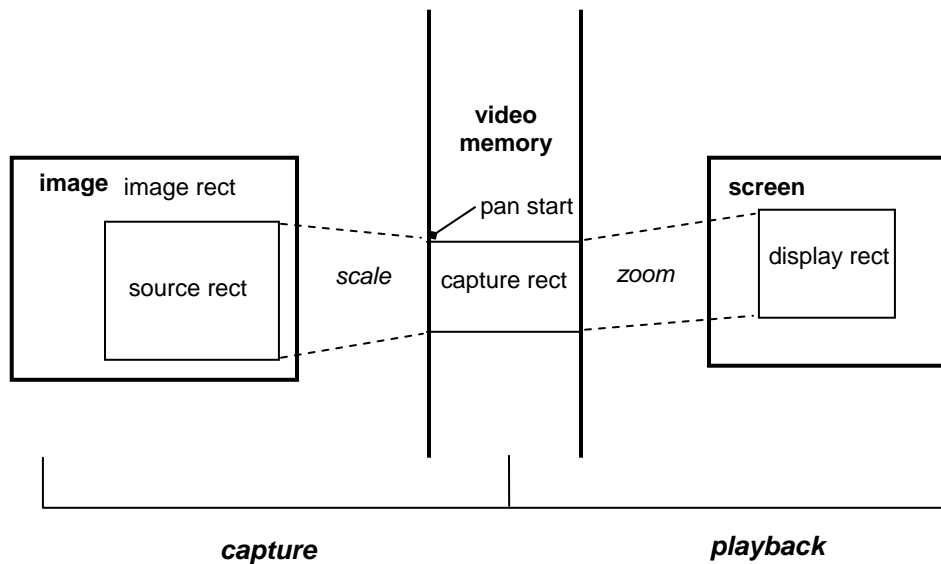
Video Capture Programming Example

This example shows how to use some of the SDL video capture extensions. The example starts by initializing the graphics and video capture hardware, then sets up the hardware to capture and display the upper left quarter of the video source image. The color-key feature is used to show video only in a specific window on the display. Next, the video image is resized to full screen capture and display. The use of color-keying to overlay graphics objects on top of the video image is demonstrated, followed by a single frame capture. Finally, the video image is rescaled to one-half full size and displayed in the center of the screen.

Listing of captest.c

```c
/* example video capture program */
#include <sdl.h>
#include <extern.h>
#include <colors.h>

/* used for overlay graphics */
sPoint triangle[3] = { {120, 380}, {80, 460}, {160, 460} };

int captest(int argc, char **argv)
{
  /* initialize graphics hardware */
  initGraphics(argc, argv);

#ifndef SDL2_X
  /* in SDL 3.1 and later, we need to configure the video decoder  */
  /* and capture port on the graphics controller NTSC analog input */
  /* to first decoder and YUV422 output on SPI port                */
  setVideoDeviceConfig(VDEV_DECODER1, VDEC_ANALOG, NTSC, VDEC_SPI,
      VID_YUV422);
  /* associate first video decoder with first graphics controller, */
  /* with input to the graphics controller coming from the Vport   */
  /* interface in YUV422 pixel format                              */
  setVideoDisplayConfig(GDEV_0, VDEV_DECODER1, VPORT_VPORT, VID_YUV422,
      CAPTURE_ODD, 0, 0);
#endif

  /* initialize video capture hardware for NTSC video */
  initCapture(NTSC);

  /* select first composite video input (VIN0) */
  videoSelect(CVIDEO1, FALSE);

  /* set source and display rectangles for 1/4 frame  */
  /* capture and display (full size would be 640x480) */
  setVideoSourceRect(0, 0, 320, 240);
  setVideoDisplayRect(0, 0, 320, 240);

  /* enable video capture hardware */
  capture(1);

  /* enable video playback */
  playback(1);
```

```
  /* grab continuous frames of video */
  grabFrame(0);

  /* 1/4 frame capture; colorkeyed display */
  /* (video only where blue rectangle is)  */

  /* turn video playback off, while we set things up */
  playback(0);

  /* draw a blue rectangle in video memory */
  setForeground(LightBlue);
  filledRectangle(100, 100, 160, 120);

  /* set color key color and enable color-keying */
  setColorKey(LightBlue);
  colorKey(1);

  /* turn video playback on */
  playback(1);

  /* resize source and display rectangles to full frame */
  setVideoSourceRect(0, 0, 640, 480);
  setVideoDisplayRect(0, 0, 640, 480);

  /* draw some graphics objects on the screen */
  clearScreen();
  setForeground(LightRed);
  filledRectangle(160, 200, 320, 60);
  setForeground(LightBlue);
  circle(100, 100, 60);
  setForeground(Yellow);
  rectangle(320, 280, 100, 100);
  setForeground(LightMagenta);
  filledPolygon(3, triangle);

  /* By setting the color key to Black, the video image will be     */
  /* displayed anywhere there is black - which is everywhere there   */
  /* are no graphics drawn. This achieves the effect of having the   */
  /* graphics appear as an overlay on top of the video image when    */
  /* color-keying is turned on, or video overlayed on top of graphics */
  /* when color-keying is turned off.                                */
  setColorKey(Black);

  /* full frame capture with overlayed video - no graphics visible */
  colorKey(0);

  /* full frame capture with overlayed graphics - graphics "on top" */
  colorKey(1);

  /* single frame grab */
  grabFrame(1);

  /* continuous grab */
  grabFrame(0);

  /* scale acquisition by 1/2 (centered) */
#ifndef SDL2_X
  setVideoSourceRect(0, 0, 320, 240);
#else
```

```
  /* note that capture width is 2x the pixel width */
  setCaptureSize(640, 240); /* half-size */
#endif

  /* resize displayed rectangle to match and */
  /* position it in the center of the screen */
  setVideoDisplayRect(160, 120, 320, 240);

  /* turn off capture and playback */
  capture(0);
  playback(0);
  colorKey(0);

  /* graphics driver cleanup */
  closeGraphics();
}
```

Video Capture Extensions C Function Summary

Function Name	Description
void **capture**(int con)**;**	Enable/disable video capture
int **checkDma**(sDmaInfo *pDma);	Check status of current DMA transfer
void **colorKey**(int kon)**;**	Enable/disable color key insert of video
int **decoderFrameIRQ**(void)	Report status of video decoder interrupt
void **decoderFrameIRQClear**(void)	Clear any pending video decoder interrupts
void **doubleBuffer**(int dbon);	Enable/disable double buffer capture
void **enableDecoderIRQ**(void)	Enable/disable video decoder interrupt
void **enableVBlankIRQ**(void)	Enable/disable video blanking interrupt
int **getCapBuffSize**(void)	Get size of the video capture buffer
int **getDisplayBuffer**(void)	Get current video display buffer number
unsigned char ***getVideoFramebufPtr**(void)**;**	Get pointer to video capture buffer
void **getVideoImage**(int x, int y, int w, int h, unsigned char *buff, int buffno);	
	Copy image from capture to host memory
void **grabFrame**(int sshot);	Grab frames continuous or single-shot
void **initCapture**(int mode);	Initialize video capture hardware
void **interlace**(int ion, int zoom);	Force interlaced or non-interlaced capture
int **lastBuffer**(void);	Report last buffer filled (double buffering)
void **playback**(int pon);	Enable/disable video playback
void **putVideoImage**(unsigned char *buff, int x, int y, int w, int h, int buffno);	
	Copy image from host to capture memory
void **setCaptureSize**(int cwidth, int cheight);	Set size of video capture buffer
void **setColorKey**(unsigned long color); insert	Set color used for color-keying video
void **setDisplayBuffer**(int buffnr)	Set current video display buffer number
int **setVideoCaptureConfig**(int gdev, int vdec, int iport, int format, int fields, int skip, int window);	
	Configure video capture/display hardware
void **setVideoCropOffset**(int ox, int oy);	Set offset to default cropping rect for video capture
int **setVideoDeviceConfig**(int vdev, int iport, int iformat, int oport, int oformat);	
	Configure video decoder hardware
void **setVideoDisplayRect**(int x, int y, int w, int h);	Set size and position of video insert
void **setVideoFormatRect**(int x, int y, int w, int h, int vf);	Set size of video source data
void **setVideoPanStart**(int sx, int sy);	Pan displayed video insert rectangle
void **setVideoSourceRect**(int x, int y, int w, int h);	Set size and position of capture window
sDmaInfo ***startDma**(int dev, int flags, unsigned char *vptr, unsigned char *pptr);	
	Initiate a DMA transfer
void **stopDma**(sDmaInfo *pDma);	Stop current DMA transfer
int **vBlankIRQ**(void)	Report status of vertical blanking interrupt
void **vBlankIRQClear**(void)	Clear any pending graphics controller interrupts
char **videoBrightness**(char bright);	Set brightness level of digitized video
int **videoContrast**(int cont);	Set contrast level of digitized video
char **videoHue**(char hue);	Set hue level of digitized video
int **videoSatU**(int satu);	Set U saturation level of digitized video

int **videoSatV**(int satv);   Set V saturation level of digitized video
void **videoSelect**(int channel, int monochrome);   Select video capture source
int **waitForFrame**(int rdy);   Wait for frame ready (or not ready)
int **waitForSync**(void);   Wait for Vsync or a timeout
void **writeYUV**(unsigned int pixel, void *fp, int gray, int raw);   Write 32-bit word as 2 YUV pixels
void **writeRGB16**(unsigned int pixel, void *fp, int gray, int raw); Write 32-bit word as 2 565 pixels
void **writeRGB32**(unsigned int pixel, void *fp, int gray, int raw); Write 32-bit word as 3 byte RGB

capture

NAME

capture - enable/disable video capture

SYNOPSIS

```
void capture
     (
     int con                    /* 0 == disable, 1 == enable */
     )
```

DESCRIPTION

When *con* is TRUE, this function configures the video capture hardware and enables it. When *con* is FALSE, the video capture hardware is disabled.

No video data will actually be stored in capture memory unless the graphics processor hardware has been started with **grabFrame()**. No video data will be visible on the display unless video playback has been enabled with **playback()**.

INCLUDE FILES

sdl.h

SEE ALSO

grabFrame(), playback()

checkDma

NAME

checkDma – test for completion of a DMA operation

SYNOPSIS

```
int checkDma
    (
    sDmaInfo *dma              /* DMA info for this transfer */
    )
```

DESCRIPTION

This function may be used to check for completion of a non-continuous DMA operation previously started with *startDma()*.

RETURNS

int /* DMA status: 0 = not finished, 1 = completed */

INCLUDE FILES

sdl.h

SEE ALSO

startDma(), stopDma()

colorKey

NAME

colorKey - enable/disable color key insert of video

SYNOPSIS

```
void colorKey
    (
    int kon                    /* 0 == disable, 1 == enable */
    )
```

DESCRIPTION

Color keying is used to "window" the displayed video image onto the screen. When color keying is enabled, the pixels of the screen that are displaying the color key color are replaced by the corresponding pixel of the video image data. When *kon* is TRUE, color keying is enabled; when *kon* is FALSE, color keying is disabled. Use **setColorKey()** to set the color key color.

Color keying can also be used to simulate a display that has a video image with overlayed graphics. Normally, on a graphics display, the background color is black. By setting the color key color to black and enabling color keying, all of the black pixels will display parts of the video image, while graphics other than black will display those graphics. This makes it appear as though the graphics were drawn on top of (overlayed on) the video image. This will not work if the graphics data contains objects that are drawn or filled in black.

INCLUDE FILES

sdl.h

SEE ALSO

setColorKey()

video image

color key region

color keyed result
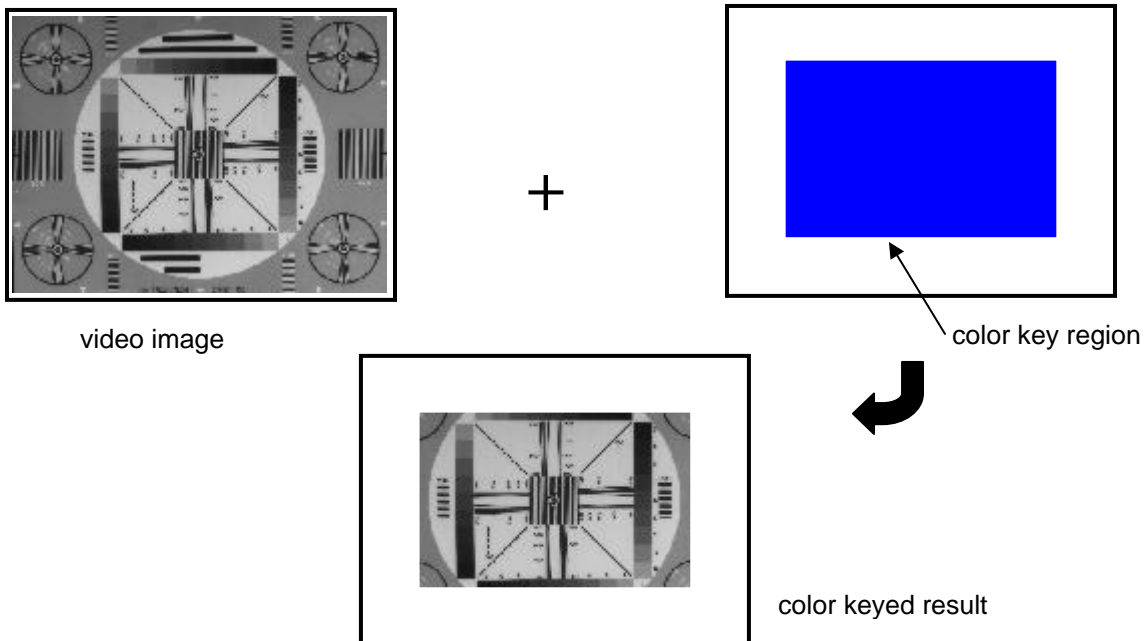
decoderFrameIRQ

NAME

decoderFrameIRQ – report status of video decoder interrupt

SYNOPSIS

```
int decoderFrameIRQ(void)
```

DESCRIPTION

This function can be used to help determine the source of an interrupt in an interrupt handler. If the function returns true, the interrupt should be cleared by calling **decoderFrameIRQClear()** after doing any required interrupt processing.

RETURNS

int /* 0 = no pending interrupt, 1 = interrupt pending */

INCLUDE FILES

sdl.h

SEE ALSO

decoderFrameIRQClear(), enableDecoderIRQ()

decoderFrameIRQClear

NAME

decoderFrameIRQClear – clear any pending video decoder interrupts

SYNOPSIS

```
void decoderFrameIRQClear(void)
```

DESCRIPTION

This function is called at the end of an interrupt handler or polling loop to clear the interrupt request after it has been handled.

INCLUDE FILES

sdl.h

SEE ALSO

decoderFrameIRQ(), enableDecoderIRQ()

doubleBuffer

NAME

doubleBuffer – enable or disable double buffering on capture

SYNOPSIS

```
void doubleBuffer
    (
    int dbon                /* 0 == disable, 1 == enable */
    )

void doubleBuffer
    (
    int extra_bufs          /* number of extra buffers (Atlas) */
    )
```

DESCRIPTION

This function may be used to override the default use of double buffer for video capture. The default for non-interlaced sources is to not use double buffering. It is enabled, by default, when capturing interlaced video on the RG-101 boards.

Double buffering places each captured field (or frame if non-interlaced) into alternating memory buffers. This is most often used when capturing interlaced video to allow interleaving the two fields to build a complete frame for display. It is also useful when doing post processing on the video data, as it allows capturing a new frame while working on another frame.

With the VFG-M boards, interlaced video is automatically stored in interleaved memory such that when both fields have been captured, the full frame is available in contiguous capture memory and the double buffering flag is usually ignored.

Starting with SDL version 3.6.1 the Atlas driver supports multiple capture buffers. Each buffer will contain a full frame of video data. The *extra_bufs* argument to **doubleBuffer()** specifies the number of extra capture buffers desired. Each buffer is of length **getCapBuffSize()** and adjacent to each other in memory. An argument value of 1 means one extra buffer (total of two), an argument of 2 means two extra buffers (three total). The default is 0 (no extra buffers).

INCLUDE FILES

sdl.h

SEE ALSO

interlace(), getCapBuffSize()

enableDecoderIRQ

NAME

enableDecoderIRQ – enable/disable video decoder interrupt

SYNOPSIS

```
void enableDecoderIRQ(int onOff)
```

DESCRIPTION

EnableDecoderIRQ is used to enable (*onOff* = 1) or disable (*onOff* = 0) interrupts from the video decoder. Interrupts are generated at the end of each frame, which may consist of odd fields only, even fields only, or both odd and even fields. Video decoder interrupts are not supported on all graphics boards.

INCLUDE FILES

sdl.h

SEE ALSO

decoderFrameIRQ(), decoderIRQClear()

enableVBlankIRQ

NAME

enableVBlankIRQ – enable/disable video blanking interrupt

SYNOPSIS

```
void enableVBlankIRQ(int onOff)
```

DESCRIPTION

EnableVBlankIRQ is used to enable (*onOff* = 1) or disable (*onOff* = 0) interrupts from the graphics controller. Interrupts are generated at the start of each vertical blanking interval. Video blanking interrupts are not supported on all graphics boards.

INCLUDE FILES

sdl.h

SEE ALSO

vBlankIRQ(), vBlankIRQClear()

getCapBuffSize

NAME

getCapBuffSize – get size of the video capture buffer

SYNOPSIS

```
int getCapBuffSize(void)
```

DESCRIPTION

This routine returns the size of the video capture buffer. This size is valid only for the current device context. If the active video device is changed, the size may become invalid, so this function should be called again to obtain the updated value.

This is most often used in double buffer situations to find the address of the second and subsequent video capture or display buffers given the starting address of the capture region.

RETURNS

int /* size of video capture buffer */

INCLUDE FILES

sdl.h

SEE ALSO

doubleBuffer(), getVideoFramebufPtr()

getDisplayBuffer

NAME

getDisplayBuffer – gets current video display buffer number

SYNOPSIS

`int getDisplayBuffer(void)`

DESCRIPTION

This routine returns the index of the video display buffer in current use.

This is most often used in double buffer situations. Not all graphics boards support multiple video display buffers. Note this is different from multiple graphics display buffers (pages), manipulated by **getDisplayPage()** and **getWritePage()**. The video display buffer contains captured video data for display by the graphics controller harware overlay feature.

RETURNS

int /* index (0-n) of video display buffer */

INCLUDE FILES

sdl.h

SEE ALSO

setDisplayBuffer()

getVideoFramebufPtr

NAME

getVideoFramebufPtr – gets current value of the video capture buffer pointer

SYNOPSIS

`unsigned char *getVideoFramebufPtr(void)`

DESCRIPTION

This routine looks up the current value of the video capture buffer pointer. This pointer is only valid for the current device context. If the active graphics device is changed, the pointer may become invalid, so this function should be called again to obtain the updated value.

RETURNS

unsigned char * /* pointer to video capture buffer memory (in CPU address space) */

INCLUDE FILES

sdl.h

SEE ALSO

getCapBuffSize(), getFramebufPtr()

getVideoImage

NAME

getVideoImage - copy image from capture memory to host memory

SYNOPSIS

```
void getVideoImage
    (
    int x,                  /* x coord of upper left corner */
    int y,                  /* y coord of upper left corner */
    int w,                  /* width of image to copy */
    int h,                  /* height of image to copy */
    unsigned char *buff,    /* buffer for image data */
    int buffno              /* buffer number (0 or 1) */
    )
```

DESCRIPTION

This function copies image data from the video capture buffer and puts it in the buffer pointed to by *buff*. *X* and *y* are the starting position within the capture buffer; *w* and *h* are the width and height of the rectangular area copied. Note that the video data is typically 16-bits wide (4:2:2 YUV), so the size of the buffer pointed to by *buff* should be twice the width times height. In double buffer configurations, setting *buffno* to 1 will copy the data from the second capture buffer.

INCLUDE FILES

sdl.h

SEE ALSO

putVideoImage(), writeRGB16(), writeRGB32, writeYUV(), **video capture example programs**
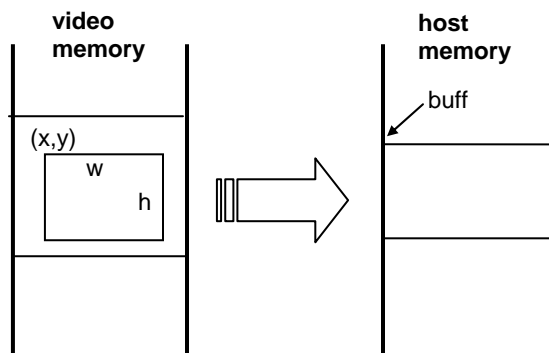
grabFrame

NAME

grabFrame - grab frames continuously or as a single-shot

SYNOPSIS

```
void grabFrame
    (
    int sshot                   /* 0==continuous, 1==single shot */
    )
```

DESCRIPTION

This function signals the graphics processor to read and store the digitized video into the video capture buffer. When *sshot* is 1 or -1, a single frame is grabbed and transferred to the video capture buffer, otherwise the video data in the capture buffer is continuously updated. Normally, the grab is not initiated until after the next vertical sync. The special value –1 for *sshot* skips the wait for vertical sync in single shot mode, allowing the capture to overlap other processing.

No video data will actually be stored in capture memory unless the video capture hardware has been enabled with **capture()**. No video data will be visible on the display unless video playback has been enabled with **playback()**.

INCLUDE FILES

sdl.h

SEE ALSO

capture(), playback()

initCapture

NAME

initCapture - initialize video capture hardware

SYNOPSIS

```
void initCapture
      (
      int mode                    /* video source format */
      )
```

DESCRIPTION

This function initializes the video capture and processing hardware. It does not actually start the capture or display process. In SDL version 3.1 and later, the video decoder hardware and video capture hardware are configured independently, as some graphics boards have multiple decoders and multiple graphics controllers on the same board. This configuration is done with the **setVideoDeviceConfig()** and **setVideoCaptureConfig()** functions. The available modes are listed in *sdl.h* as:

```
#define NTSC              0    /* NTSC square pixel [640x480] */
#define PAL               1    /* PAL square pixel [768x576] */
#define CCIR_NTSC         2    /* NTSC CCIR601 [720x480] */
#define CCIR_PAL          3    /* PAL CCIR601 [720x576] */
#define NTSC_2_1          4    /* NTSC square pixel CIF (2:1 scaling) [320x240] */
#define PAL_2_1           5    /* PAL square pixel CIF (2:1 scaling) [384x288] */
#define SECAM             8    /* SECAM */
#define CCIR_656          16   /* CCIR 656 digital video */
#define SMPTE_125         17   /* Modified SMPTE-125 digital video */
#define VGA_RGB           18   /* RGBHV input to AD9882 on Stratus */
#define VGA_MONO          19   /* monochrome RGB input to AD9882 on Stratus */
#define VGA_YC            19   /* old name */
#define VGA_RGB_SOG       20   /* RGB+SOG input to AD9882 on Stratus */
#define VGA_MONO_SOG      21   /* monochrome RGB+SOG input to AD9882 */
#define VGA_DVI           22   /* DVI input to AD9882 on Stratus */
#define STANAG_A          23   /* STANAG-A input to AD9882 [1080x808 interlaced]
*/
#define STANAG_B          24   /* STANAG-B input to AD9882 [768x574 interlaced] */
#define STANAG_C          25   /* STANAG-C input to AD9882 [640x484 interlaced] */
#define SONY_DXC990       26   /* Sony DXC-990 cam. with AD9882 [768x494 ilace] */
```

Note: not all capture modes are available on all graphics boards. The STANAG-A/B/C capture modes allow capturing a monochrome composite STANAG format signal using the onboard AD9882 decoder on the Stratus and Garnet boards. The video source is fed to the Red (VIN0) input. The Sony DXC-990 capture uses the RGB + SOG output from the camera. The decoder is forced into interlaced capture.

INCLUDE FILES

sdl.h

SEE ALSO

capture(),   grabFrame(),   playback(),   setVideoDeviceConfig(), setVideoCaptureConfig()

interlace

NAME

interlace - force interlaced/non-interlaced video capture

SYNOPSIS

```
void interlace
      (
      int ion,              /* 0 = non-interlaced, 1 = interlaced */
      int zoom              /* 0 = normal, 1 = 2X vertical zoom */
      )
```

DESCRIPTION

This routine is used to override the default setting of the interlaced capture flag. The default for most video decoders is to enable interlaced operation. The default for the AD9882 decoder, used on the some graphics boards, is non-interlaced. When *ion* is TRUE, interlaced video capture is enabled. This also enables double buffering.

The *zoom* flag enables a 2x vertical zoom in non-interlaced mode. This would be useful when aquiring video data from an interlaced source with the controller in non-interlaced mode. This has the effect of capturing half the active lines and compressing them into half the normal memory space. E.g. on a 640x480 interlaced camera image, acquiring in non-interlaced mode will use only 640x240*2 bytes of capture memory. The 2x vertical zoom will restore the original vertical size (with duplicated lines).

INCLUDE FILES

sdl.h

SEE ALSO

doubleBuffer (), initCapture()

lastBuffer

NAME

lastBuffer – gets number of the last buffer filled

SYNOPSIS

`int lastBuffer(`void`)`

DESCRIPTION

In double buffer capture (default for interlaced video sources), two memory buffers are used for the digitized video data (one for each field). This function can be used to determine which buffer was last filled.

RETURNS

int /* last buffer filled (0 or 1) */

INCLUDE FILES

sdl.h

SEE ALSO

doubleBuffer()

playback

NAME

playback - enable/disable video playback

SYNOPSIS

```
void playback
     (
     int pon                    /* 0 = disable, 1 = enable */
     )
```

DESCRIPTION

This routine is used enable and disable the display of the captured video data on the screen. When *pon* is TRUE, playback is enabled; when *pon* is FALSE, playback is disabled.

No video data will actually be stored in capture memory unless the video capture hardware has been enabled with ***capture**()* and the graphics processor hardware has been started with ***grabFrame()***.

INCLUDE FILES

sdl.h

SEE ALSO

capture(), grabFrame()

putVideoImage

NAME

putVideoImage - copy image from host memory to capture memory

SYNOPSIS

```
void putVideoImage
    (
    unsigned char *buff,    /* buffer containing image data */
    int x,                  /* x coord of upper left corner */
    int y,                  /* y coord of upper left corner */
    int w,                  /* width of image to copy */
    int h,                  /* height of image to copy */
    int buffno              /* buffer number (0 or 1) */
    )
```

DESCRIPTION

This function copies image data from the buffer pointed to by *buff* and puts it in the video capture buffer. *X* and *y* are the starting position within the capture buffer; *w* and *h* are the width and height of the rectangular area copied. Note that the video data is typically 16-bits wide (4:2:2 YUV), so the size of the buffer pointed to by *buff* should be twice the width times height. In double buffer configurations, setting *buffno* to 1 will copy the data from the second capture buffer.

INCLUDE FILES

sdl.h

SEE ALSO

getVideoImage(), writeRGB16(), writeRGB32, writeYUV(), **video capture example programs**
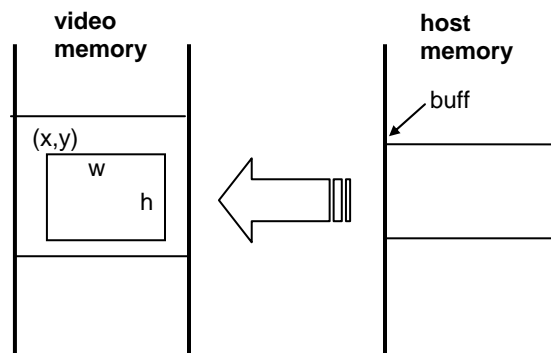
setCaptureSize

NAME

setCaptureSize - set size of video capture buffer

SYNOPSIS

```
void setCaptureSize
    (
    int cwidth,                 /* width of capture buffer rect */
    int height                  /* height of capture buffer rect */
    )
```

DESCRIPTION

This function sets the size (width and height) of the memory region used for the video capture buffer. The digitized video stream is scaled, by dropping pixels or scan lines, to fit into the capture buffer width and height. When possible, downsizing is done in the video decoder device, then via the V-Port hardware on the graphics chip, if applicable.

The function *setVideoSourceRect()* automatically calls *setCaptureSize()* to resize the capture buffer to match the source size. To change the scaling, then, *setCaptureSize()* must be called after any calls to *setVideoSourceRect()*.

Note: for proper image display, the capture buffer size (in pixels) should be at least as large as the video display rectangle.

NOTES

In SDL version 2.x, the width should be set to twice the desired pixel width, as the digitized video stream is 16-bits per pixel. In SDL 3.1 and later, the actual capture buffer width is automatically computed based on the video data pixel size. In SDL 3.1 and later, this function is typically not used. In most instances, *setVideoSourceRect()* should be used instead.

INCLUDE FILES

sdl.h

SEE ALSO

setVideoDisplayRect(), setVideoSourceRect()

setColorKey

NAME

setColorKey - set color used for color-keying video insert

SYNOPSIS

```
void setColorKey
      (
      unsigned long color      /* color key color */
      )
```

DESCRIPTION

Color keying is used to "window" the displayed video image onto the screen. When color keying is enabled, the pixels of the screen that are displaying the color key color are replaced by the corresponding pixel of the video image data. *Color* specifies the color to use as the color key. Use ***colorKey()*** to actually enable and disable the color key operation.

Color keying can also be used to simulate a display that has a video image with overlayed graphics. Normally, on a graphics display, the background color is black. By setting the color key color to black and enabling color keying all of the black pixels will display parts of the video image, while graphics other than black will display those graphics. This makes it appear as though the graphics were drawn on top of (overlayed on) the video image. This will not work if the graphics data contains objects that are drawn or filled in black.

INCLUDE FILES

sdl.h, colors.h

SEE ALSO

colorKey()

video image

+

color key region

color keyed result
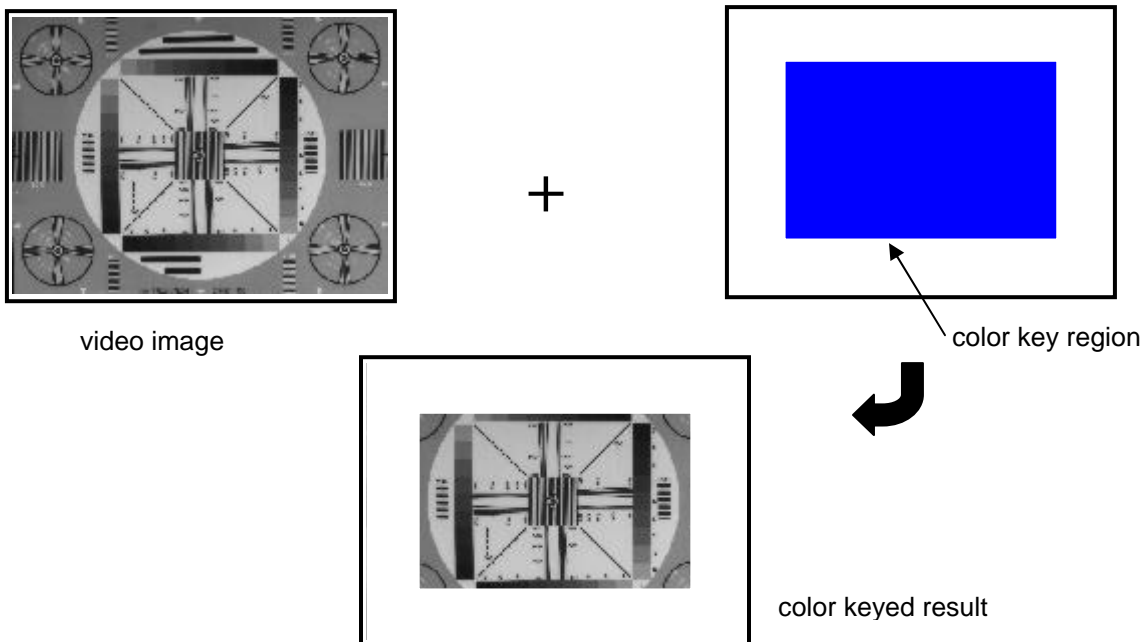
setDisplayBuffer

NAME

setDisplayBuffer – set current video display buffer number

SYNOPSIS

```
void setDisplayBuffer
    (
    int buffer                /* buffer number to display */
    )
```

DESCRIPTION

This routine sets the index of the active video display buffer. The change becomes effective at the next video blanking interval.

This is most often used in double buffer situations. Not all graphics boards support multiple video display buffers. The Atlas board supports upto six video buffers. Note this is different from using multiple graphics display buffers (pages), as manipulated by *setDisplayPage()* and *setWritePage()*. The video display buffer contains captured video data for display by the graphics controller hardware overlay feature.

INCLUDE FILES

sdl.h

SEE ALSO

doubleBuffer(), getDisplayBuffer()

setVideoCaptureConfig

NAME

setVideoCaptureConfig - configure video capture/display hardware

SYNOPSIS

```
void setVideoCaptureConfig
      (
      int gdev,                  /* graphics controller device */
      int vdec,                  /* video decoder device */
      int iport,                 /* input port on graphics device */
      int format,                /* pixel format */
      int fields,                /* which fields to capture */
      int skip,                  /* number of fields/frames to skip*/
      int window                 /* video window number */
      )
```

DESCRIPTION

This function configures the video capture and display hardware and associates a video decoder device to this capture/display configuration. This function should be called after the video decoder is configured via **setVideoDeviceConfig()**. *Window* can be set to force the use of a specific overlay window when the hardware supports more than one, or to force the use of software overlay when set to 0. Not all graphics boards will support all the features of this command. [E.g., the RG-101 has only one video decoder and one graphics controller, while the VFG-M has two decoders and two controllers.] **Note:** this function is only available in SDL version 3.1 and later.

Macro definitions for most of the function parameters are defined in *sdl.h*:

/* graphics controller device */
#define GDEV_0	0x000	/* first (or only) graphics controller */
#define GDEV_1	0x100	/* second graphics controller */

/* video decoder device */
#define VDEV_HOST	0	/* host supplied video data */
#define VDEV_DECODER1	1	/* first (or only) video decoder */
#define VDEV_DECODER2	2	/* second video decoder */
#define VPORT_VPORT	0	/* capture digitized video on V-Port */
#define VPORT_PCI	1	/* capture digitized video from the PCI bus */

/* decoder input port */
#define VDEC_ANALOG	0	/* standard analog input to the multiplexor */
#define VDEC_DIGITAL	1	/* CCIR 656 digital video input */
#define VDEC_DVI	2	/* DVI digital video input */

/* decoder output pixel format */
#define VID_YUV422	0	/* YUV 4:2:2 pixel format */
#define VID_RGB16	1	/* RGB 5-6-5 pixel format */
#define VID_RGB24	2	/* RGB 8-8-8 pixel format */
#define VID_RGB32	3	/* RGB 8-8-8-8 pixel format */
#define VID_Y8	4	/* 8-bit luminance only (for monochrome) */

/* field selection */
#define CAPTURE_EVEN	1	/* capture/display EVEN field only */
#define CAPTURE_ODD	2	/* capture/display ODD field only */
#define CAPTURE_BOTH	3	/* capture/display both EVEN and ODD fields */

INCLUDE FILES

sdl.h

SEE ALSO

initCapture(), setVideoDeviceConfig()

setVideoCropOffset

NAME

setVideoCropOffset – set offset to default capture cropping rectangle

SYNOPSIS

```
void setVideoCropOffset
    (
    int ox,                    /* x offset to default left edge */
    int oy                     /* y offset to default top edge */
    )
```

DESCRIPTION

Set current values for video cropping offset. The offsets determine how much of the incoming video image to skip before starting to save data into video memory. This can be used to fine tune the image as it appears in the capture memory to account for differences in vertical or horizontal blanking intervals. It is primary used with the AD982 video capture on the Stratus graphics board. Offset can be positive or negative and is an offset from the default values. *Ox* is specified in pixels and *oy* is lines.

INCLUDE FILES

sdl.h

SEE ALSO

setVideoSourceRect()

setVideoDeviceConfig

NAME

setVideoDeviceConfig - configure video decoder hardware

SYNOPSIS

```
void setVideoDeviceConfig
    (
    int vdev,              /* video decoder device */
    int iport,             /* input port on decoder */
    int iformat,           /* input video format */
    int oport,             /* output port on decoder */
    int oformat            /* output pixel format */
    )
```

DESCRIPTION

This function configures the video decoder hardware on a graphics board. This function should be the first function called prior to using the video capture and display hardware. It provides a way to specify the origin and destination of the video signal/data and the video/pixel format. **Note:** this function is only available in SDL version 3.1 and later.

Macro definitions for most of the function parameters are defined in *sdl.h*:

```
/* video decoder device */
#define VDEV_HOST          0        /* host supplied video data */
#define VDEV_DECODER1      1        /* first (or only) video decoder */
#define VDEV_DECODER2      2        /* second video decoder */

/* decoder input port */
#define VDEC_ANALOG        0        /* standard analog input to the multiplexor */
#define VDEC_DIGITAL       1        /* CCIR 656 digital video input */
#define VDEC_DVI           2        /* DVI digital video input */

/* video input format */
#define NTSC               0        /* standard full scale NTSC video */
#define PAL                1        /* standard full scale PAL video */
#define CCIR_NTSC          2
#define CCIR_PAL           3
#define NTSC_2_1           4        /* 2:1 scaled NTSC video */
#define PAL_2_1            5        /* 2:1 scaled PAL video */
#define CCIR_NTSC_2_1      6
#define CCIR_PAL_2_1       7
#define SECAM              8        /* SECAM */
#define CCIR_656           16       /* CCIR 656 digital video */
#define SMPTE_125          17       /* Modified SMPTE-125 digital video */
#define VGA_RGB            18       /* RGBHV input to AD9882 on Stratus */
#define VGA_MONO           19       /* monochrome RGB input to AD9882 on Stratus */
#define VGA_YC             19       /* old name */
#define VGA_RGB_SOG        20       /* RGB+SOG input to AD9882 on Stratus */
#define VGA_MONO_SOG       21       /* monochrome RGB+SOG input to AD9882 */
#define VGA_DVI            22       /* DVI input to AD9882 on Stratus */
#define STANAG_A           23       /* STANAG-A input to AD9882 [1080x808 interlaced] */
#define STANAG_B           24       /* STANAG-B input to AD9882 [768x574 interlaced] */
#define STANAG_C           25       /* STANAG-C input to AD9882 [640x484 interlaced] */
#define SONY_DXC990        26       /* Sony DXC-990 cam. with AD9882 [768x494 ilace] */
```

/* decoder output port */

#define VDEC_SPI	1	/* Streaming Pixel Interface (V-Port) */
#define VDEC_PCI	2	/* PCI bus */

/* decoder output pixel format */

#define VID_YUV422	0	/* YUV 4:2:2 pixel format */
#define VID_RGB16	1	/* RGB 5-6-5 pixel format */
#define VID_RGB24	2	/* RGB 8-8-8 pixel format */
#define VID_RGB32	3	/* RGB 8-8-8-8 pixel format */
#define VID_Y8	4	/* 8-bit luminance only (for monochrome) */

INCLUDE FILES

sdl.h

SEE ALSO

initCapture(), setVideoCaptureConfig()

setVideoDisplayRect

NAME

setVideoDisplayRect - set size and position of inserted video window

SYNOPSIS

```
void setVideoDisplayRect
    (
    int x,                     /* x coord of origin on screen */
    int y,                     /* y coord of origin on screen */
    int w,                     /* width of displayed image */
    int h                      /* height of displayed image */
    )
```

DESCRIPTION

This function sets the origin and size of the video display window on the screen. If the displayed image are is larger than the source image, the image is zoomed (horizontally and vertically as needed) by pixel and/or row replication before display. Portions of the display rectangle can be off screen, in which case the image is clipped to the screen boundary.

Note: the width and height of the video display rectangle should not be larger than the pixel width and height of the capture buffer.

INCLUDE FILES

sdl.h

SEE ALSO

setCaptureSize(), setVideoSourceRect()
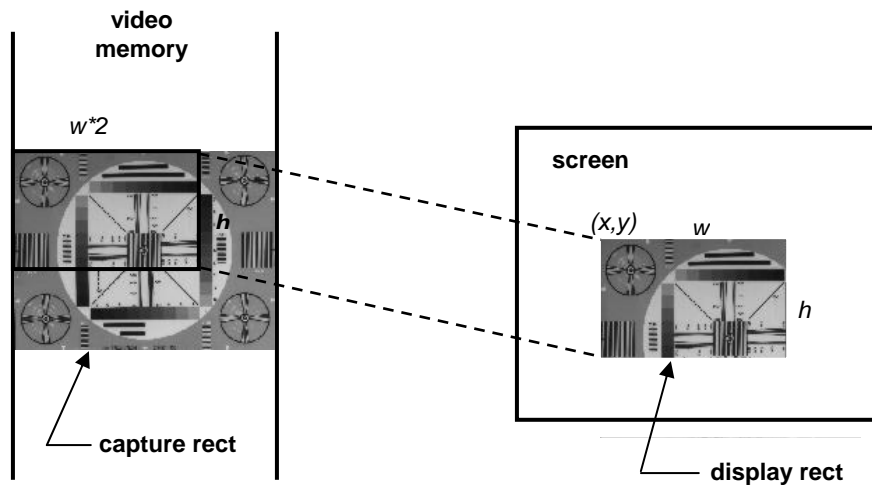
setVideoFormatRect

NAME

setVideoFormatRect - set size and position of capture window

SYNOPSIS

```
void setVideoFormatRect
    (
    int x,                      /* x coord of source origin */
    int y,                      /* y coord of source origin */
    int w,                      /* width of image to capture */
    int h,                      /* height of image to capture */
    int vf                      /* vertical refresh frequency */
    )
```
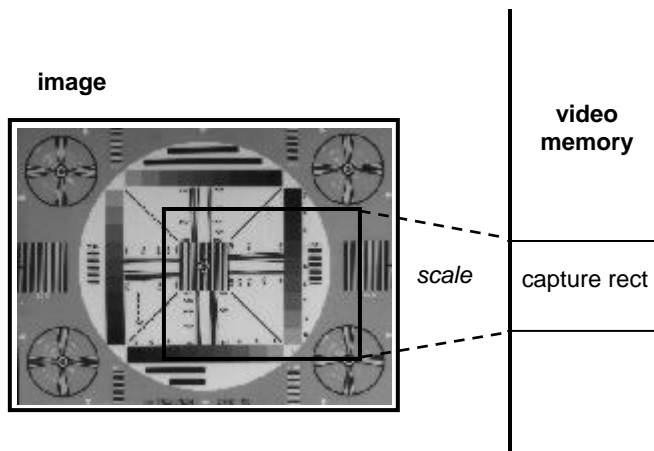
DESCRIPTION

This function sets the origin and size of the video source window (rectangle). This is used primarily when the AD9882 is selected as the video source to "window" a smaller source image in the display. By default, the source image size is assumed to be the same as the display size. This function also provides a method for the driver to optimize the video capture for the specific refresh frequency of the analog RGB image data. *Vf* is specified in Hz. *X* and *y* are usually always zero.

INCLUDE FILES

sdl.h

SEE ALSO

setCaptureSize(), setVideoSourceRect(), setVideoDisplayRect()

source image

(x,y)   *w*

h
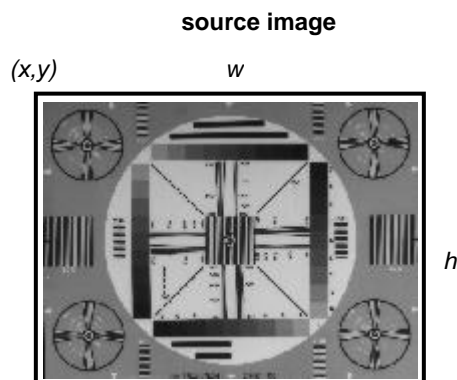
setVideoPanStart

NAME

setVideoPanStart - pan displayed video insert window

SYNOPSIS

```
void setVideoPanStart
    (
    int sx,                     /* x coord of pan starting point */
    int sy                      /* y coord of pan starting point */
    )
```

DESCRIPTION

This routine sets the origin in capture buffer memory for the video display window (rectangle). The default origin is (0, 0) – or the start of the capture buffer. This function, in conjunction with *setVideoDisplayRect()*, allows displaying just a portion of (or window into) the captured video image.

INCLUDE FILES

sdl.h

SEE ALSO

setVideoDisplayRect()

pan start at (0,0)   pan start at (sx,sy)
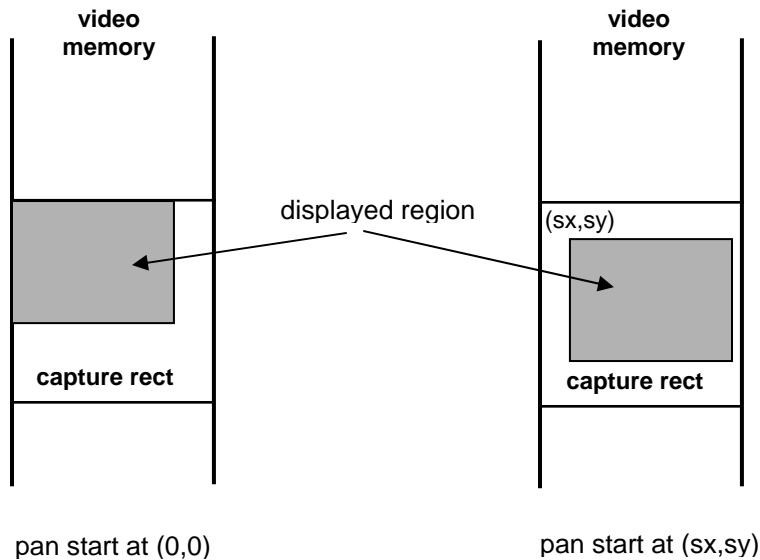
setVideoSourceRect

NAME

setVideoSourceRect - set size and position of capture window

SYNOPSIS

```
void setVideoSourceRect
    (
    int x,                    /* x coord of source origin */
    int y,                    /* y coord of source origin */
    int w,                    /* width of image to capture */
    int h                     /* height of image to capture */
    )
```

DESCRIPTION

This function sets the origin and size of the video source window (rectangle). This can be used to capture all, or just a portion of, the digitized video stream. If the source image is larger than the capture buffer size, the image is scaled (horizontally and vertically as needed) by pixel and/or scan line removal before storing in memory.

INCLUDE FILES

sdl.h

SEE ALSO

setCaptureSize(), setVideoDisplayRect()

startDma

NAME

startDma – start a DMA operation

SYNOPSIS

```
sDmaInfo *startDma
    (
    int dmaDev,             /* DMA source this transfer */
    int flags,              /* DMA flags */
    unsigned char *vptr,    /* virtual address of destination */
    unsigned char *pptr     /* physical address of dest. */
    )
```

DESCRIPTION

This function initiates a DMA transfer from graphics or video memory to another block of memory. The destination is often system memory, but could be the framebuffer of another graphics/video controller or another device on the PCI bus. The value of *vptr* is not used by current drivers, but is saved in the DMA info structure for the convenience of the user.

DmaDev is the source device and can have the following values, defined in *sdl.h*:

```
/* DMA source flags (support varies by board type) */
#define DMA_NONE            0
#define DMA_GDEV0_GRMEM     1      /* graphics device 0 - graphics memory */
#define DMA_GDEV0_VIMEM     2      /* graphics device 0 - video capture  memory */
#define DMA_GDEV1_GRMEM     3      /* graphics device 1 - graphics memory */
#define DMA_GDEV1_VIMEM     4      /* graphics device 1 - video capture  memory */
#define DMA_VDEC0           5      /* video decoder 0 */
#define DMA_VDEC1           6      /* video decoder 1 */
#define DMA_ADEC0           7      /* audio decoder 0 */
#define DMA_ADEC1           8      /* audio decoder 1 */
```

The *flags* argument is used to indicate the transfer type and is OR of the flags bits defined in sdl.h:

```
/* DMA transfer flags (support varies by board type) */
#define DMA_WAIT            0      /* polled wait for entire transfer */
#define DMA_NOWAIT          1      /* interrupt driven transfer */
#define DMA_CONTINUOUS      2      /* copy on each Vblank or ZV port interrupt */
#define DMA_INTERLACE       4      /* transfer even fields only */
#define DMA_GRAPHICS_MEM    8      /* copy graphics mem instead of video mem */
```

RETURNS

sDmaInfo * /* pointer to DMA info structure */

NOTES

DMA_CONTINUOUS and DMA_NOWAIT are available with VxWorks and some Linux versions only. ***The functionality of DMA is very dependent on the CPU board and host bridge chip. Many PowerPC systems have problems, but it is reported to work on the Motorola MVME5100.***

INCLUDE FILES

sdl.h

SEE ALSO

checkDma(), stopDma()

stopDma

NAME

stopDma – stop a DMA operation

SYNOPSIS

```
void stopDma
    (
    sDmaInfo *dma            /* DMA info for this transfer */
    )
```

DESCRIPTION

This function may be used to terminate a DMA operation previously started with *startDma()*.

INCLUDE FILES

sdl.h

SEE ALSO

checkDma(), startDma()

vBlankIRQ

NAME

vBlankIRQ – report status of vertical blanking interrupt

SYNOPSIS

`int vBlankIRQ(void)`

DESCRIPTION

This function can be used to help determine the source of an interrupt in an interrupt handler. If the function returns true, the interrupt should be cleared by calling **vBlankIRQClear()** after doing any required interrupt processing. If the graphics controller supports multiple channels, the return value will indicate which channel generated the interrupt.

RETURNS

int /* 0 = no pending interrupt, non-zero = interrupt pending and channel */

INCLUDE FILES

sdl.h

SEE ALSO

vBlankIRQClear(), enableVBlankIRQ()

vBlankIRQClear

NAME

vBlankIRQClear – clear any pending graphics controller interrupts

SYNOPSIS

```
void vBlankIRQClear(void)
```

DESCRIPTION

This function is called at the end of an interrupt handler or polling loop to clear the interrupt request after it has been handled.

INCLUDE FILES

sdl.h

SEE ALSO

vBlankIRQ(), enableVBlankIRQ()

videoBrightness

NAME

videoBrightness - Set brightness level of digitized video

SYNOPSIS

```
char videoBrightness
     (
     char bright                    /* signed brightness adjustment */
     )
```

DESCRIPTION

This routine is used to adjust the brightness to the digitized video. This value is added to the luminance value to adjust the final brightness. The brightness can be adjusted in 255 steps, from –100% (–128) to +100% (+127). Each step is a 0.78% change with respect to the full scale. The previous brightness adjustment value is returned.

The default value for the brightness adjust is 0 (0%).

INCLUDE FILES

sdl.h

RETURNS

char   /* old brightness adjustment */

SEE ALSO

videoContrast(), videoHue(), videoSatU(), videoSatV()

videoContrast

NAME

videoContrast - set contrast level of digitized video

SYNOPSIS

```
int videoContrast
    (
    int cont                    /* luma gain adjustment */
    )
```

DESCRIPTION

This routine is used to adjust the luma gain of the digitized video. This value is multiplied by the luminance value to provide contrast adjustment. The luma gain can be adjusted in 512 steps, from 236.57% (511) to 0% (0). Each step is a 0.46% change with respect to the incoming luma value. The previous luma gain value is returned.

The default value for luma gain is 216 (100%).

INCLUDE FILES

sdl.h

RETURNS

int   /* old gain value */

SEE ALSO

videoBrightness(), videoHue(), videoSatU(), videoSatV()

videoHue

NAME

videoHue - set hue level of digitized video

SYNOPSIS

```
char videoHue
      (
      char hue                  /* signed hue adjustment */
      )
```

DESCRIPTION

This routine is used to adjust the hue to the digitized video. The hue can be adjusted in 256 steps, from –90° (–128) to +89.3° (+127). Each step is a 0.7° change. The previous hue adjustment value is returned.

The default value for hue adjustment is 0 (0°).

INCLUDE FILES

sdl.h

RETURNS

char   /* old hue adjustment */

SEE ALSO

videoBrightness(), videoContrast(), videoSatU(), videoSatV()

videoSatU

NAME

videoSatU - set U saturation level of digitized video

SYNOPSIS

```
int videoSatU
    (
    int satu                    /* chroma U value */
    )
```

DESCRIPTION

This routine is used to add a gain adjustment to the U component of the video [chroma] signal. This value is multiplied by the luminance value to provide contrast adjustment. The U gain can be adjusted in 512 steps, from 201.18% (511) to 0% (0). Each step is a 0.39% change with respect to the incoming U value. The previous U gain value is returned.

By adjusting the U and V color components of the video stream by the same amount, the saturation is adjusted. For normal saturation adjustment, the gain in both color difference paths must be the same (i.e. the ratio between the U gain and the V gain should be kept constant at the default ratio).

The default value for the U gain is 254 (100%).

INCLUDE FILES

sdl.h

RETURNS

int   /* old gain value */

SEE ALSO

videoBrightness(), videoContrast(), videoHue(), videoSatV()

videoSatV

NAME

videoSatV - set V saturation level of digitized video

SYNOPSIS

```
int videoSatV
    (
    int satv                    /* chroma V value */
    )
```

DESCRIPTION

This routine is used to add a gain adjustment to the V component of the video [chroma] signal. This value is multiplied by the luminance value to provide contrast adjustment. The V gain can be adjusted in 512 steps, from 283.89% (511) to 0% (0). Each step is a 0.56% change with respect to the incoming V value. The previous V gain value is returned.

By adjusting the U and V color components of the video stream by the same amount, the saturation is adjusted. For normal saturation adjustment, the gain in both color difference paths must be the same (i.e. the ratio between the U gain and the V gain should be kept constant at the default ratio).

The default value for the V gain is 180 (100%).

INCLUDE FILES

sdl.h

RETURNS

int   /* old gain value */

SEE ALSO

videoBrightness(), videoContrast(), videoHue(), videoSatU()

videoSelect

NAME

videoSelect - select video capture source

SYNOPSIS

```
void videoSelect
     (
     int channel,              /* video input select */
     int monochrome            /* 0 == color, 1 == monochrome */
     )
```

DESCRIPTION

This function selects which video source is digitized and sent to the graphics processor (when more than one channel is supported). The RG-101 supports three video input sources — two composite video and one component video (Svideo). The VFG-M supports four composite and one component video sources. They are listed in *sdl.h* as:

```
#define CVIDEO1        0     /* composite video 1 (Mux0 on VFG-M) */
#define CVIDEO2        1     /* composite video 2 (Mux1 on VFG-M) */
#define CVIDEO3        3     /* composite video 3 (Mux2 on VFG-M) */
#define CVIDEO4        4     /* composite video 4 (Mux3 on VFG-M) */
#define SVIDEO         2     /* S-video 1 */
#define GPIO_DIG       16    /* digital video on GPIO port */
#define RGBHV          17    /* analog RGBHsVs video */
#define TEST_PATN      64    /* internal test pattern (e.g. color bars) */
#define LOOPBACK       65    /* loopback from composite video output */
```

Set *monochrome* to TRUE to optimize the video digitization when the video signal is known to be monochrome (black and white).

The default value for the video source is CVIDEO1 (color).

INCLUDE FILES

sdl.h

waitForFrame

NAME

waitForFrame – wait for capture of a new frame

SYNOPSIS

```
int waitForFrame
     (
     int rdy             /* wait flag: 0 = not ready, 1 = ready */
     )
```

DESCRIPTION

This function may be used to check the status of a capture operation previously started with **grabFrame().**This is most often used to detect when a full image has been captured so the host program can do processing on the video data. The timeout is approximately 60-80 ms.

RETURNS

int /* Frame status: 0 = no frame (timeout), 1 = match found (grab or no grab) */

INCLUDE FILES

sdl.h

SEE ALSO

waitForSync()

waitForSync

NAME

waitForSync – wait for vertical sync or timeout

SYNOPSIS

`int waitForSync(`void`)`

DESCRIPTION

This function may be used to delay a program until a vertical sync is recognized by the capture engine or video decoder. It can also be used as a check for presence of a valid video input for testing for a timeout error return. The timeout is approximately 60-80 ms.

RETURNS

int /* Vsync status: 0 = no sync (timeout), 1 = Vsync found */

INCLUDE FILES

sdl.h

SEE ALSO

waitForFrame()

writeRGB16

NAME

writeRGB16 – convert and write 16-bit pixels to a file

SYNOPSIS

```
int writeRGB16
    (
    unsigned int pixel,     /* 32-bit word from framebuffer */
    void *fp,               /* FILE pointer */
    int gray,               /* grayscale flag */
    int raw                 /* raw data write flag */
    )
```

DESCRIPTION

This function converts the 32-bit word, *pixel*, into two 16-bit (5-6-5 RGB) pixels, which are wriiten to the file specified by *fp*. Byte swapping is done on *pixel* if required by the CPU architecture. If *gray* is non-zero, the image data is treated as grayscale, rather than color. The *raw* flag switches between binary (if non-zero), or hexadecimal (with a leading "0x") if zero.

RETURNS

int /* 0 = sucess, -1 = error */

EXAMPLE

The following example writes a 16-bit RGB video image from the capture buffer to a file:

```
FILE *fp;
int i, size = 640*480*2;
unsigned int *vbuff = (unsigned int *)malloc(size);

getVideoImage(0, 0, 640, 480, (unsigned char *)vbuff);
fp = fopen("myfile", "w");
size /= 4;   /* convert count from bytes to 32-bit words */
for (i=0; i<size; i++)
    writeRGB16(*vbuff++, fp, 0, 1);
free(vbuff);
```

INCLUDE FILES

sdl.h

SEE ALSO

getVideoImage(), writeRGB32(), writeYUV(), **video capture example programs**

writeRGB32

NAME

writeRGB32 – convert and write 32-bit pixels to a file

SYNOPSIS

```
int writeRGB32
    (
    unsigned int pixel,     /* 32-bit word from framebuffer */
    void *fp,               /* FILE pointer */
    int gray,               /* grayscale flag */
    int raw                 /* raw data write flag */
    )
```

DESCRIPTION

This function converts the 32-bit word, *pixel*, into three 8-bit bytes (8-8-8 RGB), which are wriiten to the file specified by *fp*. Byte swapping is done on *pixel* if required by the CPU architecture. If *gray* is non-zero, the image data is treated as grayscale, rather than color. The *raw* flag switches between binary (if non-zero), or hexadecimal (with a leading "0x") if zero.

RETURNS

int /* 0 = sucess, -1 = error */

EXAMPLE

The following example writes a 32-bit RGB video image from the capture buffer to a file:

```
FILE *fp;
int i, size = 640*480*2;
unsigned int *vbuff = (unsigned int *)malloc(size);

getVideoImage(0, 0, 640, 480, (unsigned char *)vbuff);
fp = fopen("myfile", "w");
size /= 4;  /* convert count from bytes to 32-bit words */
for (i=0; i<size; i++)
    writeRGB32(*vbuff++, fp, 0, 1);
free(vbuff);
```

INCLUDE FILES

sdl.h

SEE ALSO

getVideoImage(), writeRGB16(), writeYUV(), **video capture example programs**

writeYUV

NAME

writeYUV – convert and write 16-bit YUV format pixels to a file

SYNOPSIS

```
int writeYUV
    (
    unsigned int pixel,      /* 32-bit word from framebuffer */
    void *fp,                /* FILE pointer */
    int gray,                /* grayscale flag */
    int raw                  /* raw data write flag */
    )
```

DESCRIPTION

This function converts the 32-bit word, *pixel*, into two 16-bit (YUV) pixels, which are wriiten to the file specified by *fp*. Byte swapping is done on *pixel* if required by the CPU architecture. If *gray* is non-zero, the image data is treated as grayscale, rather than color. The *raw* flag switches between binary (if non-zero), or hexadecimal (with a leading "0x") if zero.

RETURNS

int /* 0 = sucess, -1 = error */

EXAMPLE

The following example writes a YUV video image from the capture buffer to a file:
```
FILE *fp;
int i, size = 640*480*2;
unsigned int *vbuff = (unsigned int *)malloc(size);

getVideoImage(0, 0, 640, 480, (unsigned char *)vbuff);
fp = fopen("myfile", "w");
size /= 4;   /* convert count from bytes to 32-bit words */
for (i=0; i<size; i++)
    writeYUV(*vbuff++, fp, 0, 1);
free(vbuff);
```

INCLUDE FILES

sdl.h

SEE ALSO

***getVideoImage(), writeRGB16(), writeRGB32(),* video capture example programs**

Index